BILKENT UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINERING

EEE-491

Electrical and Electronics Engineering Design I

Spoken Number Recognition Project

PROJECT REPORT

Prepared by GROUP: 02

Pelinsu Acar-21602073

İbrahim Burak Yorulmaz-21703175

Alihan Koçoğlu-21702502

12.01.2021

TABLE OF CONTENTS

1 PROJECT OBJECTIVE AND SPECIFICATIONS

The purpose of this project is to design a spoken number recognition system on Vivado using Basys3 and some analog component. The number will be spoken to the microphone soldered in PCB. The sound signal will be amplified and filtered with the circuit implemented on the PCB. The filter will have 80dB/decade attenuation performance by using cascaded Sallen-Key low-pass filter (LPF) stages. The corner frequency of the LPF will be at 4 KHz. The LPF output signal swing will be at least 3.2V peak to peak. Then using an ADC the sound signal will reach to Basys3. The ADC sample rate will be 16K samples per second and each sampled 12-bit data shall be converted to 16-bit data with leading zeros and stored in a dual port RAM in the FPGA. The size of the RAM shall be capable of storing 16384 samples. There will be a rom to store the hamming window coefficients and the ADC data will be multiplied by these coefficients. Then for each frame 512-point FFT operation will be executed by the IP block of the Vivado design suite. Absolute value of the resulting 32 bit real and 32 bit imaginary output (which is 64 bit) will be calculated and stored in a ram. After that Mel-Frequency Cepstral Coefficients (MFCC) will be calculated and gathered to have a feature vector for the spoken number. The feature vectors will be used for the recognition of a spoken number. After obtaining a feature vector of a spoken number, the new feature vector will be compared with the pre-stored ten feature vectors. The recognition process will be completed by identifying the number with the minimum Euclidean distance between the pre-stored feature vector and the new obtained feature vector. The recognized number will be displayed on a 7-segment display on the Basys-3 FPGA board. For the debug purposes, a transmitter module will be implemented and used to transfer the data to MATLAB. The baud rate of this UART protocol will be 256000.

2 SYSTEM DESCRIPTION

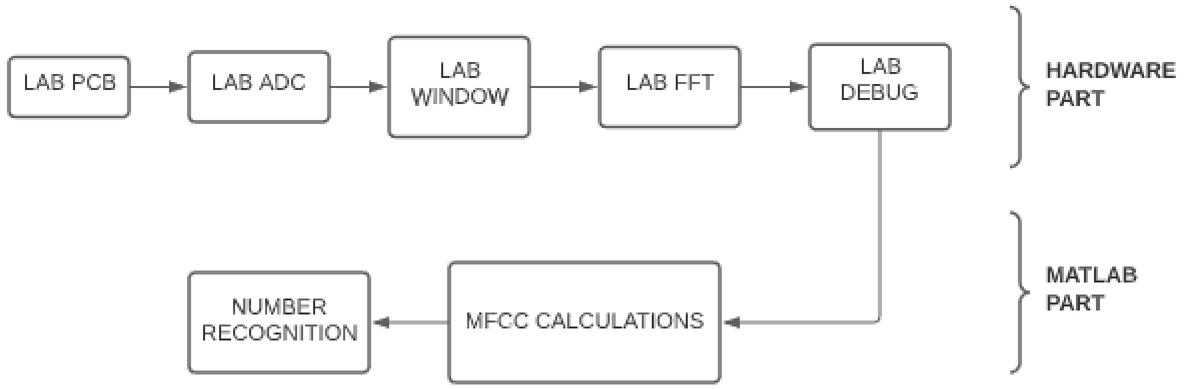


Fig. 1: Block Diagram of the Overall System

In our design, a voice signal is perceived with a microphone on PCB and after the amplification and filtering it is transferred into a dual port ram with the XADC of Basys3. In Vivado, the signal is cut into frames and multiplied by Hanning coefficients. Then for each frame, 512 point FFT operation is executed and using “Lab Debug” the FFT result is sent to MATLAB. Up to that point, all operations are done using PCB and Basys3 and this part constitutes the hardware part of the project. The rest of the process is implemented on the MATLAB because of the time restriction. MATLAB creates the Mel filter bank and by taking the logarithm and DCT of the data, it creates a feature vector for each spoken number. Then calculating the Euclidian distances between feature vectors, the algorithm decides its guess about the number and it is displayed on MATLAB screen.

3 HARDWARE PART

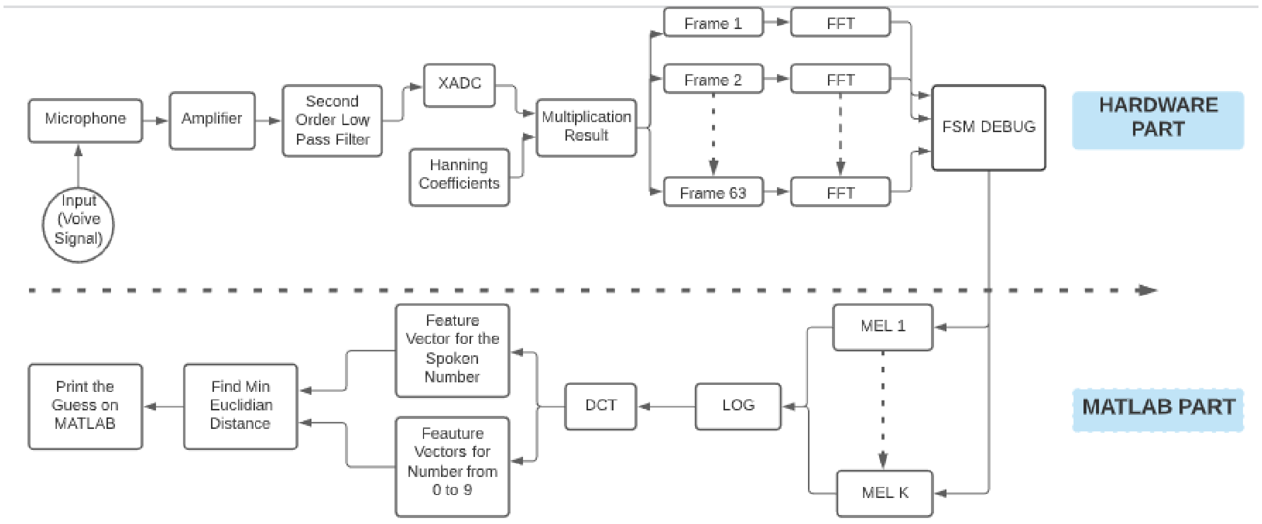


Fig. 2: Block Diagram of the Hardware Part with Interfacing to MATLAB

As mentioned above, the amplifier circuit with low pass Sallen-key filter, internal XADC of Basys3, Hanning window block, FFT window block and FSM debug block constitutes the hardware part of the project. The remaining parts that are Mel-filter, LOG and DCT blocks are implemented on MATLAB.

3.1 Microphone Amplifier with Anti-Aliasing Filter

For this part, a microphone amplifier and Sallen-Key low pass filter circuits are implemented. The amplifier has a single operational amplifier (OPAMP) as the active component. The filter has two OPAMPs as the active components. We used TL082 OPAMP to implement our circuit. The filter has 78 dB/decade attenuation performance by using cascaded Sallen-Key low-pass filter (LPF) stages. The corner frequency of the LPF is at 4 kHz. The PCB use single power supply and reference bias is provided by unity gain OPAMP for high power supply rejection ratio (PSRR). The input varies between -50 and +50 mV and the LPF output signal swing should be at most 1V peak to peak since the internal ADC of basys3 does not accept more than 1 V.

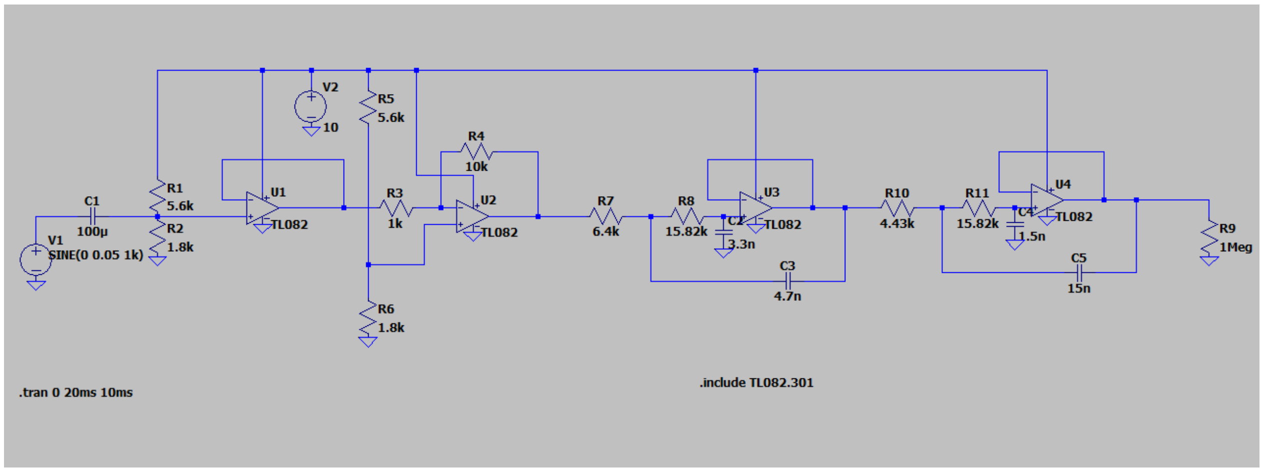


Fig 3: LTspice Schematic of the circuit design

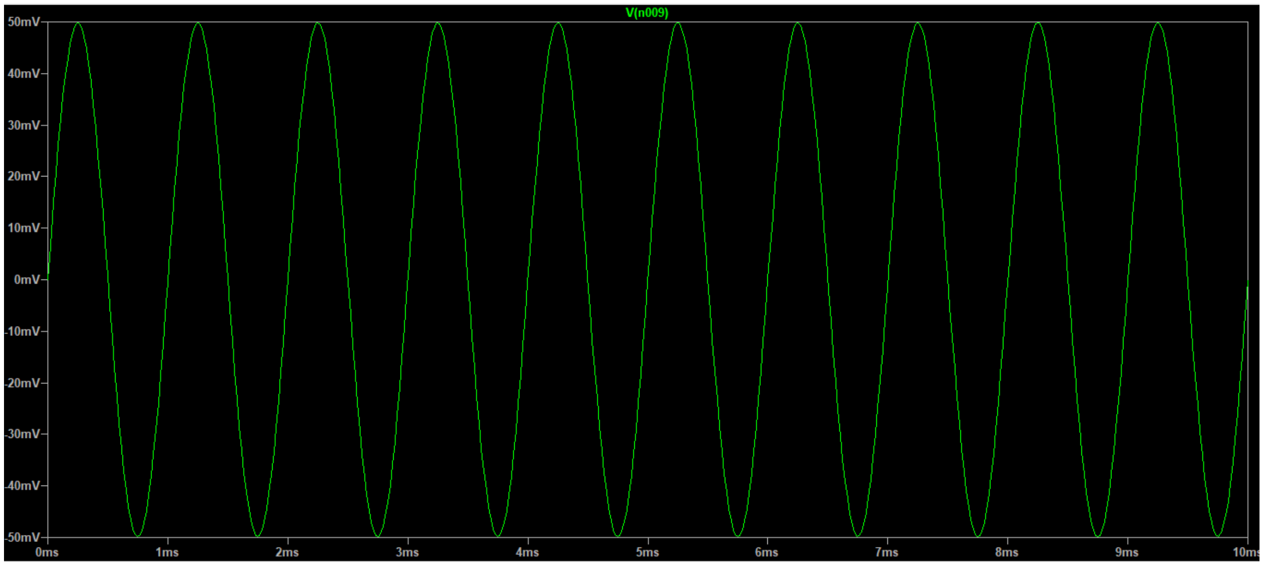


Fig 4: 100 mV$_{P-P}$ Input Signal at 1 kHz

In the first stage of the circuit, using a voltage follower biasing method, we give an offset voltage to the input signal. This method is useful for reducing power consumption in a circuit, and for giving more accurate offset values.

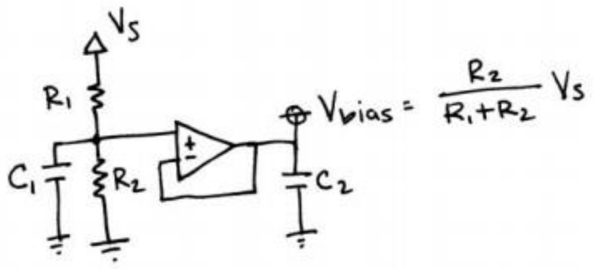
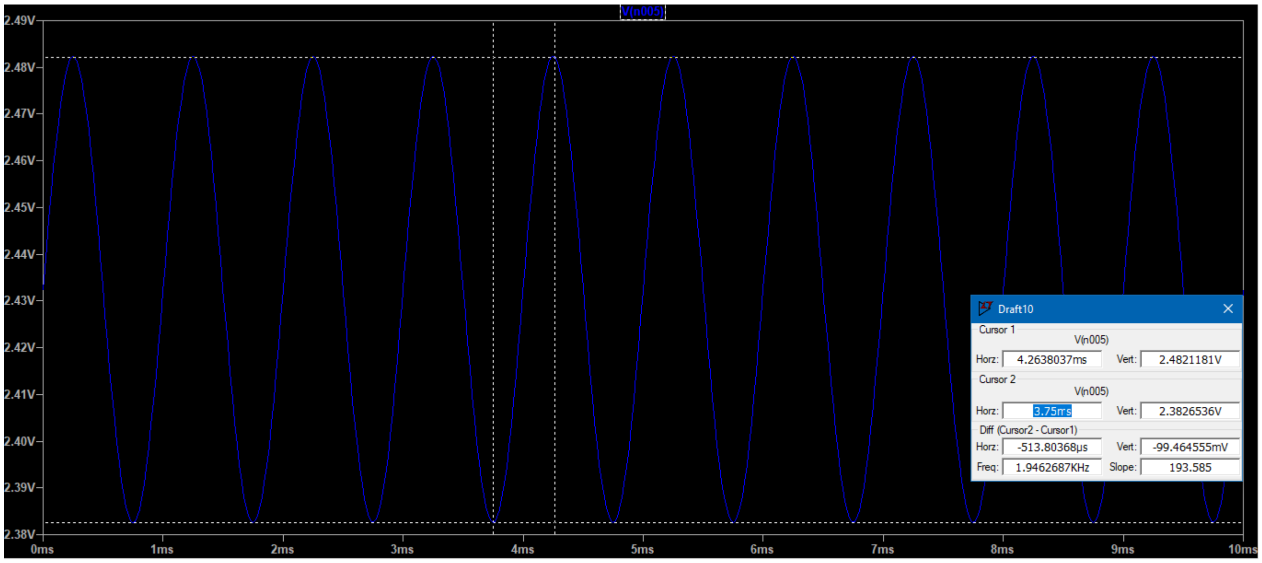


Fig 5: Voltage Follower Biasing Circuit [1]

From figure 3 and 5, we can calculate our biasing voltage. Since Vs = 10 V, R1 = 5.6K and R2 = 1.8K, Vbias = $10\ x\ \frac{1.8}{5.6+1.8} = 2.43\ V$. This calculation can be confirmed from LTspice schematic of the first stage as well. The figure is below.



```
Draft10
Cursor 1
                    V(n005)
Horz: 4.2638037ms        Vert: 2.4821181V
Cursor 2
                    V(n005)
Horz: 3.75ms             Vert: 2.3826536V
Diff (Cursor2 - Cursor1)
Horz: -513.80368µs       Vert: -99.464555mV
Freq: 1.9462687KHz       Slope: 193.585
```

Fig 6: Output of the First Stage with 2.43 V offset

For the amplifier stage (second stage of the circuit), the AC coupled inverting amplifier is used. The figure for the amplifier stage is below.

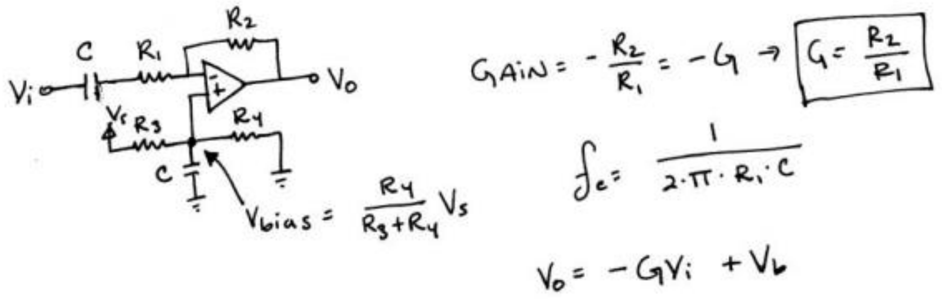
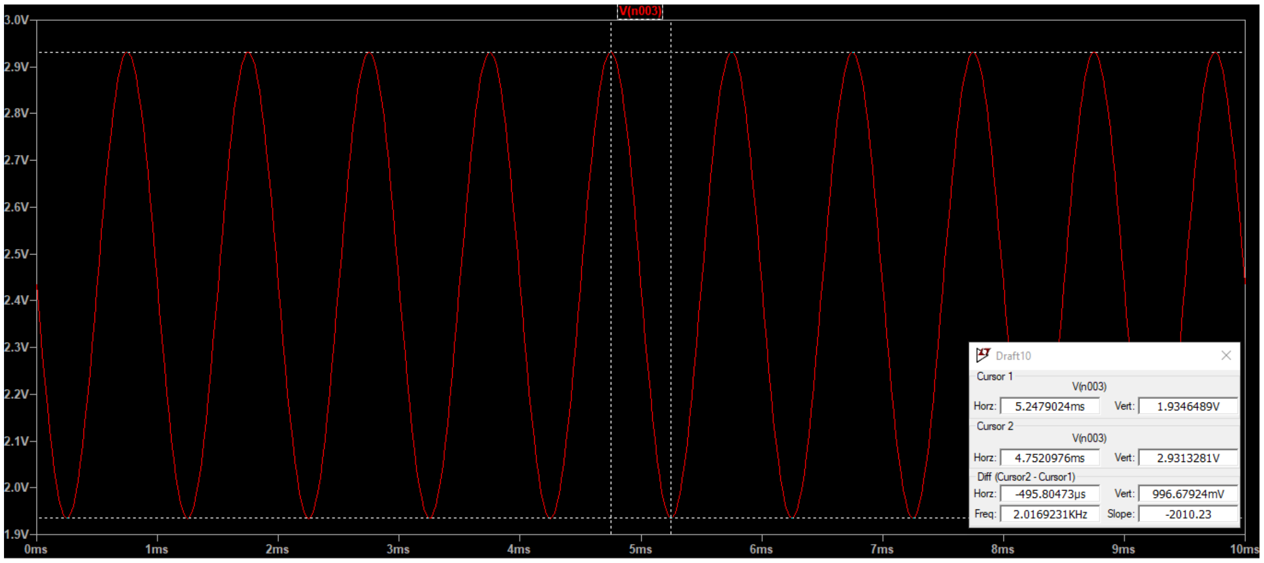


Fig 7: AC Coupled Inverting Amplifier Circuit [1]

Choosing R1 as 1K and R2 as 10K, we reach the necessary gain which is 10 dB for our circuit since the output should not exceed 1 $V_{p-p.}$

Gain = $-\frac{10}{1} = 10$

Vo(max) = $(-10)x(-0.05) + 2.43 = 2.93\ V$ and Vo(min) = $(-10)x(0.05) + 2.43 = 1.93\ V$



```
Cursor 1
                V(n003)
Horz: 5.2479024ms     Vert: 1.9346489V
Cursor 2
                V(n003)
Horz: 4.7520976ms     Vert: 2.9313281V
Diff (Cursor2 - Cursor1)
Horz: -495.80473µs    Vert: 996.67924mV
Freq: 2.0169231KHz    Slope: -2010.23
```

Fig 8: Output of the Circuit (Amplified Signal with 997 mV$_{P\text{-}P}$)

From figure 8, it is seen that the output signal changes from 1.93 V to 2.93 V with 9.97 V$_{p\text{-}p}$. Therefore, the calculations are tested with LTspice simulations as well and the amplifier part of the circuit is completed.

For the filter stage, we use two cascaded Sallen-Key low pass filter. The circuit schematic for one stage is below.

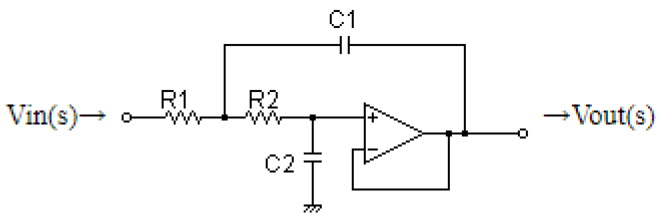


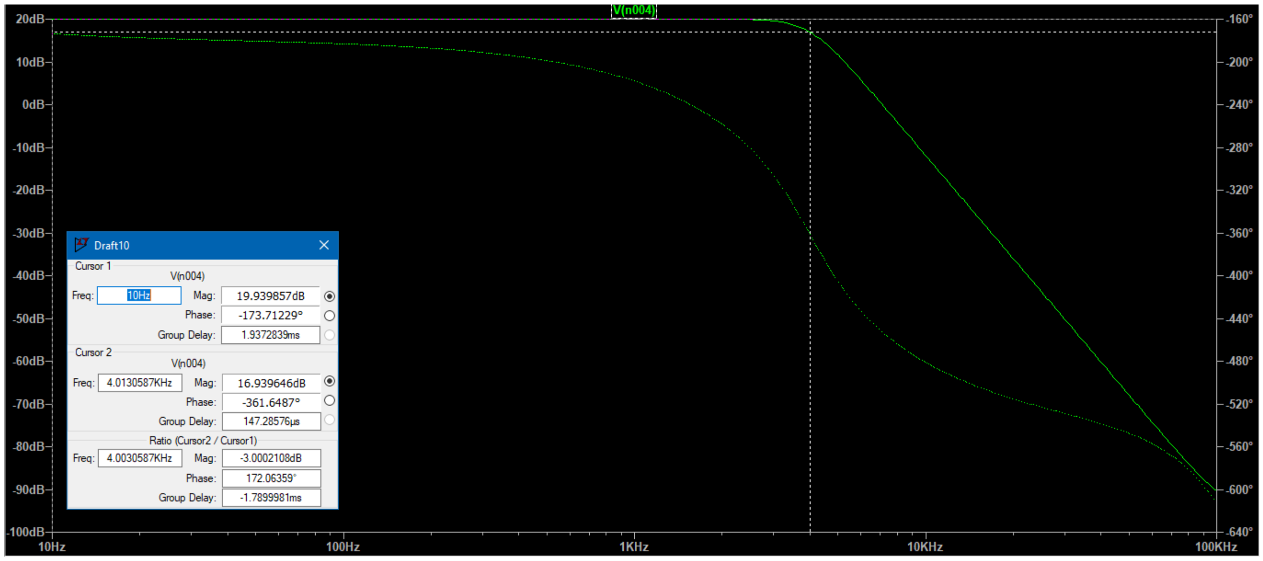
Fig 9: Sallen Key Low Pass Filter Circuit [2]

Cutoff Frequency: fc = $\frac{1}{2\pi\sqrt{R1xR2xC1xC2}}$

From figure 3 and 9, fc values can be calculated as follows:

fc (1. stage) = $\frac{1}{2\pi\sqrt{6.4Kx15.82Kx3.3nx4.7n}} = 4\ kHz$

fc (2. Stage) = $\frac{1}{2\pi\sqrt{4.43Kx15.82Kx1.5nx15n}} = 4\ kHz$

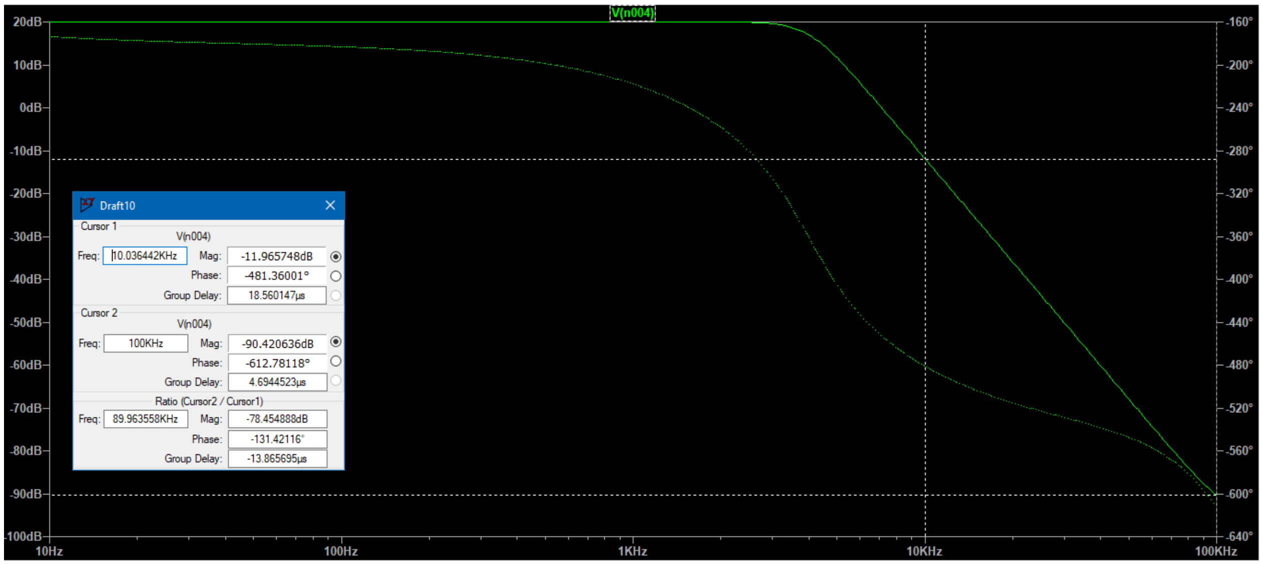


```
Draft10
Cursor 1
                V(n004)
Freq: 10Hz             Mag: 19.939857dB
                     Phase: -173.71229°
               Group Delay: 1.9372839ms
Cursor 2
                V(n004)
Freq: 4.0130587KHz     Mag: 16.939646dB
                     Phase: -361.6487°
               Group Delay: 147.28576µs
        Ratio (Cursor2 / Cursor1)
Freq: 4.0030587KHz     Mag: -3.0002108dB
                     Phase: 172.06359°
               Group Delay: -1.7899981ms
```

Fig 10: AC Analysis of Circuit Showing the -3 dB Attenuation

From figure 10, -3dB attenuation is at 4 kHz so we succeed to limit the bandwidth of the output signal.

Since we use two stage low pass filter, we expect to obtain a total of 80 dB/decade, 40 dB/decade from each stage and in figure 11, it is seen that there is sharp decrease after 2 kHz and the filter has 78 dB/decade attenuation performance.



```
Draft10
Cursor 1
                V(n004)
Freq: 10.036442KHz     Mag: -11.965748dB
                     Phase: -481.36001°
               Group Delay: 18.560147µs
Cursor 2
                V(n004)
Freq: 100KHz           Mag: -90.420636dB
                     Phase: -612.78118°
               Group Delay: 4.6944523µs
        Ratio (Cursor2 / Cursor1)
Freq: 89.963558KHz     Mag: -78.454888dB
                     Phase: -131.42116°
               Group Delay: -13.865695µs
```

Fig 11: AC Analysis of Circuit showing the 78 Db/decade Attenuation Performance

After the LTspice simulations, we use Proteus for the PCB design. The PCB layout schematic is below.

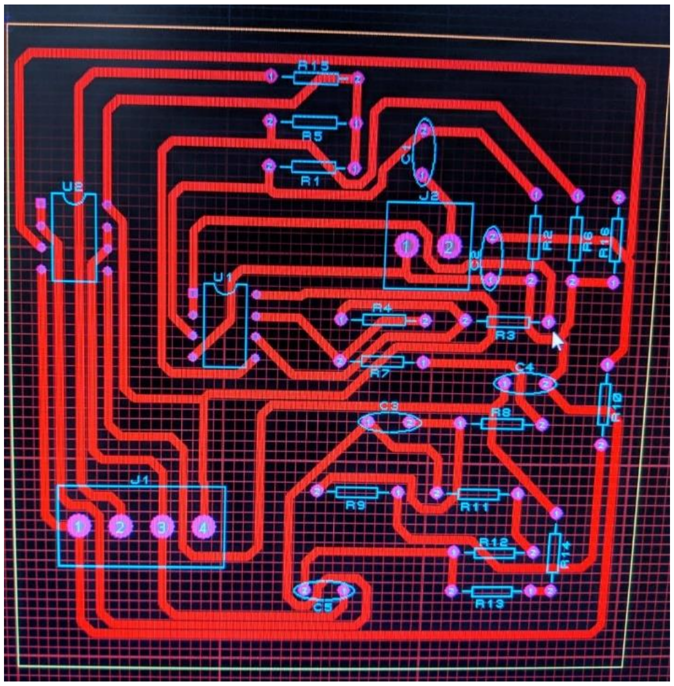


Fig 12: PCB Layout

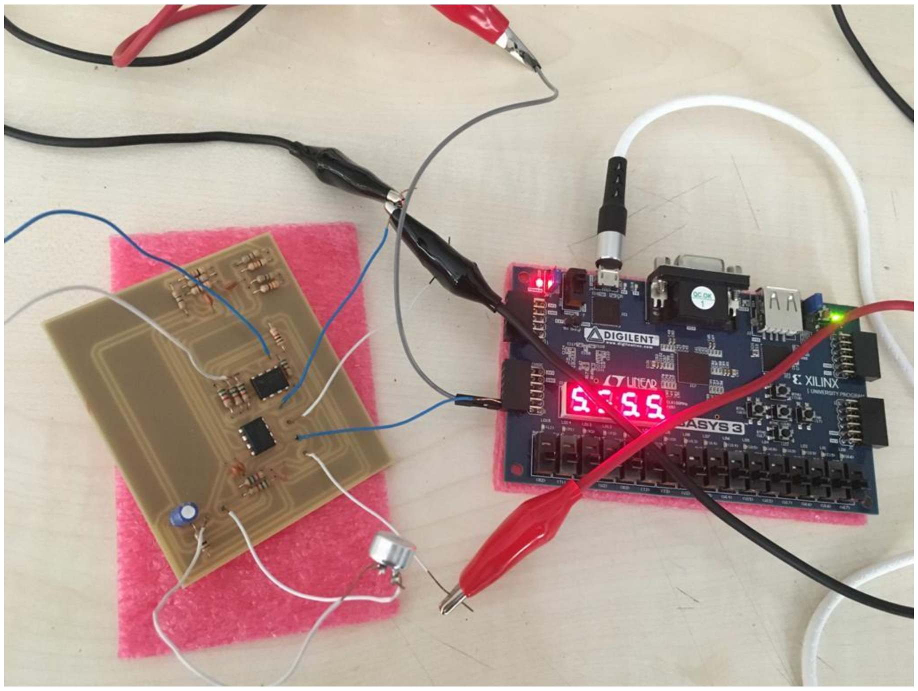


Fig 13: Soldered PCB Integrated to Basys3

Output peak-to-peak voltage levels at 50 Hz, 1 KHz, 10 KHz and 100 KHz can be seen below. From figure 14 and 15, it is seen that the amplifier circuit has approximately 10 dB gain at 50 Hz and 1 KHz since the input is 0.2 Vpp and the outputs are 1.92 Vpp, 1.84 Vpp respectively. Furthermore, from figure 16 to 17, there is a sharp decrease on the voltage peak to peak value (1.28 Vpp to 0.16 Vpp) meaning that our circuit can filter higher frequencies successfully.

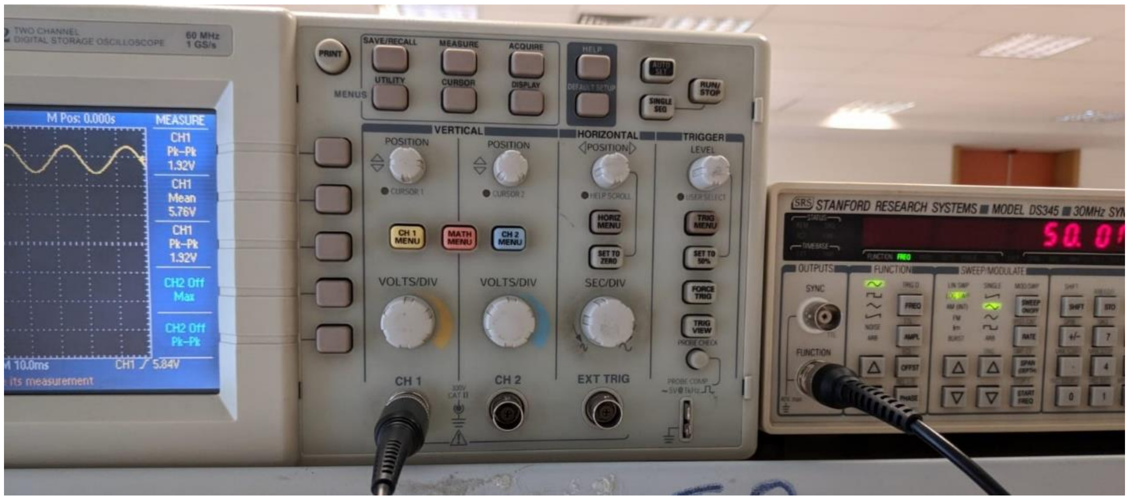


Fig 14: Output Signal with 0.15 Vpp Sinusoidal Input at 50 Hz

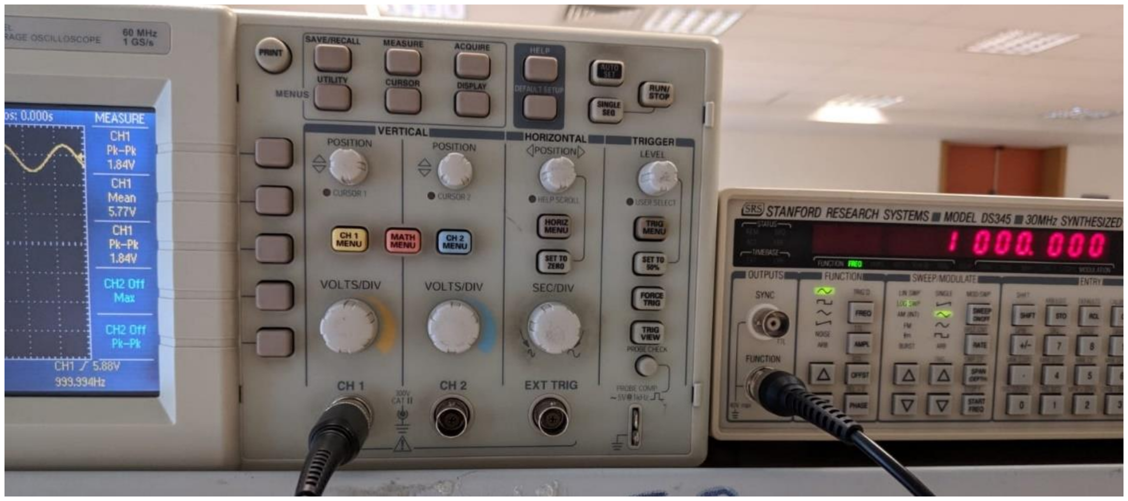


Fig 15: Output Signal with 0.15 Vpp Sinusoidal Input at 1 kHz

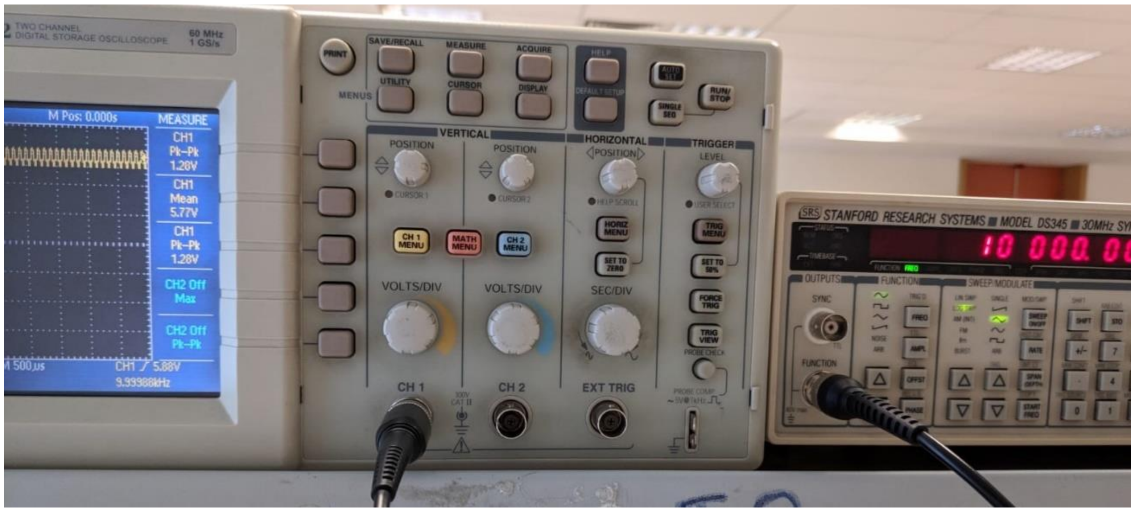


Fig 16: Output Signal with 0.15 Vpp Sinusoidal Input at 10 kHz

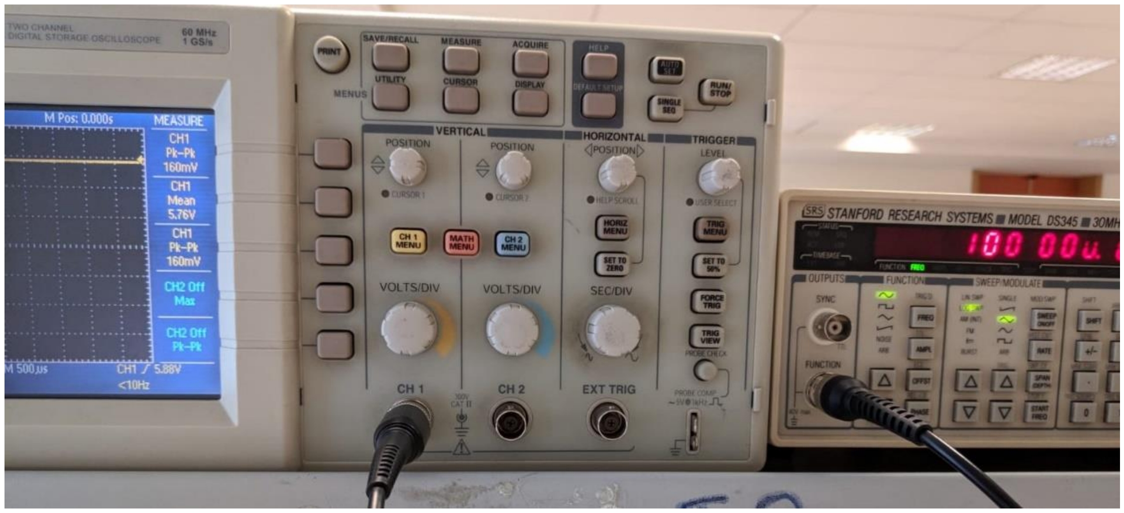


Fig 17: Output Signal with 0.15 Vpp Sinusoidal Input at 100 kHz

After the oscilloscope verification, we integrate the PCB to our ADC and debug system to check the result from MATLAB as well. We give speech and sinusoidal signal sounds from the mobile phone at different frequencies to the microphone on PCB and plot the output in MATLAB. The output graphs can be seen below.

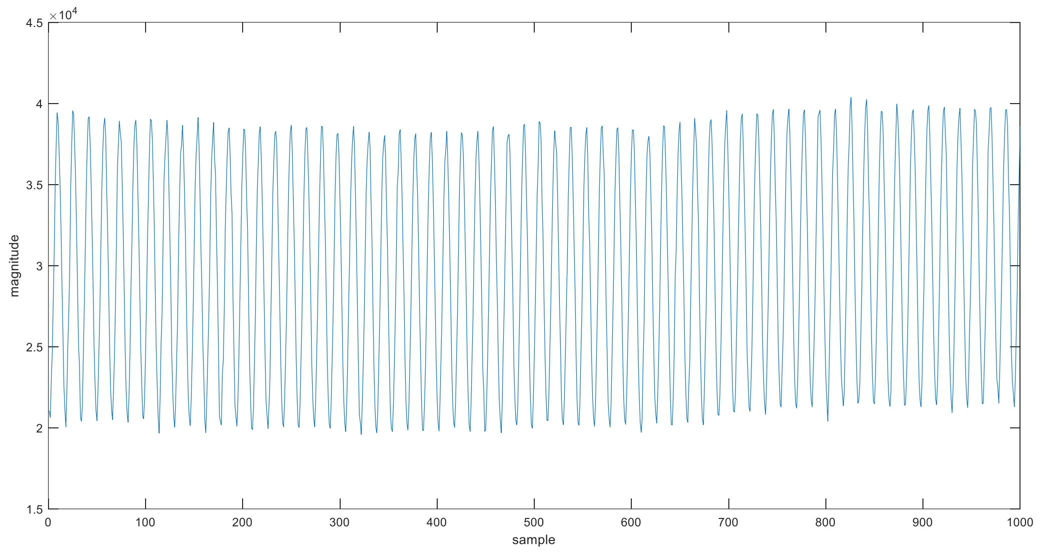


Fig 18: Output Signal with Input Signal of Sine Wave Sound at 1 kHz

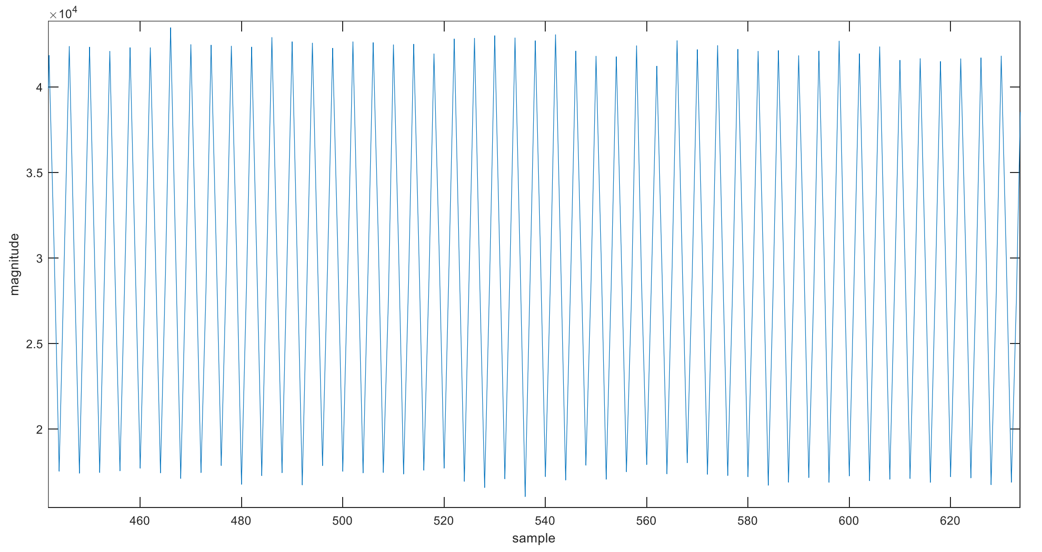


Fig 19: Output Signal with Input Signal of Sine Wave Sound at 4 kHz

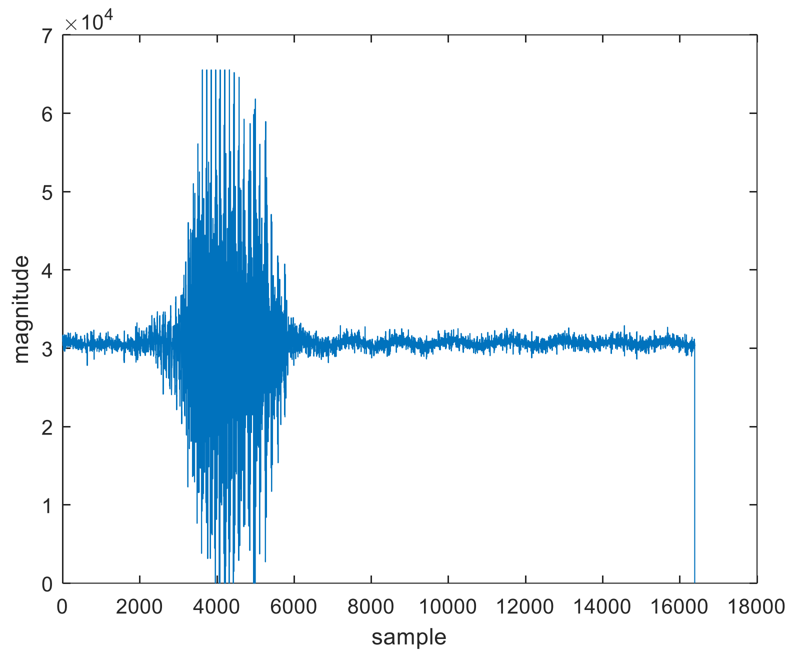


Fig 20: Output Signal with Input Signal of Speech “One”

3.2 A/D Converter Module

For the lab ADC assignment, the Basys3's XADC port (internal adc) is used. The behavior is as follows:

- Voltage levels change from 0 to 1 Volt, obtained from the JXADC header [3].
- Basys3’s XADC Pmod connector has 4 differential analog pairs. The corresponding XADC channels are 6, 7, 14, and 15. The pinout of the Pmod Header is below [3].

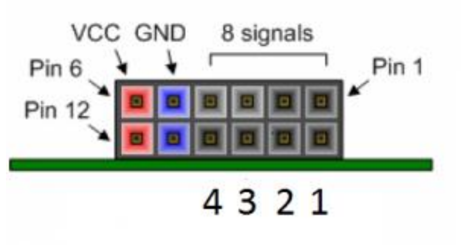


Fig 21: Pmod ports; front view as loaded on PCB [3]

- The Basys3 schematic showing the internal connections of the signals is below.

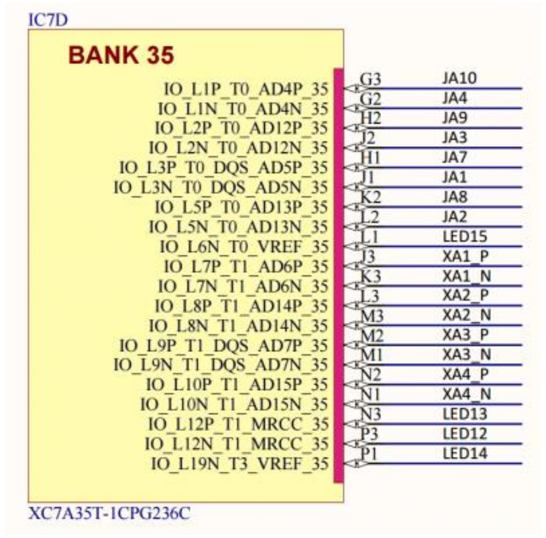


Fig 22: Basys3 Schematic of the internal connections [3]

- From the schematic we can see XA1 is connected to XADC channel 6, XA2 is connected to XADC channel 14, XA3 is connected to XADC channel 7, and XA4 is connected to XADC channel 15 [3].

For this demo, the AD6 pins are used on the JXADC header. We hooked up a signal generator to our pins. The block diagram of the internal ADC is below.

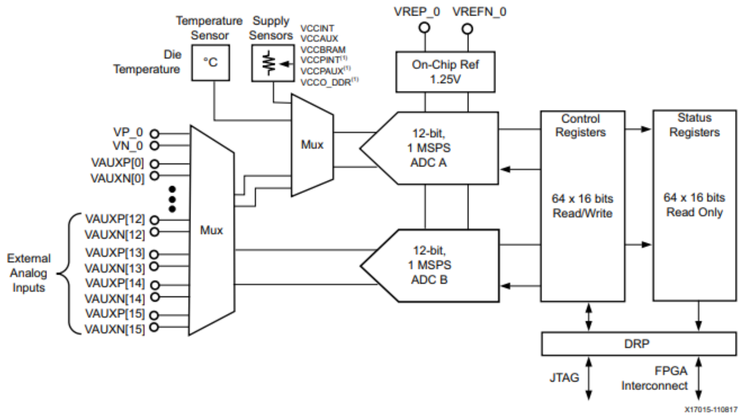
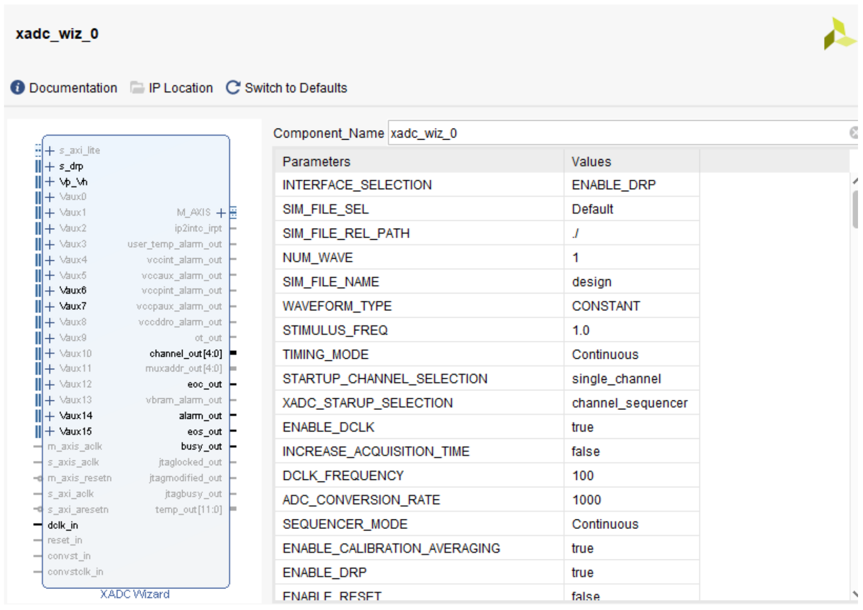


Fig 23: XADC Block Diagram [4]

Since XADC is a built in device, we do not have to write a sampler block. Specifying the sampler frequency of the internal ADC using the IP core is enough for the controlling purpose. The detail of these configurations can be seen in figure 24 below.



Component_Name xadc_wiz_0

Parameters	Values
INTERFACE_SELECTION	ENABLE_DRP
SIM_FILE_SEL	Default
SIM_FILE_REL_PATH	./
NUM_WAVE	1
SIM_FILE_NAME	design
WAVEFORM_TYPE	CONSTANT
STIMULUS_FREQ	1.0
TIMING_MODE	Continuous
STARTUP_CHANNEL_SELECTION	single_channel
XADC_STARUP_SELECTION	channel_sequencer
ENABLE_DCLK	true
INCREASE_ACQUISITION_TIME	false
DCLK_FREQUENCY	100
ADC_CONVERSION_RATE	1000
SEQUENCER_MODE	Continuous
ENABLE_CALIBRATION_AVERAGING	true
ENABLE_DRP	true
ENABLE_RESET	false

Fig 24: XADC IP Core

In "adc_to_ram" module, the data from the internal adc is written to our dual port ram and since ADC sample rate shall be at 16K samples per second, a count signal which function as mod $(\frac{100\ MHz}{16000}) = 6250$ is used. Each sampled 12-bit data should be converted to 16-bit data with leading zeros and stored in a dual port RAM which is capable of storing 16384 samples in the FPGA but since the internal adc of the basys3 gives 16-bit data in 12-bit resolution, adding zero process is skipped. Then using "uart" and "fsm_debug" modules, adc data is sent to MATLAB as it is done in LAB_DEBUG.

The block diagram of our LAB_ADC design is below:

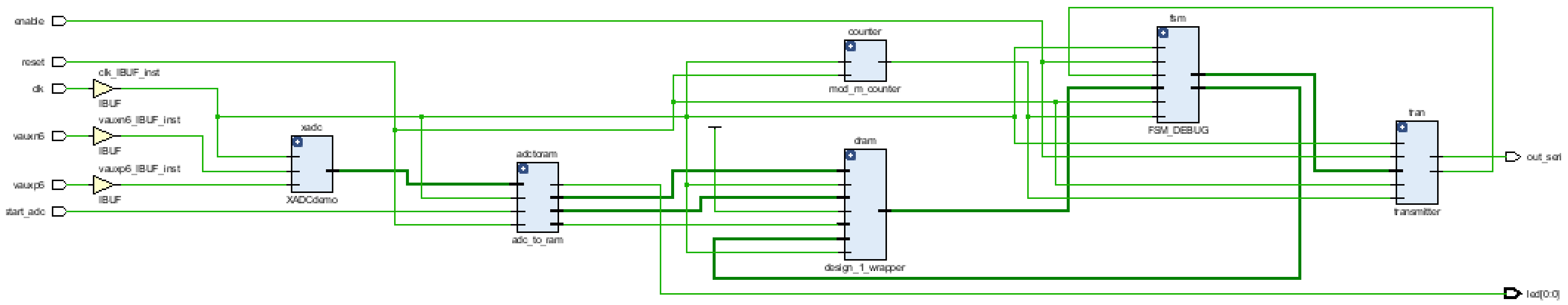


Fig 25: LAB_ADC Elaborated Design Schematic

As it is seen from figure 25, "xadc" module represents the internal adc of the basys3. In that module, the adc data is received and given to "adc_to_ram" module to be stored in our dual port ram ("adc_to_ram" module sends the data to "design_1_wrapper" module which is our dual port ram). The rest is the same with LAB_DEBUG. The only difference is that “fsm_debug” module receives the data from dual port ram which is filled the data in XADC. In LAB_DEBUG, we filled our dual port ram manually for testing purpose.

Since test bench simulation is not possible for this stage, we test our design on MATLAB. We give both DC voltage and sinusoidal signals at different frequencies and amplitudes.

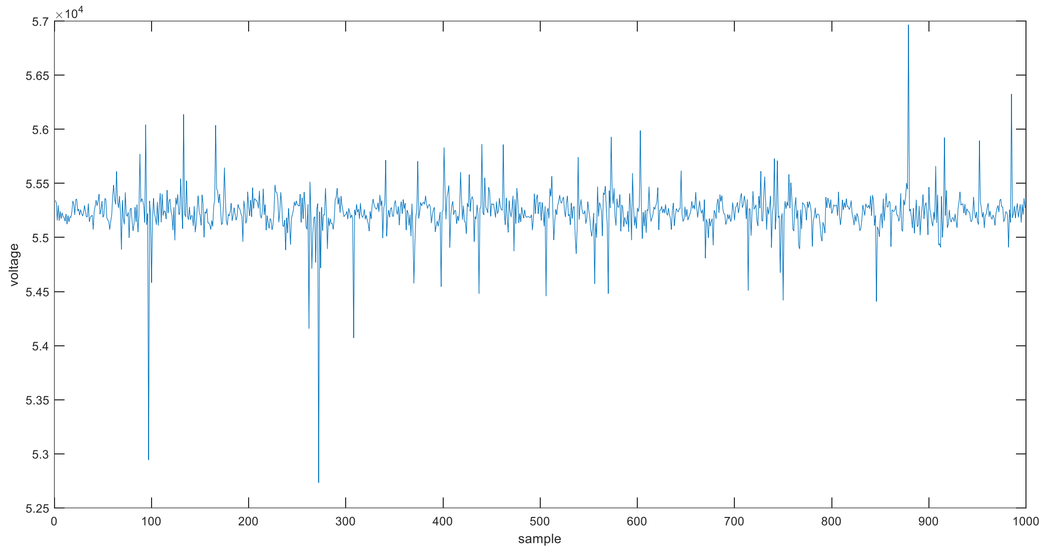


Fig 26: MATLAB Graph Output When 0.7 V DC Voltage Input Applied

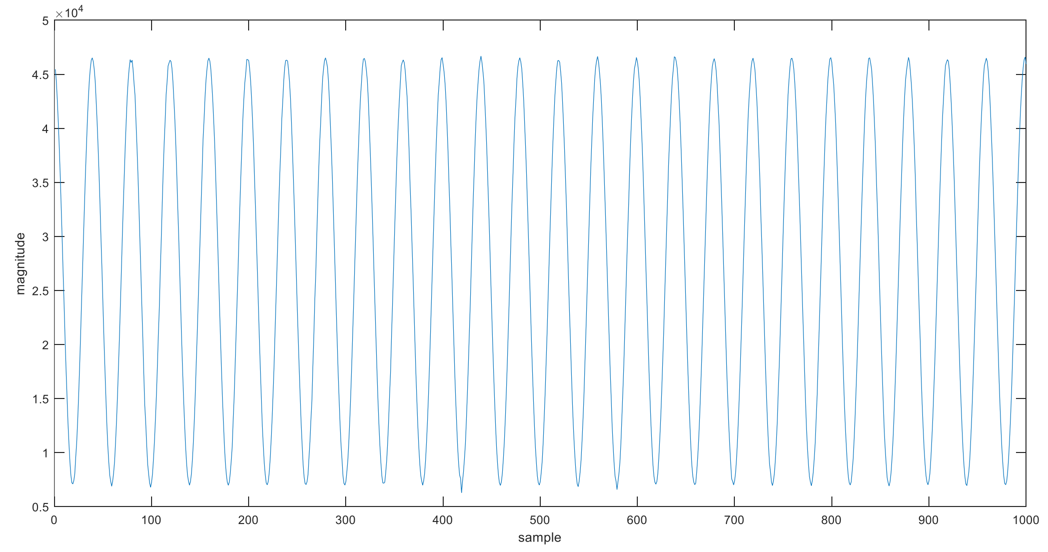


Fig 27: MATLAB Graph Output When 0.5 Vpp Sinusoidal Analog Input at 400 Hz Applied

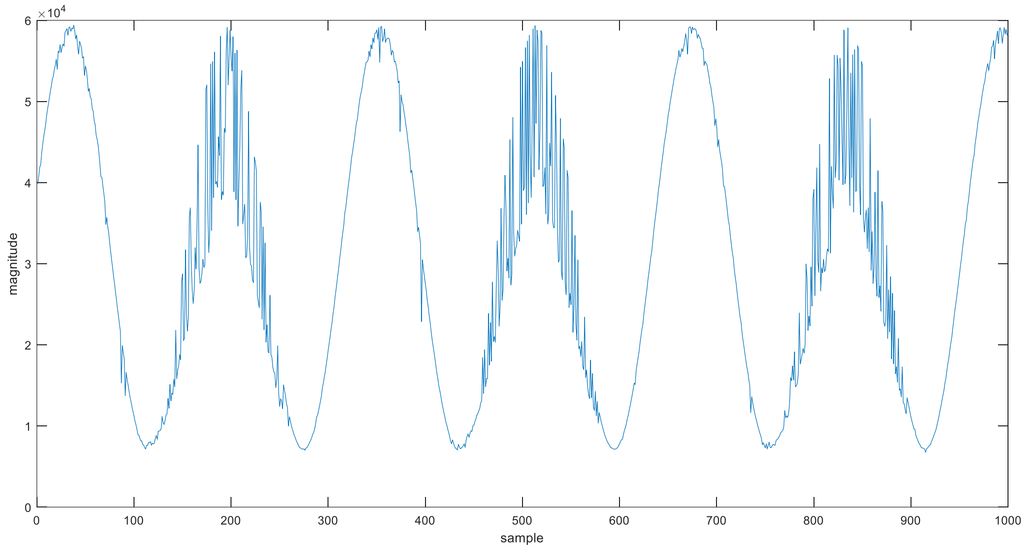


Fig 28: MATLAB Graph Output When 0.7 Vpp Sinusoidal Analog Input at 100 Hz Applied

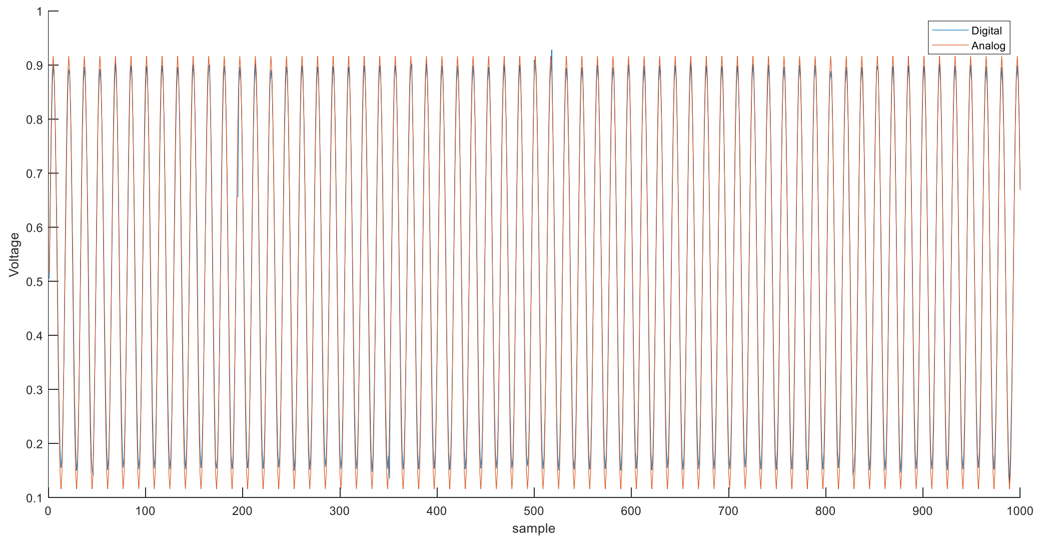
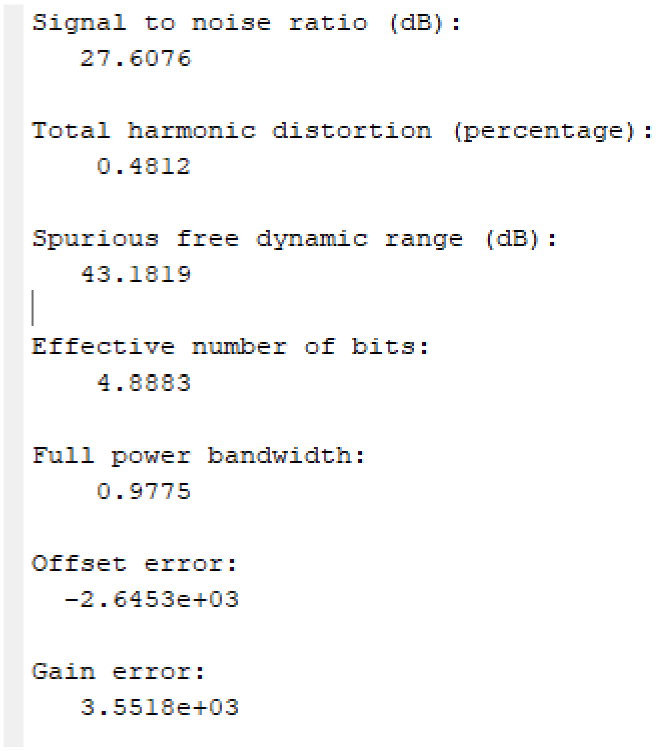


Fig 29: MATLAB Graph Showing Both Digital Output and Analog 0.7 Vpp Sinusoidal Input at 1 kHz

Lastly, we calculate the performance of our system using differential non-linearity (DNL), integral non-linearity (INL), effective number of bits (ENOB), full power bandwidth, gain error, offset error, signal to noise ratio (SNR), spurious free dynamic range (SFDR), total harmonic distortion (THD). The calculation result can be seen below.



```
Signal to noise ratio (dB):
   27.6076

Total harmonic distortion (percentage):
    0.4812

Spurious free dynamic range (dB):
   43.1819

Effective number of bits:
    4.8883

Full power bandwidth:
    0.9775

Offset error:
  -2.6453e+03

Gain error:
   3.5518e+03
```

Fig 30: MATLAB Performance Measurement of the System

For the signal to noise ratio, a ratio of 25-40dB is considered as good so we have 26.6 dB. The total harmonic distortion of the system is %0.4812 which is very low and acceptable. The DNL and INL graphs can be seen below.

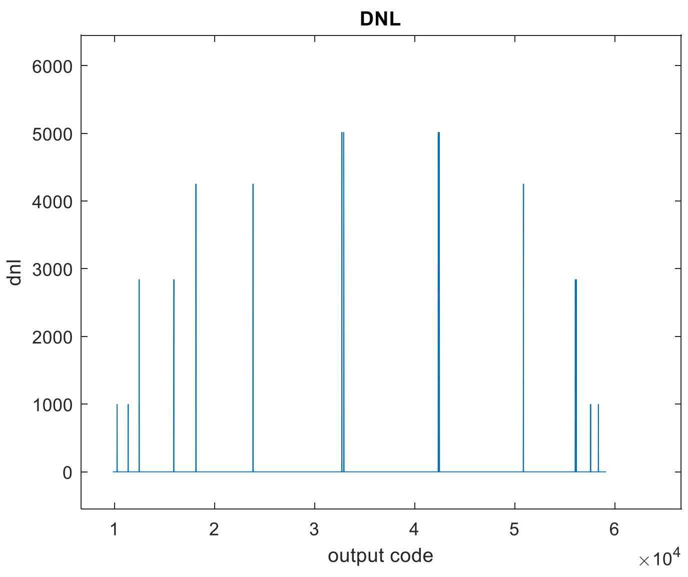


Fig 31: MATLAB Graph Showing ADC's DNL Performance

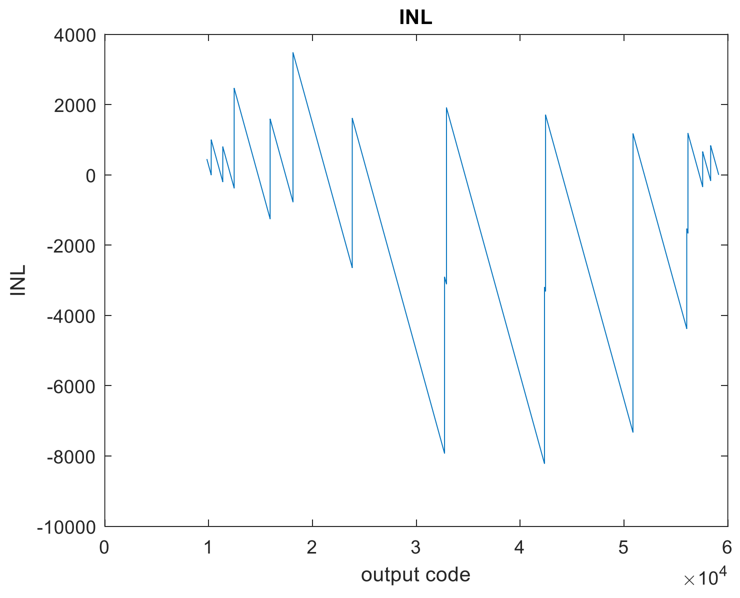


Fig 32: MATLAB Graph Showing ADC's INL Performance

3.3 FPGA Debug Block

First, a transmitter part of the UART is implemented and tested. This part loads the parallel data when enable is 1, then shifts out as serial according to baud rate. The baud rate generator is used to generate a sampling signal. The sampling signal works as an enable ticks instead of a clock not to create a new clock domain. The Basys3 board has a clock rate of 100 MHz. Since we use 9600 bps as baud rate, the counter should function as mod-10417 ($\frac{100\ MHz}{9600}$) so that every 10417 clock cycles, enable ticks becomes one for one-clock-cycle.

To trig the UART component eight times in order to transmit a 64-bit data, FSM_DEBUG module that reads multiple 64-bit data from a dual-port RAM inside the FPGA is implemented. When the transmitter module finishes the process of 8-bit data, it triggers the FSM_DEBUG module to receive another 8-bit data. In FSM_DEBUG, address of the dual port ram is given as data and the ram is filled with 1 to 2^14 x 64 respectively so that transmitter part can be tested. Therefore, FPGA Debug consists of four different blocks as it is seen below.

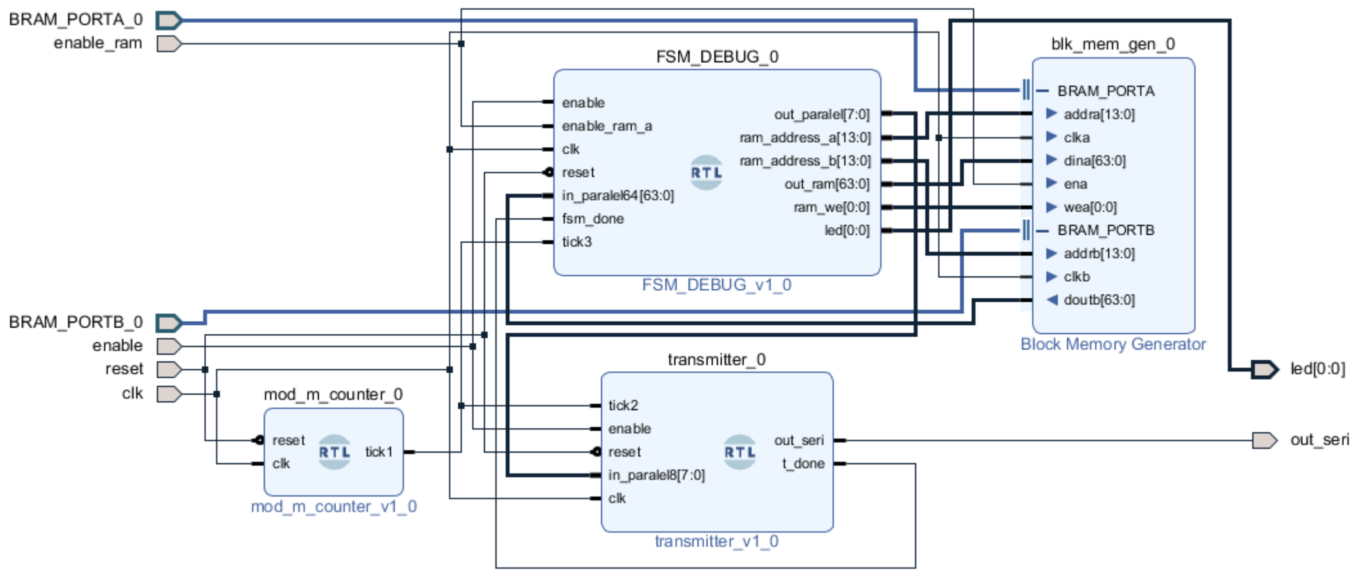
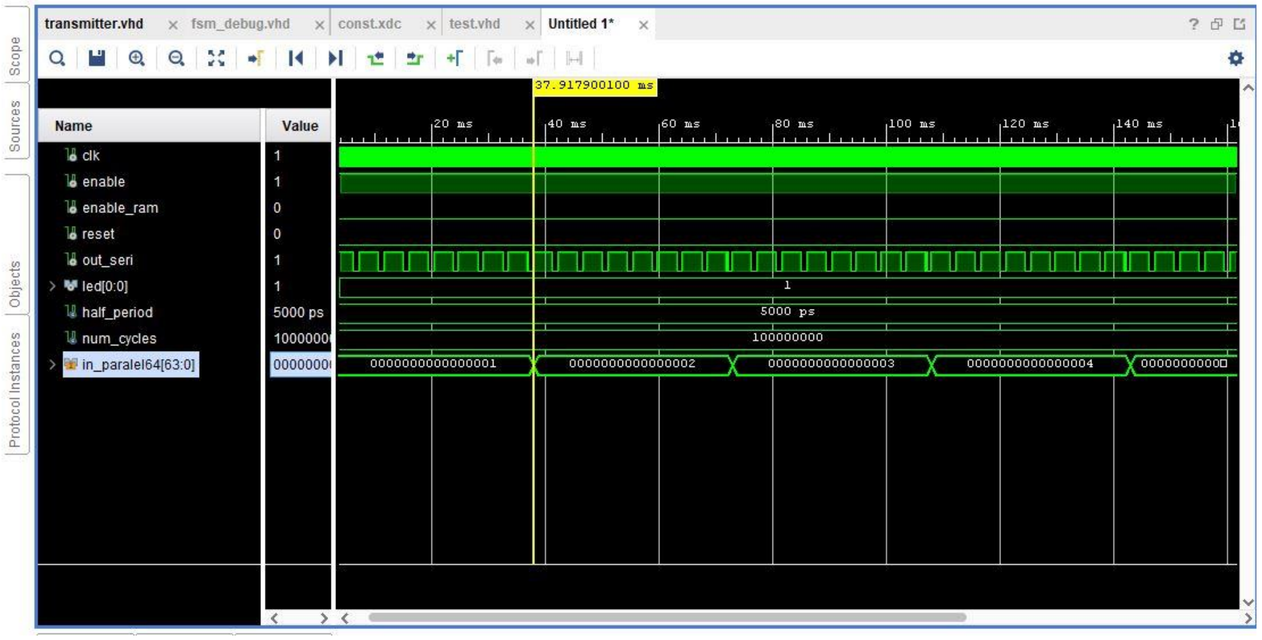


Fig 33: Block Design Schematic of Lab Debug Assignment

In the test bench simulation result," in_paralel64" represents the data in the dual port ram meaning that it is given to the transmitter in 8 bit chunks and "out_seri" represents the serial output of the transmitter.



Name	Value
clk	1
enable	1
enable_ram	0
reset	0
out_seri	1
led[0:0]	1
half_period	5000 ps
num_cycles	1000000(
in_paralel64[63:0]	0000000(

```
1
5000 ps
100000000
0000000000000001  0000000000000002  0000000000000003  0000000000000004  0000000000
```

Fig 34: Test Bench Simulation Result of Lab Debug Assignment

In figure 34, it is seen that "in_paralel64" increases one by one and it confirms that the address values are given as data.

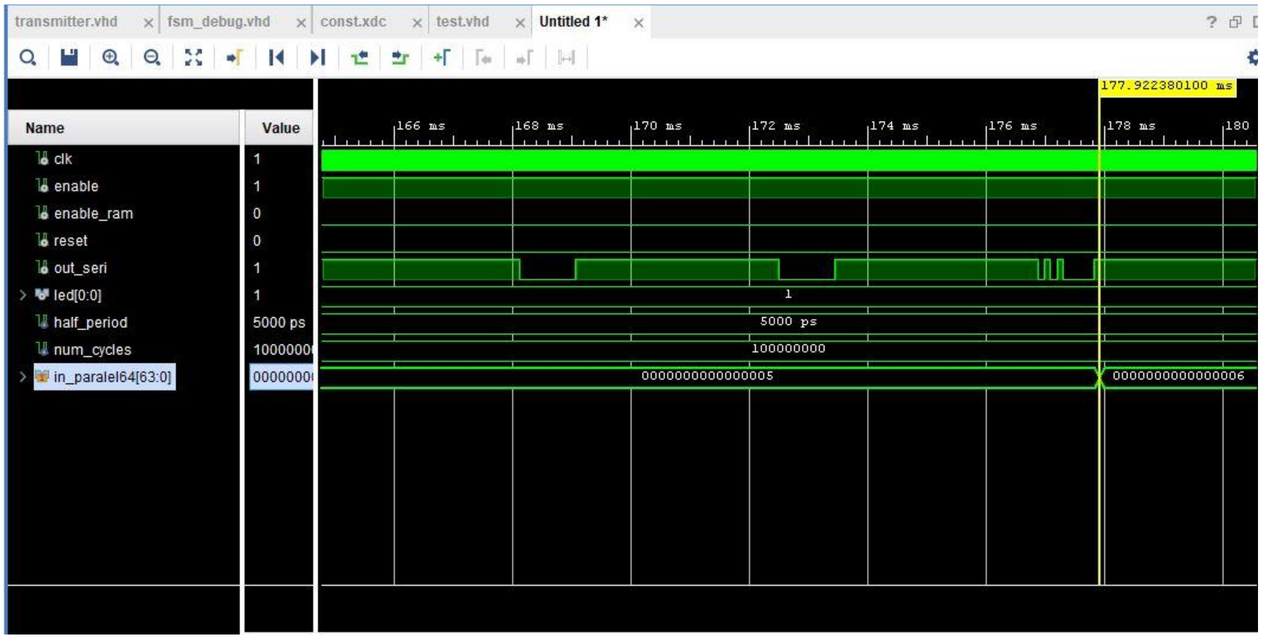


Fig 35: Test Bench Simulation Result of Lab Debug Assignment (Zoomed)

In Figure 35, looking at the zoomed output result, when 64-bit data passes from 5 to 6, the change in the last 8-bit part of “out_seri” confirms this transition.

Lastly, the design is tested on Matlab with the same dual port ram data used in test bench simulation.

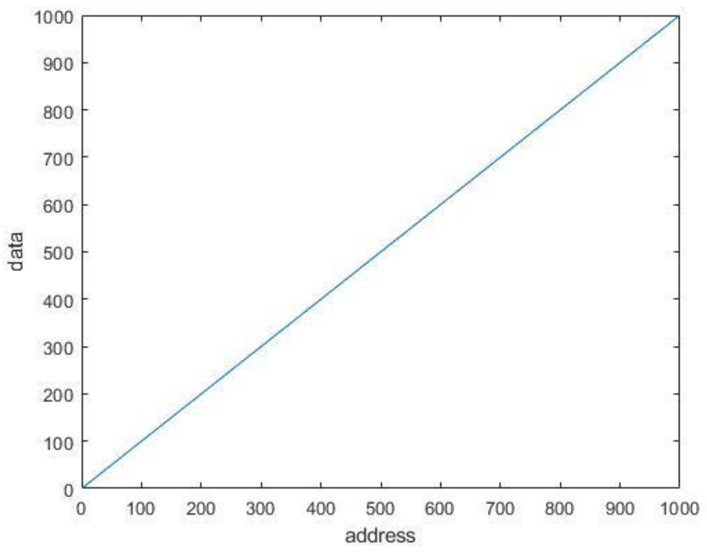


Fig 36: MATLAB graph of the transferred 2^14 x 64-bit data in a dual-port RAM

Since the address and the data of the dual port ram are same and increase linearly, the MATLAB graph should be linear as well and from figure 36, the result meets the expectations of the design and proves that it works correctly.

3.4 FPGA Hamming Block

In Lab-WINDOW, we upload the hamming window coefficients we calculate from MATLAB to the "hann_rom" in Vivado by creating a coe file. The coefficients can be found in Appendix A. In the module "adc_to_ram", we obtain a 32-bit result by multiplying the coefficient coming from this "hann_ rom" with the 16-bit value we get from the internal ADC for each frame. This result is truncated down to 16 bits and stored in a second dual port ram called "hannram". At the end of each frame (when this process is repeated 512 times), "adc_to_ram" module triggers the "fsm_debug" module to send the 16-bit multiplication result to the transmitter as 8-bit data in 2 times. When the "fsm_debug" module completes this process for 512 samples, it again triggers the "adc_to_ram" module to initialize the second frame. So these two modules work by triggering each other, respectively.

Since each frame contains 512 x 16-bit data and in our dual port ram that contains ADC data has 2^14 x 16-bit data, there should be N = $\frac{2^{14}}{512}$ = 32 frames. But there is an overlapping set to %50 of the previous frame. Therefore this process is activated as many times as the number of overlapped frames which is 2N-1 = 63.

The Lab-WINDOW block has two status signal outputs: "frame_done" and "all_frames_done". "frame_done" signal is used to indicate that one frame data is processed and written to the dual port RAM and "all_frames_done" signal output is used to indicate that all the overlapped frames are processed as it is explained above.

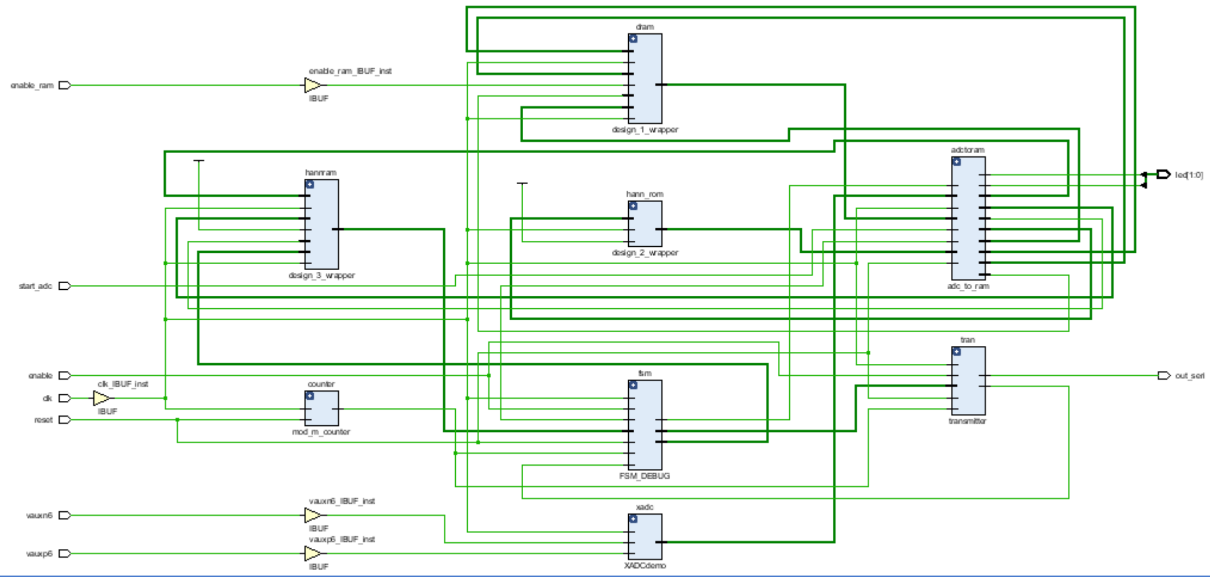
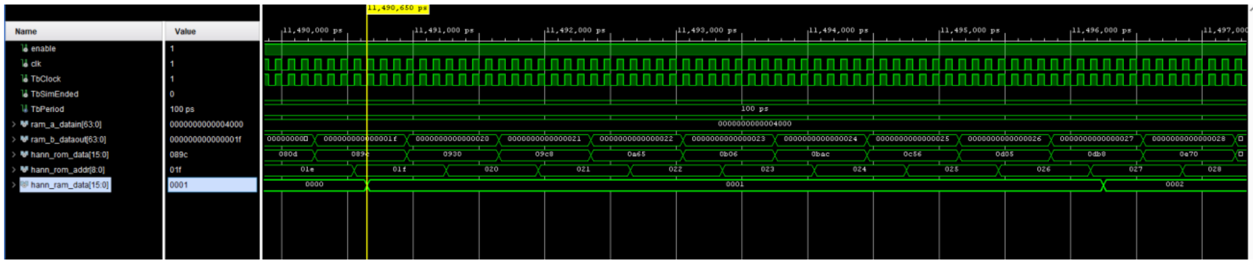


Fig 37: Elaborated Design Schematic of Lab Window

In figure 37, all modules and their connections can be seen. As it is explained, "design_1_wrapper" ("xadc" data) and "hann_rom" sends their data to "adc_to_ram" and the multiplication result is stored in "design_3_wrapper" ("hannram"). Then using "fsm_debug" and "transmitter" modules, the data inside "hannram" is sent to MATLAB.



Name	Value
enable	1
clk	1
TbClock	1
TbSimEnded	0
TbPeriod	100 ps
ram_a_datain[63:0]	0000000000004000
ram_b_dataout[63:0]	000000000000001f
hann_rom_data[15:0]	089c
hann_rom_addr[8:0]	01f
hann_ram_data[15:0]	0001

Fig 38: Test Bench Simulation Result of Lab Window

In figure 38, simulations result shows the both coefficients, ADC data and the multiplication result. “ram_b_dataout” represents the analog signal coming from XADC, “hann_rom_data” represents the hamming window coefficients calculated in MATLAB and “hann_ram_data” represents the multiplication result which is truncated from 32-bit to 16-bit.

3.5 FPGA FFT Block

The FFT block makes use of an FFT IP block provided by Xilinx in Vivado. Our FFT block is essentially a controller for the IP block.

The IP block has 32 bits as input and 64 as output, and performs 512 point FFT. In the 32 bit input, 16 bits are used for the real part and 16 bits are used for the imaginary part. We only use the 16 bit real part in our input, and insert zeros for the other 16 bits. Out of the 64 output bits, 26 bits are used for the real part and 26 bits are for the imaginary part. The remaining bits are of no use to us.

The IP block also features a configuration channel. While creating the IP block, we had selected options that makes most of the parameters static, therefore in our use case the configuration channel cannot control many options. There wasn’t an option in the IP block to always perform forward FFT, so the configuration channel is capable of setting the IP block to perform inverse FFT. As we are not interested in performing IFFT, the configuration channel was not used, and all the inputs were set to zero.

Our FFT controller takes 16 bits as input and 16 bits as output. It takes the 16 bit output of the hamming block as input. Because we are interested in the magnitude of the FFT output, we store the 8 most significant bits of the real part of the output, and the 8 most significant bits of the imaginary part. The FFT controller communicates with the IP block by setting the IP block’s input data to the first value in the frame, then setting a signal to “1” to indicate that we are ready to send data, then changing the IP block’s input value to the next value of the frame each clock cycle. After sending all the values in the frame, we wait for a signal from the IP block to become “1”, then receive the frame one value at a time each clock cycle.

Our top module controls the FFT block so that after we receive all the values in the frame, the process may be repeated for the next frame.

Because we are sending and receiving values each clock cycle, and writing to our DRAM block requires more than one clock cycle, both our FFT controller and the FFT IP block use a 25 MHz clock while the DRAM block uses a 100 MHz clock.

We tested our FFT controller by using matlab to create a 4 kHz sine wave sampled at 16 kHz, the sampling rate of our ADC block, and stored the signal in a text file. We supplied the signal in the text file as input to our FFT controller during the testbench, and stored the output of the FFT controller in a text file also. We then loaded the signal in the output into matlab to examine. The input and output signals can be seen below.

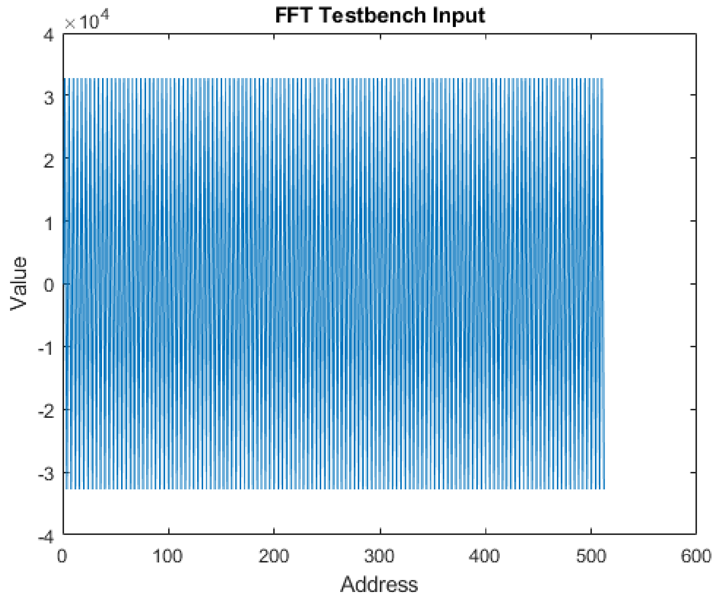


Fig 39: FFT Test Bench Input Signal

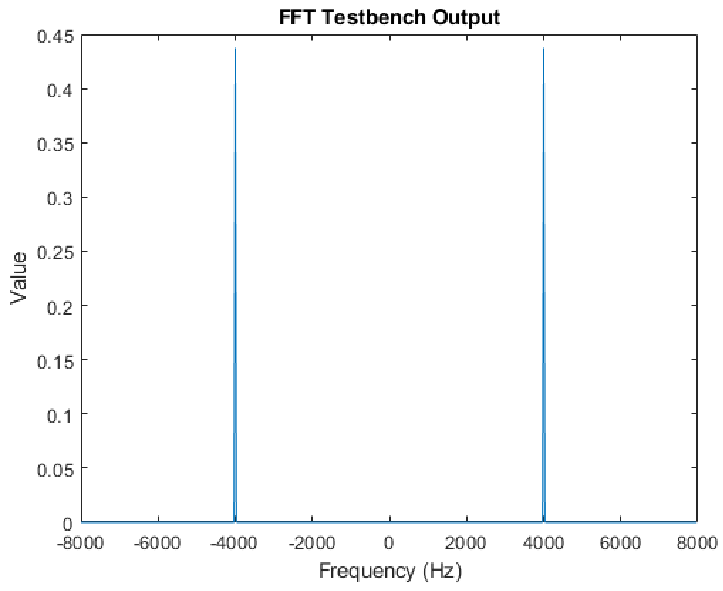
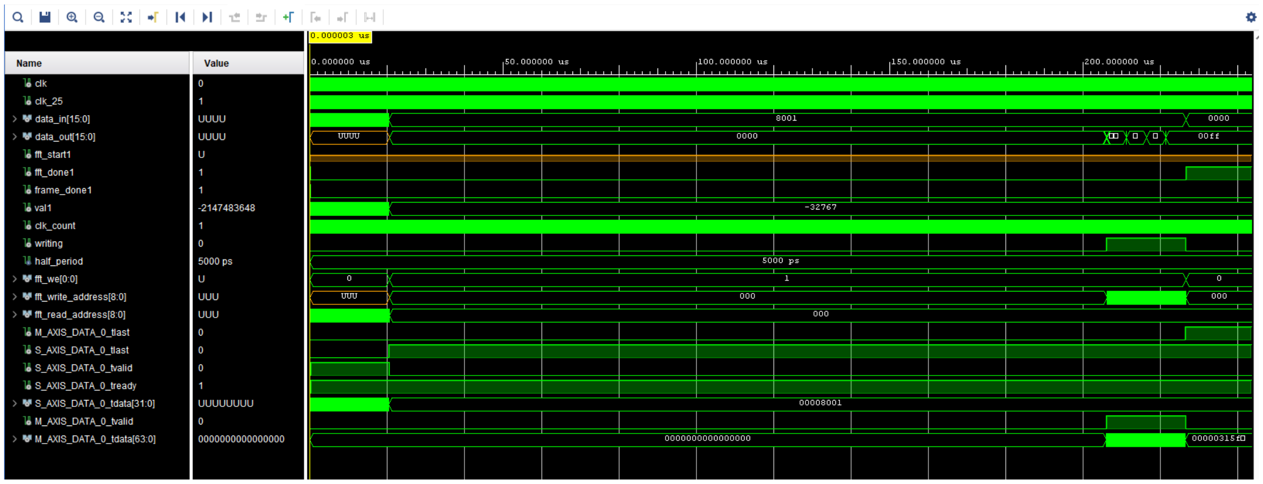


Fig 40: FFT Test Bench Output Signal

The signals in the test bench can be seen below.



Name	Value
clk	0
clk_25	1
data_in[15:0]	UUUU
data_out[15:0]	UUUU
fft_start1	U
fft_done1	1
frame_done1	1
val1	-2147483648
clk_count	1
writing	0
half_period	5000 ps
fft_we[0:0]	U
fft_write_address[8:0]	UUU
fft_read_address[8:0]	UUU
M_AXIS_DATA_0_tlast	0
S_AXIS_DATA_0_tlast	0
S_AXIS_DATA_0_tvalid	0
S_AXIS_DATA_0_tready	1
S_AXIS_DATA_0_tdata[31:0]	UUUUUUUU
M_AXIS_DATA_0_tvalid	0
M_AXIS_DATA_0_tdata[63:0]	0000000000000000

Fig 41: Test Bench Simulation of the FFT Block

“S_AXIS_DATA_0_tdata” is the IP block’s input and “M_AXIS_DATA_0_tdata” is its output. We set “S_AXIS_DATA_0_valid” to “1” to indicate we are sending data at the beginning and set it to “0” after we are done. We then wait for “M_AXIS_DATA_0_tvalid” to become “1”, and we store the output data while it is “1”.

As a final test, we gave a played a 3 kHz sine wave audio next to the microphone in our PCB, so that the ADC would give that signal as input to our FFT controller, and our debug block would send the data to matlab. The results can be seen below.

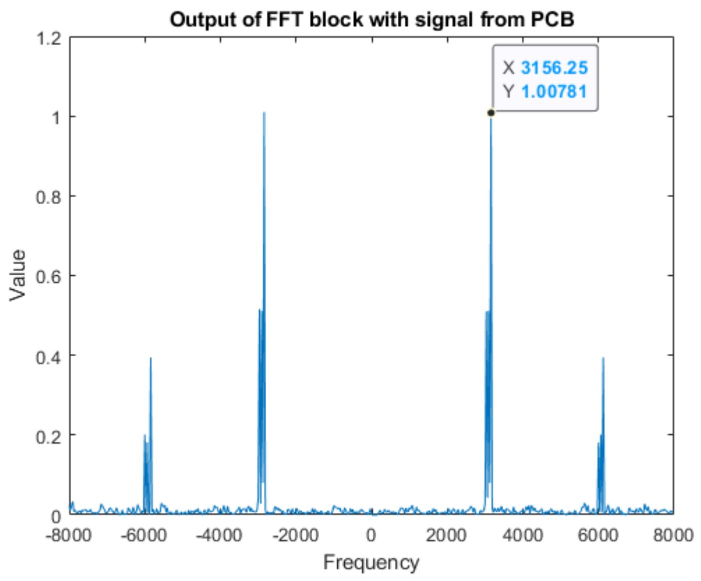


Fig 42: MATLAB Graph of the FFT Block with Input Signal from PCB

The harmonic at 6 kHz is most likely a result of the components on the PCB.

4 MATLAB PART

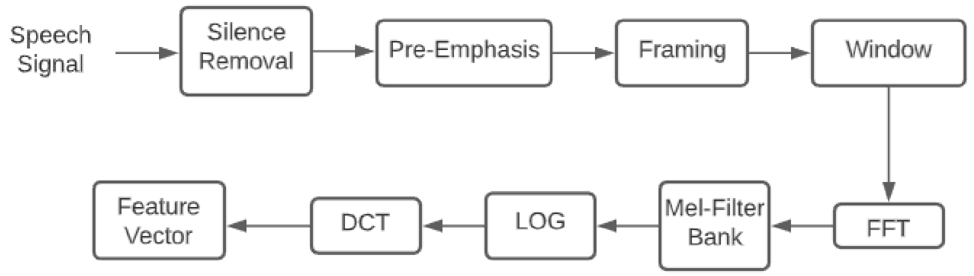


Fig 43: Pre-Processing and Feature Extraction

Firstly, a silence removal process is implemented on the captured audio signal since it may have silent part at different positions because of the timing of the speaker. It is important to remove these unimportant parts for speech recognition so that the model is not trained incorrectly. Then we use a function called MFCC(S,FS,TW,TS,ALPHA,WINDOW,R,M,N,L) and this function returns Mel frequency cepstral coefficients of the speech signal (S) which is sampled at FS. Since the ADC of our project use 16K as a sampling rate, we set the FS as 16000. Usually, the speech signal is pre-emphasized using a first order high pass filter and the ALPHA represents the pre-emphasis coefficient. TW and TS represent frame durations and frame shifts. We use 30msec duration audio frames with about 50% overlapped (15msec) with the previous audio frame (512 samples per frame) and we use hamming window function which is a most commonly used function for speech recognition and suitable for Fourier analysis. This process is followed by filter bank design to compute the magnitude spectrum. The filter bank has M triangular filters which can be specified from function MFCC. With the logarithmic filter bank energies, discrete cosine transform is implemented to calculate cepstral coefficients and finally applying a sinusoidal lifter, MFCCs is produced. After obtaining the feature vector for each number, speech recognition can be made by looking at the minimum Euclidian distance between the tested number and each vector in this set.

4.1 Sampling the PC Microphone Block

First of all, using audiorecorder(Fs,nBits,NumChannels) function of MATLAB, the speech signal is recorded with Fs = 16000, nBits = 16, NumChannels = 1 during 2 seconds. Since the internal ADC sample rate is 16K, we set Fs as 16000. To get the similar data from ADC, we should convert 16-bit speech signal in to a 12-bit resolution. From figure 44 and 45, the original signal in 16-bit and the new signal in 12-bit resolution can be seen. For this signal, the sound of number “seven” is used.

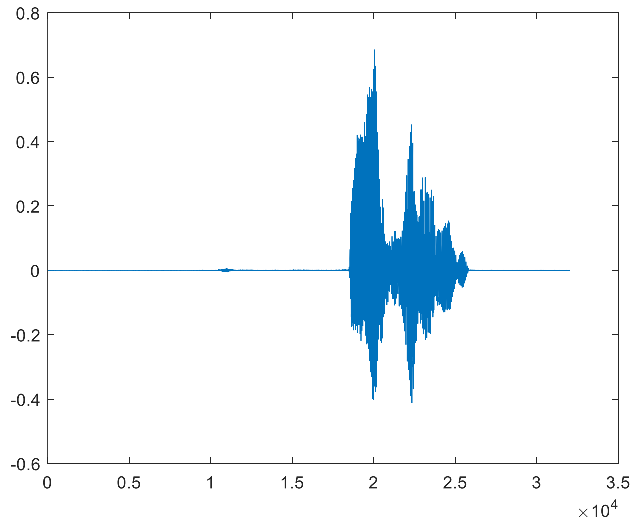


Fig 44: Speech Signal in 16-bit(Double)

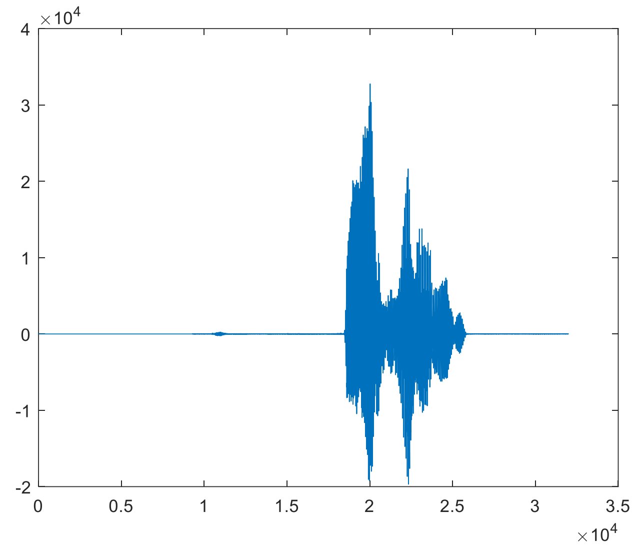


Fig 45: Speech Signal in 12-bit Resolution(Integer)

To remove the silence part of the speech signal for more accurate results, we use end-point detection algorithm. To specify the background noise, the mean and the standard deviation of the first 800 samples (16000 sample/ 2 seconds) are calculated. For each sample, $\frac{|x-\mu|}{\sigma}$ distance is checked to mark the samples as voiced (1) or unvoiced (0). If number of zeros are greater than number of ones, the ones and zeros are converted and at the end only the voiced parts with label 1 are collected to get the silence removed signal. The output of this algorithm is seen in figure 46 (again for number "seven").

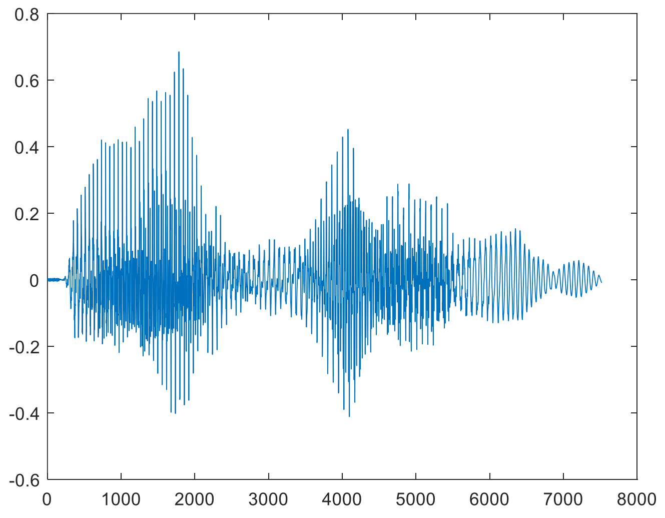


Fig 46: Output Signal from End-Point Detection Algorithm

Lastly, using a first order high pass filter, the speech signal is pre-emphasized. The reason for this, the energy level at lower frequencies is higher than the higher frequencies. With pre-emphasizing, the energy in the higher frequencies becomes more visible and it increases the accuracy of recognition. The filter equation can be written in time domain with input x[n] as follows:

$$y[n] = x[n] - \mathbf{a}x[n-1]$$

where **a** is the pre-emphasis coefficient between 0.9 and 1. We use **a** = 0.95.

4.2 Windowing Block

Since the speech can be categorized as a non-stationary signal, we use small windows of the speech to extract the spectral features and hamming window is a good candidate for the speech recognition systems. Hamming window function graph and formula is seen below.

$$S_w(n) = 0.54 - 0.46\cos\left(\frac{2\pi[0{:}N-1]}{N-1}\right)$$

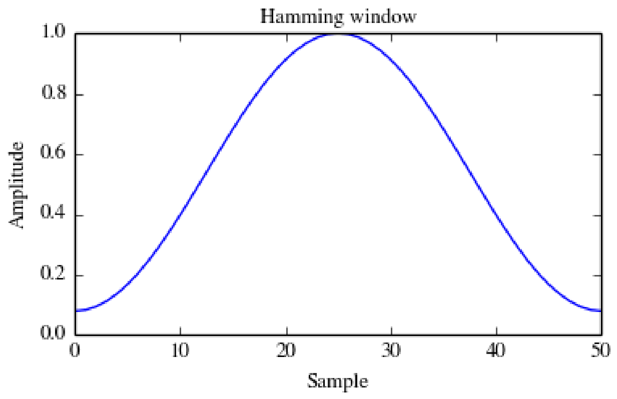


Fig 47: Hamming Window Graph

We divide our speech signal into 63 frames with 512 samples for each as it is explained and implemented in the hardware part (Lab Window). Then, by multiplying the each frame of the speech signal with the window function, we get the resulting signal. From figure 48, the windowing effect can be seen for one frame.

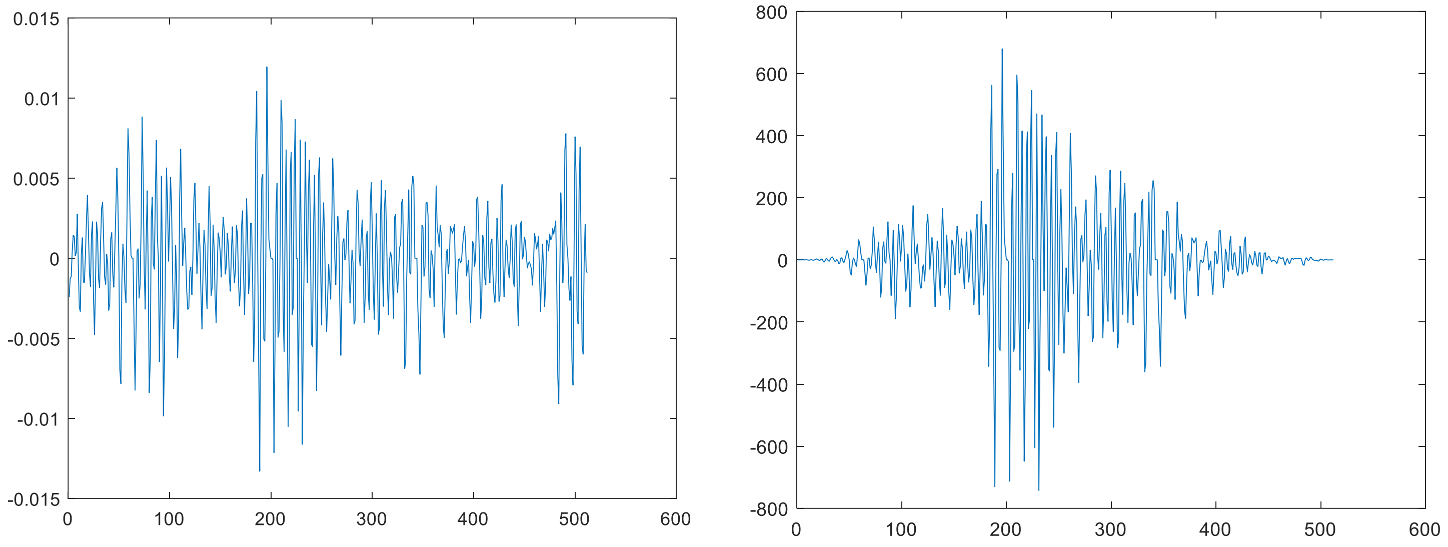


Fig 48: A Single Frame (10th) of Speech Signal Before (Left) and After (Right) Windowing

4.3 FFT Block

In this part, the Fast Fourier Transform of each windowed signal is used to reach the frequency content of each frame. As it is done in Vivado, we take the 512 point FFT and the absolute value of the FFT result for each frame. Since the magnitude of the signal at various frequencies is necessary and the phase is not important for us, we take the magnitude of it. The FFT result which is symmetric around zero of the previous frame can be seen below.

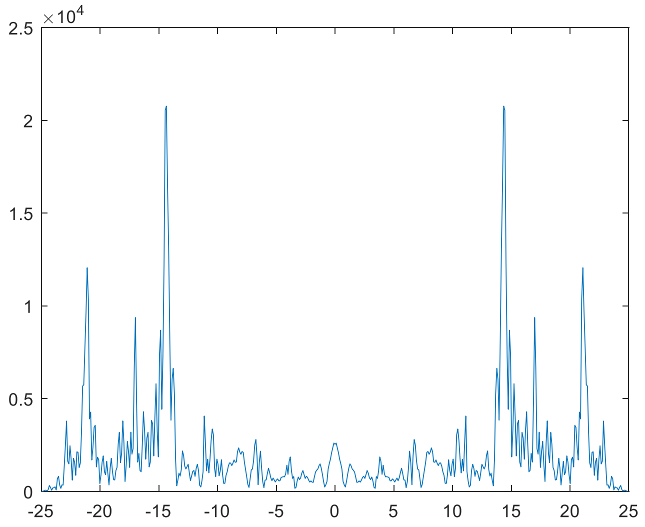


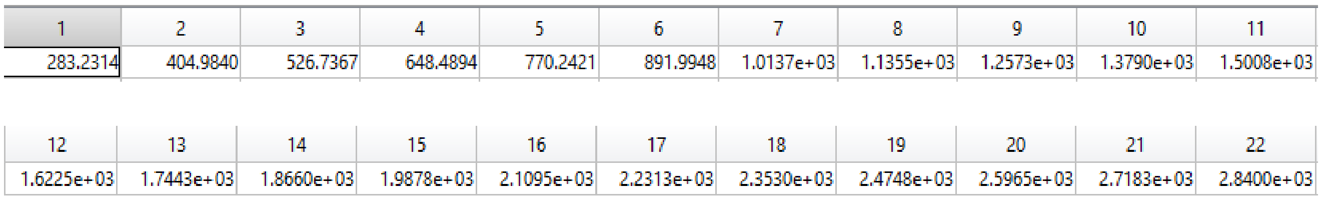
Fig 49: FFT Output of the same frame (10th)

4.4 MEL Filter Block

In this part, FFT result of each frame is multiplied by Mel Frequency Cepstral Coefficients. Before calculating MFCC, a transformation is implemented using the formula below:

Mel(x) = $2595\ x \log(1 + \frac{x}{700})$ where x is the linear frequency.

Generally, MFCC calculations are done using a filter bank where filters centered on the Mel frequencies. In our design, we use 20 different filters and the Mel frequencies of these filters is below.



1	2	3	4	5	6	7	8	9	10	11
283.2314	404.9840	526.7367	648.4894	770.2421	891.9948	1.0137e+03	1.1355e+03	1.2573e+03	1.3790e+03	1.5008e+03

12	13	14	15	16	17	18	19	20	21	22
1.6225e+03	1.7443e+03	1.8660e+03	1.9878e+03	2.1095e+03	2.2313e+03	2.3530e+03	2.4748e+03	2.5965e+03	2.7183e+03	2.8400e+03

Fig 50: Center Frequencies of the Filters

The Mel filter gives energy information of each scale band. The resulting Mel filter bank with 20 filters can be seen below.

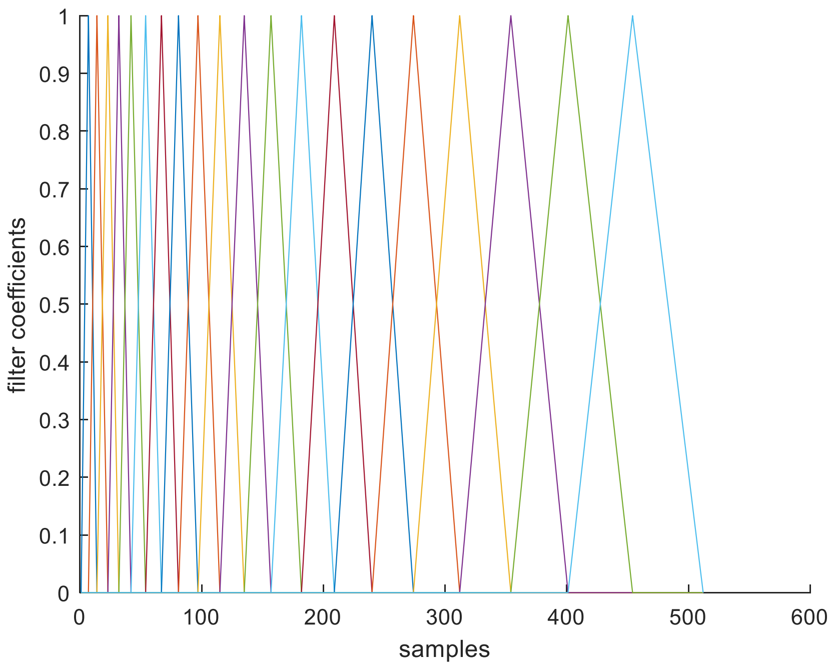


Fig 51: Mel Filter Bank

After calculating the Mel Filter Matrix, we multiply the FFT result of the speech signal with these coefficients for each frame and each filter. For the same frame (10th), the Mel coefficient graph for each filter (there are 20 filters in total) can be seen below.

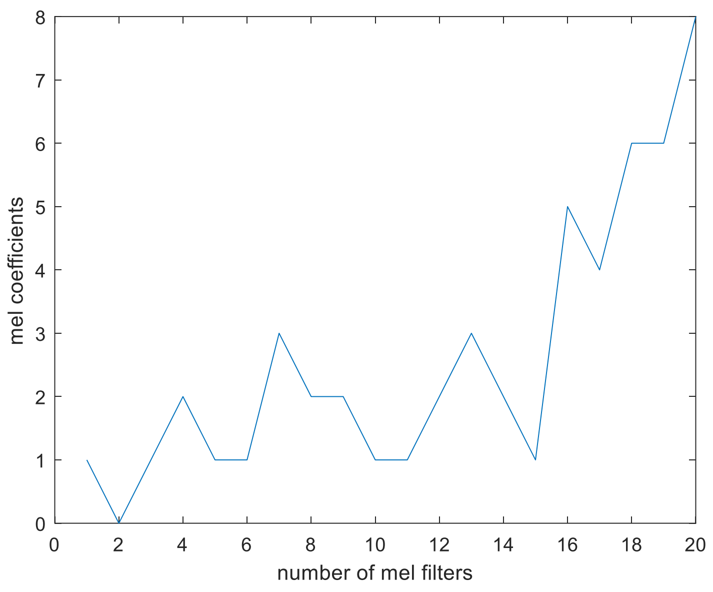


Fig 52: Mel Coefficients of the 10th Frame

As it is mentioned, by taking the logarithm of these Mel coefficients in figure 52, the convolutional distortions (e.g. microphone filter, frequency domain multiplication) convert an addition and it also gives a more realistic approach because of the logarithmic power of the real life sound. The logarithmic coefficients graph for the same frame can be seen below.

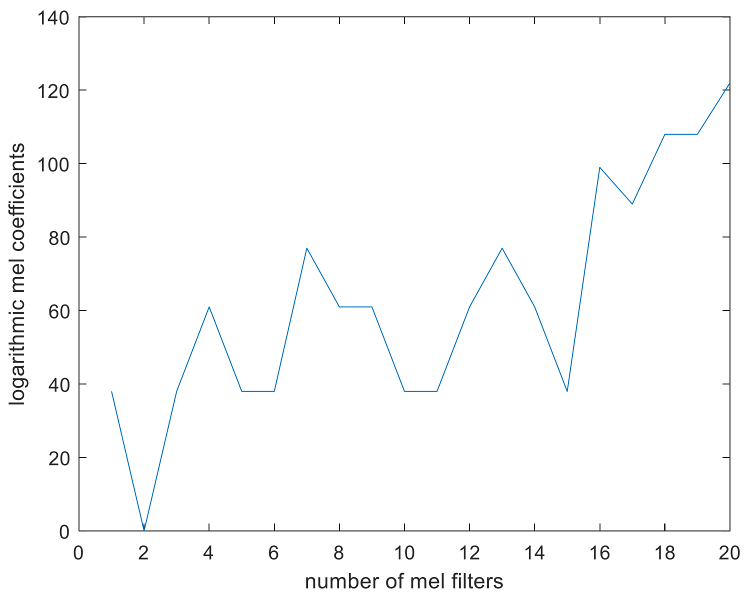


Fig 53: Logarithm of the Mel Coefficients for the 10th Frame

4.5 DCT Block

In order to get MFCC feature of each frame, discrete cosine transform is applied to the logarithmic Mel coefficients. DCT can be represented as sum of sinusoidal signals at different amplitudes and frequencies. The formula of this process is expressed as follows:

$$c[n] = \sum_{i=1}^{M} \log\big(Y(i)\big)\, x \cos[\frac{\pi n}{M}(i - 0.5)]$$

where Y(i) is the Mel coefficients, M is the number of filters which is 20 and c[n] is MFCC feature vector for each frame.

DCT of the logarithmic Mel coefficients for frame 10 is below.

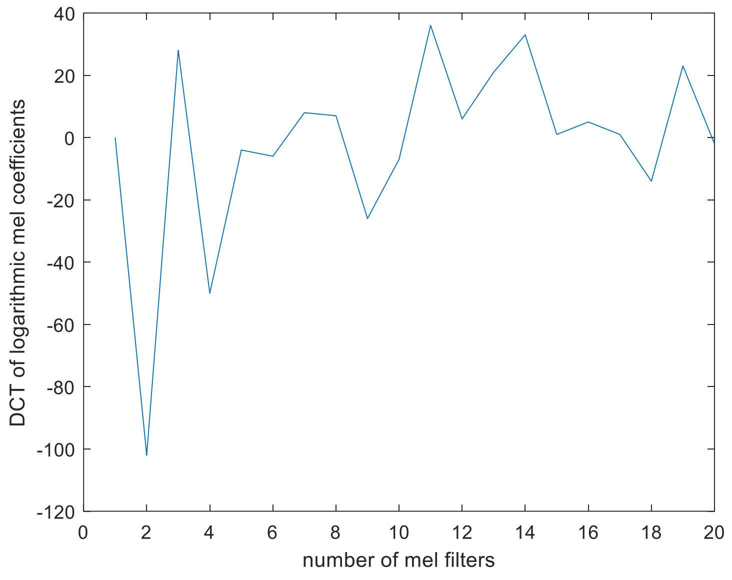


Fig 54: DCT of the Logarithmic Mel Coefficients for the 10th Frame

4.6 Classifier Block

In this block, to create a data set, we record each number 5 times and obtain 5x10 = 50 feature vectors in total. Then we looked at the Euclidian distance between the feature vector of the number we tested and the vectors in our data set and printed the number with the smallest distance on the MATLAB screen as speech recognition guess. The Euclidian distance between two vectors can be written as

$$d(p, q) = \sqrt{\sum_{i=1}^{n}(p(i) - q(i))^2}$$

where p is the feature vector the of test speech signal and q is the feature vector of the guess signal in data set.

4.7 Verification of the System on MATLAB

In the training stage, first one of the group members records his voice 5 times and test each number in a silent room. Under this condition, we get an accuracy rate of %85. Then we try the system with the other two members of the group and the accuracy rate decreases to %53. To make the system speaker independent, we create the data set using all group members rather than just one group member. Therefore, the other two members records their voice as well and we get the average of the three different feature vector set to decrease the speaker dependency of the system. With this new data set, the system gives %78 accuracy regardless of the speaker. The table showing trials for each number with the average of %78 accuracy can be seen below.

Spoken Number	Trial-1	Trial-2	Trial-3	Trial-4	Trial-5	Trial-6	Trial-7	Trial-8	Trial-9	Trial-10	Accuracy
0	0	7	0	0	0	0	0	9	0	0	%80
1	1	4	1	1	1	1	5	1	4	1	%70
2	4	2	2	2	1	2	2	2	2	2	%80
3	2	3	8	3	3	3	3	2	3	3	%70
4	5	4	4	5	4	4	4	5	4	4	%70
5	5	5	5	5	8	5	5	5	5	4	%80
6	6	2	6	6	6	6	1	6	6	6	%80
7	7	7	7	7	7	7	7	7	7	7	%100
8	9	8	8	8	5	8	8	5	8	8	%70
9	9	1	9	9	9	9	1	9	9	9	%80

Table 1: Speech Recognition Statistics for MATLAB Part

5 INTEGRATION OF HW PART AND MATLAB PART AND TESTING OF THE SYSTEM

As we explained before, after the FFT block, the rest of the project is implemented in MATLAB meaning that Mel coefficients and DCT calculations are done in MATLAB and by looking at the Euclidian distances of the feature vectors, the best possible guess is printed on the screen.

At every stage of the project, we test our algorithm by giving Basys3 sinusoidal signals as input from the signal generator and viewing their outputs on MATLAB. Graphics of these tests are given in the descriptions of the relevant labs. After the PCB implementation, we check our connections with voltmeter (short circuit test) and fix the problems that can be caused by soldering errors. We also verify the offset values through oscilloscope without giving input to our circuit. Then we test the amplifier gain and filter attenuation values over the oscilloscope by giving input from the signal generator, and we observe the output of the audio signals by soldering the microphone. For the MATLAB test of PCB, we use Lab Debug and Lab ADC to transfer the audio signals to MATLAB and check the clarity of these sounds by listening them. The graphics we observe in MATLAB for each digit input given from PCB are shown below.

By integrating the Lab Window with the Lab ADC, we plot the windowed outputs of the audio signals in MATLAB. For Lab FFT, which is the last hardware phase of the project, after we make the FFT calculations of these audio signals in Bays3, we transfer them to MATLAB and continue the remaining calculations in there.

For the speech recognition test, we apply the method in the MATLAB section and create a table by testing each number 10 times. Since the sound signals we receive from the PCB contain more noise than the PC recordings, our average accuracy in hardware speech recognition tests drops to 70%.

Spoken Number	Trial-1	Trial-2	Trial-3	Trial-4	Trial-5	Trial-6	Trial-7	Trial-8	Trial-9	Trial-10	Accuracy
0	0	0	7	0	9	9	0	0	0	0	%70
1	1	1	4	1	9	1	1	1	1	4	%70
2	2	5	2	1	2	2	2	5	2	2	%70
3	3	2	3	8	3	3	8	3	3	8	%60
4	4	4	5	4	4	4	1	4	5	4	%70
5	5	6	5	8	5	5	5	5	5	5	%80
6	6	6	6	2	6	6	6	2	6	2	%70
7	7	7	1	7	7	7	7	7	0	7	%80
8	8	5	8	2	8	8	5	8	5	8	%60
9	9	9	1	9	7	9	9	1	9	9	%70

Table 2: Speech Recognition Statistics for Hardware Part

6 DISCUSSION AND CONCLUSION:

We have successfully completed a working speech recognition system such that anti-aliasing filter and amplifier circuit implemented on PCB, Debug block, ADC block, Window block and FFT block implemented on Basys3, Mel filter block, DCT block, classifier block implemented on MATLAB. In MATLAB simulations, we obtained an accurate rate of %78 and in the hardware tests, we obtained a lower accuracy rate of %70 because of higher noise in the speech signal.

We have encountered many problems such as working with different clocks (FFT-25 MHz, Basys3-100 MHz), triggering different modules with the right timing, increasing the baud rate, running the XADC module and setting the bias voltage according to the limit of XADC but at the end, we have overcome all problems and implemented a working system. For the different clocks, by defining a count signal in parts where a 100 MHz module communicates with a 25 MHz module, we made the slow module wait and see the required signal. To set the correct bias voltage, we tried different resistor values on breadboard and then solder to PCB. For the baud rate problem, we increased the baud rate gradually and put different waiting times where a sliding may occur in the data but we could not solve the problem. Every time we increased the baud rate, our data was corrupted, so we set the baud rate as 9600.

For future work of this design, the size of the training set and the variety of speakers can be increased to get higher accuracy rate. As the classification method, cross correlation or dynamic time warping can be tried instead of Euclidian distance. Lastly, filter mechanism of the PCB can be improved to reduce the noise of the speech signal.

7 REFERENCES

[1] J. Paradiso, “Readings,” *MIT OpenCourseWare, Massachusetts Institute of Technology*. [Online]. Available: https://ocw.mit.edu/courses/media-arts-and-sciences/mas-836-sensor-technologies-for-interactive-environments-spring-2011/readings. [Accessed: 07-Jan-2021].

[2] *Sallen-Key Low-pass Filter Design Tool*. [Online]. Available: http://sim.okawa-denshi.jp/en/OPseikiLowkeisan.htm. [Accessed: 08-Jan-2021].

[3] S. K, “Basys 3 XADC Demo,” *Basys 3 XADC Demo [Digilent Documentation]*. [Online]. Available: https://reference.digilentinc.com/learn/programmable-logic/tutorials/basys-3-xadc/start. [Accessed: 07-Jan-2021].

[4] “Support,” *Xilinx*. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC. [Accessed: 08-Jan-2021].

8 APPENDIX

8.1 VHDL codes and Testbench codes of FPGA blocks

top_module.vhd

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

ENTITY top_module_uart IS
        PORT (
                enable : IN std_logic;
                start_adc : IN std_logic;
                -- enable_ram : in std_logic;
                reset : IN std_logic;
                clk : IN std_logic;
                out_seri : OUT std_logic;
                led : out std_logic_vector(15 downto 0);
                vauxp6 : in std_logic;
                vauxn6 : in std_logic
        );

END top_module_uart;

ARCHITECTURE Behavioral OF top_module_uart IS

        SIGNAL tick : std_logic;
        SIGNAL data : std_logic_vector(7 DOWNTO 0);
        SIGNAL done : std_logic;
        signal ram_a_addr: STD_LOGIC_VECTOR (13 downto 0);
        signal ram_b_addr: STD_LOGIC_VECTOR (13 downto 0);
    signal ram_a_datain: STD_LOGIC_VECTOR (63 downto 0);
    signal ram_b_dataout: STD_LOGIC_VECTOR (63 downto 0);
    signal ram_a_we: STD_LOGIC_VECTOR(0 downto 0);

    signal hann_ram_data: std_logic_vector(15 downto 0);
    signal hann_ram_addr: std_logic_vector(8 downto 0);

    signal hann_ram_write_data: std_logic_vector(15 downto 0);
    signal hann_ram_write_addr: std_logic_vector(8 downto 0);

    signal hann_write_en: std_logic_vector(0 downto 0);

    signal hann_rom_data :std_logic_vector(15 DOWNTO 0);
        signal hann_rom_addr :std_logic_vector(8 DOWNTO 0);
        -- signal frame_done : std_logic;
        -- signal fsm_done: std_logic;

         signal fft_ram_addr_read: std_logic_vector(8 downto 0);
          signal fft_ram_addr_write: std_logic_vector(8 downto 0);
          signal fft_ram_data_read: std_logic_vector(15 downto 0);
          signal fft_write_data: std_logic_vector(15 downto 0);

          signal fft_start: std_logic;
          -- signal fft_done: std_logic;
    --signal out_portb: STD_LOGIC_VECTOR (63 downto 0);
        --signal enable2: std_logic;
        --signal outInSeri : std_logic;
        --signal parity1 : std_logic;


        signal M_AXIS_DATA_0_tdata :STD_LOGIC_VECTOR ( 63 downto 0 );
    signal M_AXIS_DATA_0_tlast : STD_LOGIC;
    signal M_AXIS_DATA_0_tvalid : STD_LOGIC;
    signal S_AXIS_CONFIG_0_tdata :STD_LOGIC_VECTOR ( 7 downto 0 );
    signal S_AXIS_CONFIG_0_tready : STD_LOGIC;
    signal S_AXIS_CONFIG_0_tvalid :STD_LOGIC;
    signal S_AXIS_DATA_0_tdata : STD_LOGIC_VECTOR ( 31 downto 0 );
    signal S_AXIS_DATA_0_tlast : STD_LOGIC;
    signal S_AXIS_DATA_0_tready : STD_LOGIC;
    signal S_AXIS_DATA_0_tvalid : STD_LOGIC;
    signal aclk_0 : STD_LOGIC;
    signal aresetn_0 : STD_LOGIC;
    signal event_data_in_channel_halt_0 : STD_LOGIC;
    signal event_frame_started_0 : STD_LOGIC;
    signal event_tlast_missing_0 : STD_LOGIC;
    signal event_tlast_unexpected_0 : STD_LOGIC;



        signal write_done : STD_LOGIC;
        signal adc_data : std_logic_vector(15 downto 0);
        signal fft_we : std_logic_vector(0 downto 0);
        --signal enable_ram : std_logic := '1';

    SIGNAL clk_25: std_logic := '1';
    signal clk_count: integer := 0;

    TYPE state_type IS (s_fft, s_fsm, s_hann);
    SIGNAL state : state_type := s_hann;

    signal s_fft_en : std_logic := '0';
    signal s_fsm_en : std_logic := '0';
    signal s_hann_en : std_logic := '0';
    signal fft_done : std_logic;
    signal fsm_done : std_logic;
    signal frame_done : std_logic;



        COMPONENT mod_m_counter
                PORT (
                        reset : IN std_logic;
                        clk : IN std_logic;
                        tick1 : OUT std_logic
                );
        END COMPONENT;

        COMPONENT transmitter
                PORT (
                        tick2 : IN std_logic;
                        enable : IN std_logic;
                        in_paralel8 : IN std_logic_vector(7 DOWNTO 0);
                        reset : IN std_logic;
                        clk : IN std_logic;
                        out_seri : OUT std_logic;
                        t_done : OUT std_logic
                );

        END COMPONENT;


    COMPONENT design_3_wrapper
    port (
        BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
        BRAM_PORTA_0_clk : in STD_LOGIC;
        BRAM_PORTA_0_din : in STD_LOGIC_VECTOR ( 15 downto 0 );
        BRAM_PORTA_0_en : in STD_LOGIC;
        BRAM_PORTA_0_we : in STD_LOGIC_VECTOR ( 0 to 0 );
        BRAM_PORTB_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
        BRAM_PORTB_0_clk : in STD_LOGIC;
        BRAM_PORTB_0_dout : out STD_LOGIC_VECTOR ( 15 downto 0 )
    );
    END COMPONENT;


        component design_2_wrapper
    port(
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_dout : out STD_LOGIC_VECTOR ( 15 downto 0 );
```

```
BRAM_PORTA_0_en : in STD_LOGIC
);
   end component;




   COMPONENT FSM_DEBUG
           PORT (
           enable : IN std_logic;
           --enable_ram_a : IN std_logic;
           clk : IN std_logic;
           reset : IN std_logic;
           in_paralel16 : in STD_LOGIC_VECTOR (15 downto 0);
           -- fft_done : IN std_logic;
           fsm_done : OUT std_logic;
           s_fsm_en : in std_logic;

           tick3 : IN std_logic;
    out_paralel : out std_logic_vector(7 downto 0);
    uart_done : in std_logic;
    --ram_address_a : out STD_LOGIC_VECTOR (13 downto 0) := "00000000000000";
    ram_address_b : out STD_LOGIC_VECTOR (8 downto 0) := "000000000"
           );

   END COMPONENT;

   component design_1_wrapper

port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 13 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_din : in STD_LOGIC_VECTOR ( 63 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC;
    BRAM_PORTA_0_we : in STD_LOGIC_VECTOR ( 0 to 0 );
    BRAM_PORTB_0_addr : in STD_LOGIC_VECTOR ( 13 downto 0 );
    BRAM_PORTB_0_clk : in STD_LOGIC;
    BRAM_PORTB_0_dout : out STD_LOGIC_VECTOR ( 63 downto 0 )
  );
end component;

component XADCdemo
Port (
    CLK100MHZ : in STD_LOGIC;
    vauxp6 : in STD_LOGIC;
    vauxn6 : in STD_LOGIC;
    LED : out STD_LOGIC_VECTOR(15 downto 0)
);
end component;

component adc_to_ram
Port(
clk : in std_logic;
reset: in std_logic;
enable_adc_to_ram : in std_logic;
ram_write_data: out std_logic_vector(63 downto 0);
ram_write_addr: out std_logic_vector(13 downto 0);
ram_address_b: out std_logic_vector(13 downto 0);
ram_write_en : out std_logic_vector(0 downto 0);
hann_ram_we : OUT std_logic_vector(0 DOWNTO 0);
   dram_read : IN std_logic_vector(63 DOWNTO 0);
   hann_rom_data : IN std_logic_vector(15 DOWNTO 0);
   hann_rom_addr : OUT std_logic_vector(8 DOWNTO 0);
   hann_ram_data : OUT std_logic_vector(15 DOWNTO 0);
   hann_ram_addr : OUT std_logic_vector(8 DOWNTO 0);
adc_data: in std_logic_vector(15 downto 0);
adc_to_ram_done : out std_logic;
-- FSM_DONE: IN STD_LOGIC;

   frame_done : OUT std_logic := '0';
   s_hann_en : in std_logic;
   all_frames_done : OUT std_logic := '0'
```

```
    );
    end component;

    component fft_ram_wrapper
      port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_din : in STD_LOGIC_VECTOR ( 15 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC;
    BRAM_PORTA_0_we : in STD_LOGIC_VECTOR ( 0 to 0 );
    BRAM_PORTB_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTB_0_clk : in STD_LOGIC;
    BRAM_PORTB_0_dout : out STD_LOGIC_VECTOR ( 15 downto 0 )
  );
  end component;


  component fft_wrapper
    port (
    M_AXIS_DATA_0_tdata : out STD_LOGIC_VECTOR ( 63 downto 0 );
    M_AXIS_DATA_0_tlast : out STD_LOGIC;
    M_AXIS_DATA_0_tvalid : out STD_LOGIC;
    S_AXIS_CONFIG_0_tdata : in STD_LOGIC_VECTOR ( 7 downto 0 );
    S_AXIS_CONFIG_0_tready : out STD_LOGIC;
    S_AXIS_CONFIG_0_tvalid : in STD_LOGIC;
    S_AXIS_DATA_0_tdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    S_AXIS_DATA_0_tlast : in STD_LOGIC;
    S_AXIS_DATA_0_tready : out STD_LOGIC;
    S_AXIS_DATA_0_tvalid : in STD_LOGIC;
    aclk_0 : in STD_LOGIC;
    aresetn_0 : in STD_LOGIC;
    event_data_in_channel_halt_0 : out STD_LOGIC;
    event_frame_started_0 : out STD_LOGIC;
    event_tlast_missing_0 : out STD_LOGIC;
    event_tlast_unexpected_0 : out STD_LOGIC
  );

    end component;

    component fft_controller
    port (
    clock: in std_logic;
    fft_start: in std_logic;
    fft_done: out std_logic;
    s_fft_en : in std_logic;
    -- frame_done: in std_logic;

    -- FROM ADC'S RAM
    fft_read_address: out std_logic_vector(8 downto 0);
    fft_read_data: in std_logic_vector(15 downto 0);

    -- TO DUAL PORT RAM
    fft_write_address: out std_logic_vector(8 downto 0);
    fft_write_data: out std_logic_vector(15 downto 0);
    fft_we: out std_logic_vector(0 downto 0);

    -- FFT IP CONFIG PORTS
    ip_config_tvalid: out std_logic;
    ip_config_tready: in std_logic;
    ip_config_tdata: out std_logic_vector(7 downto 0);

    -- FFT IP DATA INPUT PORTS
    ip_in_tvalid: out std_logic;
    ip_in_tready: in std_logic;
    ip_in_tdata: out std_logic_vector(31 downto 0);
    ip_in_tlast : OUT std_logic;

    -- FFT IP DATA OUTPUT PORTS
    ip_out_tvalid: in std_logic;
    ip_out_tdata: in std_logic_vector(63 downto 0)
);
```

```
    end component;




BEGIN
        counter : mod_m_counter
        PORT MAP(
                reset => reset,
                clk => clk,
                tick1 => tick
        );

        tran : transmitter
        PORT MAP(

                clk => clk,
                tick2 => tick,
                enable => enable,
                in_paralel8 => data,
                reset => reset,
                out_seri => out_seri,
                t_done => done
        );

        fsm : FSM_DEBUG
        PORT MAP(
                enable => enable,
                reset => reset,
                clk => clk,
                in_paralel16 => fft_ram_data_read,
                ram_address_b => fft_ram_addr_read,
                out_paralel => data,
                uart_done => done,
                fsm_done => fsm_done,
                -- fft_done => fft_done,
                s_fsm_en => s_fsm_en,
                tick3 => tick

        );


         dram: design_1_wrapper
         port map(
        BRAM_PORTA_0_addr => ram_a_addr,
        BRAM_PORTA_0_clk => clk,
        BRAM_PORTA_0_din => ram_a_datain,
        BRAM_PORTA_0_en => '1',
        BRAM_PORTA_0_we => ram_a_we,
        BRAM_PORTB_0_addr  => ram_b_addr,
        BRAM_PORTB_0_clk  => clk,
        BRAM_PORTB_0_dout  => ram_b_dataout
 );

 fft: fft_wrapper
 port map(
     M_AXIS_DATA_0_tdata =>  M_AXIS_DATA_0_tdata,
    M_AXIS_DATA_0_tlast => M_AXIS_DATA_0_tlast,
    M_AXIS_DATA_0_tvalid => M_AXIS_DATA_0_tvalid,
    S_AXIS_CONFIG_0_tdata => S_AXIS_CONFIG_0_tdata,
    S_AXIS_CONFIG_0_tready => S_AXIS_CONFIG_0_tready,
    S_AXIS_CONFIG_0_tvalid => S_AXIS_CONFIG_0_tvalid,
    S_AXIS_DATA_0_tdata => S_AXIS_DATA_0_tdata,

    S_AXIS_DATA_0_tlast => S_AXIS_DATA_0_tlast,
    S_AXIS_DATA_0_tready => S_AXIS_DATA_0_tready,
    S_AXIS_DATA_0_tvalid => S_AXIS_DATA_0_tvalid,
    aclk_0 => clk_25,
    aresetn_0 => '1',
    event_data_in_channel_halt_0 => event_data_in_channel_halt_0,
    event_frame_started_0 => event_frame_started_0,
    event_tlast_missing_0 => event_tlast_missing_0,
    event_tlast_unexpected_0 => event_tlast_unexpected_0

 );


 fft_ram: fft_ram_wrapper
   port map(
    BRAM_PORTA_0_addr => fft_ram_addr_write,
    BRAM_PORTA_0_clk => clk,
    BRAM_PORTA_0_din => fft_write_data,
    BRAM_PORTA_0_en => '1',
    BRAM_PORTA_0_we => fft_we,
    BRAM_PORTB_0_addr => fft_ram_addr_read,
    BRAM_PORTB_0_clk => clk,
    BRAM_PORTB_0_dout => fft_ram_data_read
  );

    fft_cont: fft_controller
    port map (
    clock => clk_25,
    fft_start => fft_start,
    fft_done => fft_done,
    s_fft_en => s_fft_en,
    -- frame_done => frame_done,

    -- FROM ADC'S RAM
    fft_read_address =>  hann_ram_addr,
    fft_read_data => hann_ram_data,
    fft_we => fft_we,

    -- TO DUAL PORT RAM
    fft_write_address => fft_ram_addr_write,
    fft_write_data => fft_write_data,

    -- FFT IP CONFIG PORTS
    ip_config_tvalid => S_AXIS_CONFIG_0_tvalid,
    ip_config_tready => S_AXIS_CONFIG_0_tready,
    ip_config_tdata =>  S_AXIS_CONFIG_0_tdata,


    -- FFT IP DATA INPUT PORTS
    ip_in_tvalid =>  S_AXIS_DATA_0_tvalid,
    ip_in_tready => S_AXIS_DATA_0_tready,
    ip_in_tdata => S_AXIS_DATA_0_tdata,
    ip_in_tlast => S_AXIS_DATA_0_tlast,

    -- FFT IP DATA OUTPUT PORTS
    ip_out_tvalid =>  M_AXIS_DATA_0_tvalid,
    ip_out_tdata =>  M_AXIS_DATA_0_tdata
    );
    led(2) <= s_fft_en;
    led(3) <= s_fsm_en;
    led(4) <= s_hann_en;
    led(15 downto 5) <= hann_ram_data(10 downto 0);
 hannram: design_3_wrapper
 port map(
        BRAM_PORTA_0_addr =>hann_ram_write_addr,
        BRAM_PORTA_0_clk => clk,
        BRAM_PORTA_0_din => hann_ram_write_data,
        BRAM_PORTA_0_en => '1',
        BRAM_PORTA_0_we => hann_write_en,
        BRAM_PORTB_0_addr => hann_ram_addr,
        BRAM_PORTB_0_clk => clk,
        BRAM_PORTB_0_dout => hann_ram_data


 );


   xadc: XADCdemo
   port map(
      CLK100MHZ => clk,
      vauxp6 => vauxp6,
      vauxn6 => vauxn6,
      LED => adc_data
 );

 adctoram: adc_to_ram
 port map(
     clk => clk,
  enable_adc_to_ram => start_adc,
  ram_write_data => ram_a_datain,
  ram_write_addr => ram_a_addr,
  ram_write_en => ram_a_we,
  adc_data => adc_data,
  adc_to_ram_done => led(0),
  reset => reset,
  dram_read => ram_b_dataout,
  hann_ram_we => hann_write_en,
  hann_rom_data => hann_rom_data,
     hann_rom_addr => hann_rom_addr,

     hann_ram_data => hann_ram_write_data,
     hann_ram_addr => hann_ram_write_addr,
     -- FSM_DONE => fsm_done,
     s_hann_en => s_hann_en,
     frame_done => frame_done,
     ram_address_b => ram_b_addr,
     all_frames_done => led(1)
 );


     hann_rom: design_2_wrapper
  port map(
  BRAM_PORTA_0_addr => hann_rom_addr,
  BRAM_PORTA_0_clk => clk,
  BRAM_PORTA_0_dout => hann_rom_data,
  BRAM_PORTA_0_en => '1'
  );

  process(clk)
     begin
        if rising_edge(clk) then
            if clk_count > 7 then
                clk_count <= 0;
                 clk_25 <= not clk_25;
            else
                 clk_count <= clk_count + 1;
            end if;

                    CASE state IS
                           WHEN s_hann =>
                               s_hann_en <= '1';
                               s_fft_en <= '0';
                               s_fsm_en <= '0';
                               if frame_done = '1' then
                                   state <= s_fft;
                               end if;

                           WHEN s_fft =>
                  s_hann_en <= '0';
                               s_fft_en <= '1';
                               s_fsm_en <= '0';
                               if fft_done ='1' then
                                   state <= s_fsm;
                               end if;

                           WHEN s_fsm =>
                               s_hann_en <= '0';
                               s_fft_en <= '0';
                               s_fsm_en <= '1';
                               if fsm_done = '1' then
                                   state <= s_hann;
                                  end if;
                      END CASE;
            end if;
         end process;
END Behavioral;
```

baudrate.vhd

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

ENTITY mod_m_counter IS
        PORT (
                reset : IN std_logic;
                clk : IN std_logic;
                tick1 : OUT std_logic
        );
END mod_m_counter;

ARCHITECTURE Behavioral OF mod_m_counter IS

        SIGNAL count : INTEGER := 0;
        CONSTANT m : INTEGER := 10417; -- 10^8 / 9600
        --CONSTANT m : INTEGER := 1736;

BEGIN
        PROCESS (clk)
        BEGIN
                IF (reset = '1') THEN
                        count <= 0;
                ELSIF rising_edge(clk) THEN
                        IF (count = m - 1) THEN
                                count <= 0;
                                tick1 <= '1';
                        ELSE
                                count <= count + 1;
                                tick1 <= '0';
                        END IF;
                END IF;
        END PROCESS;
END Behavioral;
```

transmitter.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY transmitter IS
        PORT (
                tick2 : IN std_logic;
                enable : IN std_logic;
                reset : IN std_logic;
                busy_adc : in std_logic;
                in_paralel8 : IN std_logic_vector(7 DOWNTO 0);
                clk : IN std_logic;
                out_seri : OUT std_logic;
                t_done : OUT std_logic
        );
END transmitter;

ARCHITECTURE Behavioral OF transmitter IS

        TYPE state_type IS (idle, start, data, stop, init);
        SIGNAL state : state_type := init;
        SIGNAL N : INTEGER := 0;
        SIGNAL data_reg : std_logic_vector(7 DOWNTO 0) := (OTHERS => '0');
        SIGNAL tx_done : std_logic := '0';
        SIGNAL init_count : INTEGER := 0;
        signal idle_count : INTEGER := 0;
```

```
BEGIN
       PROCESS (clk)
       BEGIN
              IF rising_edge(clk) AND tick2 = '1' THEN

                     CASE state IS
                            WHEN init =>
                                data_reg <= in_paralel8;
                                   IF enable = '1' and busy_adc = '0' THEN
                                          IF init_count > 6 THEN
                                                 state <= idle;
                                          ELSE
                                                 init_count <= init_count + 1;
                                          END IF;
                                   END IF;

                            WHEN idle =>
                                   N <= 0;
                                   tx_done <= '0';
                                   out_seri <= '1';
                                   IF enable = '1' and reset = '0' and busy_adc = '0'  THEN
                                       if idle_count > 30 then
                                          state <= start;
                                          data_reg <= in_paralel8;
                                          idle_count <= 0;
                                          else
                                          idle_count <= idle_count + 1;
                                          end if;
                                   ELSE
                                          state <= idle;
                                   END IF;

                            WHEN start =>
                                   out_seri <= '0';
                                   N <= 0;
                                   state <= data;

                            WHEN data =>
                                   out_seri <= data_reg(N);
                                   IF (N < 7) THEN
                                          state <= data;
                                          N <= N + 1;
                                   ELSE
                                          N <= 0;
                                          state <= stop;
                                   END IF;

                            WHEN stop =>
                                   out_seri <= '1';
                                   state <= idle;
                                   tx_done <= '1';
                     END CASE;
              END IF;
       END PROCESS;
       t_done <= tx_done;
END Behavioral;
```

fsm_debug.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY FSM_DEBUG IS
       PORT (
              enable : IN std_logic;
              --enable_ram_a : IN std_logic;
              clk : IN std_logic;
              reset : IN std_logic;
              in_paralel16 : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
              -- fft_done : IN std_logic;
              fsm_done : OUT std_logic := '0';
```

```
		s_fsm_en : in std_logic;

		tick3 : IN std_logic;
		out_paralel : OUT std_logic_vector(7 DOWNTO 0);
		uart_done : IN std_logic;
		--ram_address_a : out STD_LOGIC_VECTOR (13 downto 0) := "00000000000000";
		ram_address_b : OUT STD_LOGIC_VECTOR (8 DOWNTO 0) := "000000000"
		--out_ram: out STD_LOGIC_VECTOR(63 downto 0)
		--ram_we: out STD_LOGIC_VECTOR(0 downto 0) := "0";
		--led : out std_logic_vector(0 downto 0) := "0"
	);
END FSM_DEBUG;

ARCHITECTURE Behavioral OF FSM_DEBUG IS

	--- TYPE state_type IS (one, two, three, four, five, six, seven, eight);
	TYPE state_type IS (one, two);
	SIGNAL state : state_type := one;
	SIGNAL data_reg : std_logic_vector(7 DOWNTO 0) := (OTHERS => '0');
	--SIGNAL in_paralel : std_logic_vector(63 DOWNTO 0) :=
"0100111010100010101101101010111010100111101001101000011011110110 1";
	SIGNAL start : std_logic := '1';
	SIGNAL count : INTEGER := 0;
	SIGNAL data_gen : INTEGER := 0;

BEGIN
	PROCESS (clk)
	BEGIN
		IF reset = '1' THEN

			data_gen <= 0;
			start <= '1';
			state <= one;


		ELSIF rising_edge(clk) THEN

	    -- IF data_gen < 512 THEN
		-- fsm_done <= '0';
	    -- END IF;
	    -- IF fft_done = '1' THEN
		-- fsm_done <= '0';
	    -- END IF;


			IF enable = '1' AND tick3 = '1' THEN
				IF uart_done = '1' OR start = '1' THEN
					-- IF frame_done = '1' THEN
					IF s_fsm_en = '0' THEN
					   fsm_done <= '0';
					else
						CASE state IS
							WHEN one =>
								start <= '0';
								ram_address_b <=
std_logic_vector(to_unsigned(data_gen, 9));
								data_gen <= data_gen + 1;
								data_reg <= in_paralel16(15 DOWNTO
8);
								state <= two;
							WHEN two =>
								data_reg <= in_paralel16(7 DOWNTO 0);
								state <= one;
								IF data_gen > 511 THEN
									fsm_done <= '1';
									data_gen <= 0;
								END IF;
						END CASE;
					END IF;
				END IF;
			END IF;
		END IF;
```

```
        END PROCESS;
        out_paralel <= data_reg;

END Behavioral;
```

design_1_wrapper.vhd

```
--Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------------------------------
--Tool Version: Vivado v.2019.1 (win64) Build 2552052 Fri May 24 14:49:42 MDT 2019
--Date        : Mon Oct 12 18:33:15 2020
--Host        : DESKTOP-SH64B1G running 64-bit major release  (build 9200)
--Command     : generate_target design_1_wrapper.bd
--Design      : design_1_wrapper
--Purpose     : IP block netlist
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity design_1_wrapper is
  port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 13 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_din : in STD_LOGIC_VECTOR ( 63 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC;
    BRAM_PORTA_0_we : in STD_LOGIC_VECTOR ( 0 to 0 );
    BRAM_PORTB_0_addr : in STD_LOGIC_VECTOR ( 13 downto 0 );
    BRAM_PORTB_0_clk : in STD_LOGIC;
    BRAM_PORTB_0_dout : out STD_LOGIC_VECTOR ( 63 downto 0 )
  );
end design_1_wrapper;

architecture STRUCTURE of design_1_wrapper is
  component design_1 is
  port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 13 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_din : in STD_LOGIC_VECTOR ( 63 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC;
    BRAM_PORTA_0_we : in STD_LOGIC_VECTOR ( 0 to 0 );
    BRAM_PORTB_0_addr : in STD_LOGIC_VECTOR ( 13 downto 0 );
    BRAM_PORTB_0_clk : in STD_LOGIC;
    BRAM_PORTB_0_dout : out STD_LOGIC_VECTOR ( 63 downto 0 )
  );
  end component design_1;
begin
design_1_i: component design_1
     port map (
      BRAM_PORTA_0_addr(13 downto 0) => BRAM_PORTA_0_addr(13 downto 0),
      BRAM_PORTA_0_clk => BRAM_PORTA_0_clk,
      BRAM_PORTA_0_din(63 downto 0) => BRAM_PORTA_0_din(63 downto 0),
      BRAM_PORTA_0_en => BRAM_PORTA_0_en,
      BRAM_PORTA_0_we(0) => BRAM_PORTA_0_we(0),
      BRAM_PORTB_0_addr(13 downto 0) => BRAM_PORTB_0_addr(13 downto 0),
      BRAM_PORTB_0_clk => BRAM_PORTB_0_clk,
      BRAM_PORTB_0_dout(63 downto 0) => BRAM_PORTB_0_dout(63 downto 0)
    );
end STRUCTURE;
```

fft_wrapper.vhd

```
--Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------------------------------
--Tool Version: Vivado v.2020.1 (win64) Build 2902540 Wed May 27 19:54:49 MDT 2020
--Date        : Mon Dec 21 15:04:20 2020
--Host        : DESKTOP-MPQPSPI running 64-bit major release  (build 9200)
--Command     : generate_target fft_wrapper.bd
--Design      : fft_wrapper
--Purpose     : IP block netlist
----------------------------------------------------------------------------------
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity fft_wrapper is
  port (
    M_AXIS_DATA_0_tdata : out STD_LOGIC_VECTOR ( 63 downto 0 );
    M_AXIS_DATA_0_tlast : out STD_LOGIC;
    M_AXIS_DATA_0_tvalid : out STD_LOGIC;
    S_AXIS_CONFIG_0_tdata : in STD_LOGIC_VECTOR ( 7 downto 0 );
    S_AXIS_CONFIG_0_tready : out STD_LOGIC;
    S_AXIS_CONFIG_0_tvalid : in STD_LOGIC;
    S_AXIS_DATA_0_tdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    S_AXIS_DATA_0_tlast : in STD_LOGIC;
    S_AXIS_DATA_0_tready : out STD_LOGIC;
    S_AXIS_DATA_0_tvalid : in STD_LOGIC;
    aclk_0 : in STD_LOGIC;
    aresetn_0 : in STD_LOGIC;
    event_data_in_channel_halt_0 : out STD_LOGIC;
    event_frame_started_0 : out STD_LOGIC;
    event_tlast_missing_0 : out STD_LOGIC;
    event_tlast_unexpected_0 : out STD_LOGIC
  );
end fft_wrapper;

architecture STRUCTURE of fft_wrapper is
  component fft is
  port (
    aclk_0 : in STD_LOGIC;
    aresetn_0 : in STD_LOGIC;
    event_data_in_channel_halt_0 : out STD_LOGIC;
    event_frame_started_0 : out STD_LOGIC;
    event_tlast_missing_0 : out STD_LOGIC;
    event_tlast_unexpected_0 : out STD_LOGIC;
    S_AXIS_CONFIG_0_tdata : in STD_LOGIC_VECTOR ( 7 downto 0 );
    S_AXIS_CONFIG_0_tready : out STD_LOGIC;
    S_AXIS_CONFIG_0_tvalid : in STD_LOGIC;
    M_AXIS_DATA_0_tdata : out STD_LOGIC_VECTOR ( 63 downto 0 );
    M_AXIS_DATA_0_tlast : out STD_LOGIC;
    M_AXIS_DATA_0_tvalid : out STD_LOGIC;
    S_AXIS_DATA_0_tdata : in STD_LOGIC_VECTOR ( 31 downto 0 );
    S_AXIS_DATA_0_tlast : in STD_LOGIC;
    S_AXIS_DATA_0_tready : out STD_LOGIC;
    S_AXIS_DATA_0_tvalid : in STD_LOGIC
  );
  end component fft;
begin
fft_i: component fft
     port map (
      M_AXIS_DATA_0_tdata(63 downto 0) => M_AXIS_DATA_0_tdata(63 downto 0),
      M_AXIS_DATA_0_tlast => M_AXIS_DATA_0_tlast,
      M_AXIS_DATA_0_tvalid => M_AXIS_DATA_0_tvalid,
      S_AXIS_CONFIG_0_tdata(7 downto 0) => S_AXIS_CONFIG_0_tdata(7 downto 0),
      S_AXIS_CONFIG_0_tready => S_AXIS_CONFIG_0_tready,
      S_AXIS_CONFIG_0_tvalid => S_AXIS_CONFIG_0_tvalid,
      S_AXIS_DATA_0_tdata(31 downto 0) => S_AXIS_DATA_0_tdata(31 downto 0),
      S_AXIS_DATA_0_tlast => S_AXIS_DATA_0_tlast,
      S_AXIS_DATA_0_tready => S_AXIS_DATA_0_tready,
      S_AXIS_DATA_0_tvalid => S_AXIS_DATA_0_tvalid,
      aclk_0 => aclk_0,
      aresetn_0 => aresetn_0,
      event_data_in_channel_halt_0 => event_data_in_channel_halt_0,
      event_frame_started_0 => event_frame_started_0,
      event_tlast_missing_0 => event_tlast_missing_0,
      event_tlast_unexpected_0 => event_tlast_unexpected_0
    );
end STRUCTURE;
```

fft_ram_wrapper.vhd

```
--Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------------------------------
--Tool Version: Vivado v.2020.1 (win64) Build 2902540 Wed May 27 19:54:49 MDT 2020
```

```
--Date        : Fri Dec 18 17:20:31 2020
--Host        : DESKTOP-MPQPSPI running 64-bit major release  (build 9200)
--Command     : generate_target fft_ram_wrapper.bd
--Design      : fft_ram_wrapper
--Purpose     : IP block netlist
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity fft_ram_wrapper is
  port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_din : in STD_LOGIC_VECTOR ( 15 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC;
    BRAM_PORTA_0_we : in STD_LOGIC_VECTOR ( 0 to 0 );
    BRAM_PORTB_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTB_0_clk : in STD_LOGIC;
    BRAM_PORTB_0_dout : out STD_LOGIC_VECTOR ( 15 downto 0 )
  );
end fft_ram_wrapper;

architecture STRUCTURE of fft_ram_wrapper is
  component fft_ram is
  port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_din : in STD_LOGIC_VECTOR ( 15 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC;
    BRAM_PORTA_0_we : in STD_LOGIC_VECTOR ( 0 to 0 );
    BRAM_PORTB_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTB_0_clk : in STD_LOGIC;
    BRAM_PORTB_0_dout : out STD_LOGIC_VECTOR ( 15 downto 0 )
  );
  end component fft_ram;
begin
fft_ram_i: component fft_ram
     port map (
      BRAM_PORTA_0_addr(8 downto 0) => BRAM_PORTA_0_addr(8 downto 0),
      BRAM_PORTA_0_clk => BRAM_PORTA_0_clk,
      BRAM_PORTA_0_din(15 downto 0) => BRAM_PORTA_0_din(15 downto 0),
      BRAM_PORTA_0_en => BRAM_PORTA_0_en,
      BRAM_PORTA_0_we(0) => BRAM_PORTA_0_we(0),
      BRAM_PORTB_0_addr(8 downto 0) => BRAM_PORTB_0_addr(8 downto 0),
      BRAM_PORTB_0_clk => BRAM_PORTB_0_clk,
      BRAM_PORTB_0_dout(15 downto 0) => BRAM_PORTB_0_dout(15 downto 0)
    );
end STRUCTURE;
```

fft_controller.vhd

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

ENTITY fft_controller IS
       PORT (
               -- GENERAL
               clock : IN std_logic;
               fft_start : IN std_logic;
               fft_done : OUT std_logic := '0';
               s_fft_en : IN std_logic;
               -- frame_done : IN std_logic;

               -- FROM ADC'S RAM
               fft_read_address : OUT std_logic_vector(8 DOWNTO 0);
               fft_read_data : IN std_logic_vector(15 DOWNTO 0);

               -- TO DUAL PORT RAM
               fft_write_address : OUT std_logic_vector(8 DOWNTO 0);
               fft_write_data : OUT std_logic_vector(15 DOWNTO 0);
```

```
		fft_we : OUT std_logic_vector(0 DOWNTO 0);

		-- FFT IP CONFIG PORTS
		ip_config_tvalid : OUT std_logic := '0';
		ip_config_tready : IN std_logic;
		ip_config_tdata : OUT std_logic_vector(7 DOWNTO 0) := "00000001";

		-- FFT IP DATA INPUT PORTS
		ip_in_tvalid : OUT std_logic := '0';
		ip_in_tready : IN std_logic;
		ip_in_tdata : OUT std_logic_vector(31 DOWNTO 0);
		ip_in_tlast : OUT std_logic := '0';

		-- FFT IP DATA OUTPUT PORTS
		ip_out_tvalid : IN std_logic;
		ip_out_tdata : IN std_logic_vector(63 DOWNTO 0)
	);
END fft_controller;

ARCHITECTURE Behavioral OF fft_controller IS
	TYPE state_type IS (read, read_start, write, write_start, idle);
	SIGNAL state : state_type := idle;

	SIGNAL working : std_logic := '0';
	SIGNAL addr_count : INTEGER := 0;
	-- SIGNAL writing : std_logic := '0';
	SIGNAL config_count : INTEGER := 0;

	SIGNAL fft_re : std_logic_vector (15 DOWNTO 0);
	SIGNAL fft_im : std_logic_vector (15 DOWNTO 0);
	SIGNAL fft_mag : std_logic_vector (31 DOWNTO 0);
	SIGNAL temp : std_logic_vector(15 DOWNTO 0);
	SIGNAL count : INTEGER := 0;
BEGIN
	PROCESS (clock)
	BEGIN
		IF rising_edge(clock) THEN
			-- IF frame_done = '1' THEN
			-- fft_done <= '0';
			-- END IF;
			IF count < 10 THEN
				IF s_fft_en = '0' THEN
					fft_done <= '0';
					count <= count + 1;
				ELSE
					count <= 0;
				END IF;

			ELSE
				CASE state IS
					WHEN idle =>
						fft_we <= "0";
						addr_count <= 0;
						state <= read_start;

					WHEN read_start =>
						addr_count <= 0;
						fft_read_address <=
std_logic_vector(to_unsigned(addr_count, fft_read_address'length));
						ip_in_tvalid <= '0';
						state <= read;
						fft_we <= "0";
						ip_in_tlast <= '0';

					WHEN read =>
						ip_in_tvalid <= '1';
						IF ip_in_tready = '1' THEN
							fft_read_address <=
std_logic_vector(to_unsigned(addr_count, fft_read_address'length));
							ip_in_tdata <= "0000000000000000" &
fft_read_data;
```

```
                                                IF addr_count > 510 THEN
                                                        ip_in_tlast <= '1';
                                                END IF;

                                                IF addr_count > 511 THEN
                                                        ip_in_tvalid <= '0';
                                                        state <= write_start;
                                                ELSE
                                                        addr_count <= addr_count + 1;
                                                END IF;

                                                -- IF addr_count < 256 THEN
                                                --addr_count <= addr_count + 1;
                                                -- END IF;
                                        END IF;

                                WHEN write_start =>
                                        addr_count <= 0;
                                        state <= write;

                                WHEN write =>
                                        fft_we <= "1";
                                        fft_write_address <=
std_logic_vector(to_unsigned(addr_count, fft_write_address'length));
                                        -- fft_re <= ip_out_tdata(25 DOWNTO 10);
                                        -- fft_im <= ip_out_tdata(57 DOWNTO 42);

                                        -- fft_re <= ip_out_tdata(25 DOWNTO 10);
                                        -- IF fft_re(15) = '1' THEN
                                                -- temp <= NOT fft_re;
                                                -- fft_write_data <=
std_logic_vector(unsigned(temp) + 1);
                                        -- ELSE
                                                -- fft_write_data <= fft_re;
                                        -- END IF;


                        fft_write_data(7 downto 0) <= ip_out_tdata(25 DOWNTO 18);
                                          fft_write_data(15 downto 8) <= ip_out_tdata(57 DOWNTO
50);

                                        IF ip_out_tvalid = '1' THEN
                                                addr_count <= addr_count + 1;
                                        END IF;

                                        IF addr_count > 511 THEN
                                                state <= idle;
                                                fft_done <= '1';
                                        END IF;

                        END CASE;
                END IF;
        END IF;
    END PROCESS;
END Behavioral;
```

design 3 wrapper.vhd

```
--Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------------------------------
--Tool Version: Vivado v.2020.1 (win64) Build 2902540 Wed May 27 19:54:49 MDT 2020
--Date        : Fri Dec 18 17:20:30 2020
--Host        : DESKTOP-MPQPSPI running 64-bit major release  (build 9200)
--Command     : generate_target design_3_wrapper.bd
--Design      : design_3_wrapper
--Purpose     : IP block netlist
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity design_3_wrapper is
```

```
  port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_din : in STD_LOGIC_VECTOR ( 15 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC;
    BRAM_PORTA_0_we : in STD_LOGIC_VECTOR ( 0 to 0 );
    BRAM_PORTB_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTB_0_clk : in STD_LOGIC;
    BRAM_PORTB_0_dout : out STD_LOGIC_VECTOR ( 15 downto 0 )
  );
end design_3_wrapper;

architecture STRUCTURE of design_3_wrapper is
  component design_3 is
  port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_din : in STD_LOGIC_VECTOR ( 15 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC;
    BRAM_PORTA_0_we : in STD_LOGIC_VECTOR ( 0 to 0 );
    BRAM_PORTB_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTB_0_clk : in STD_LOGIC;
    BRAM_PORTB_0_dout : out STD_LOGIC_VECTOR ( 15 downto 0 )
  );
  end component design_3;
begin
design_3_i: component design_3
     port map (
      BRAM_PORTA_0_addr(8 downto 0) => BRAM_PORTA_0_addr(8 downto 0),
      BRAM_PORTA_0_clk => BRAM_PORTA_0_clk,
      BRAM_PORTA_0_din(15 downto 0) => BRAM_PORTA_0_din(15 downto 0),
      BRAM_PORTA_0_en => BRAM_PORTA_0_en,
      BRAM_PORTA_0_we(0) => BRAM_PORTA_0_we(0),
      BRAM_PORTB_0_addr(8 downto 0) => BRAM_PORTB_0_addr(8 downto 0),
      BRAM_PORTB_0_clk => BRAM_PORTB_0_clk,
      BRAM_PORTB_0_dout(15 downto 0) => BRAM_PORTB_0_dout(15 downto 0)
    );
end STRUCTURE;
```

XADCdemo.v

```
`timescale 1ns / 1ps
module XADCdemo(
   input CLK100MHZ,
   input vauxp6,
   input vauxn6,
   output reg [15:0] LED
 );

   wire enable;
   wire ready;
   wire [15:0] data;
   reg [6:0] Address_in;

   //xadc instantiation connect the eoc_out .den_in to get continuous conversion
   xadc_wiz_0  XLXI_7 (.daddr_in(Address_in), //addresses can be found in the artix 7 XADC user
guide DRP register space
                     .dclk_in(CLK100MHZ),
                     .den_in(enable),
                     .di_in(),
                     .dwe_in(),
                     .busy_out(),
                     .vauxp6(vauxp6),
                     .vauxn6(vauxn6),
                     .vn_in(),
                     .vp_in(),
                     .alarm_out(),
                     .do_out(data),
                     //.reset_in(),
                     .eoc_out(enable),
                     .channel_out(),
                     .drdy_out(ready));
```

```
      always @( posedge(CLK100MHZ))
      begin
        if(ready == 1'b1)
        begin
        LED <= data;
//          case (data[15:12])
//            1:  LED <= 16'b11;
//            2:  LED <= 16'b111;
//            3:  LED <= 16'b1111;
//            4:  LED <= 16'b11111;
//            5:  LED <= 16'b111111;
//            6:  LED <= 16'b1111111;
//            7:  LED <= 16'b11111111;
//            8:  LED <= 16'b111111111;
//            9:  LED <= 16'b1111111111;
//            10: LED <= 16'b11111111111;
//            11: LED <= 16'b111111111111;
//            12: LED <= 16'b1111111111111;
//            13: LED <= 16'b11111111111111;
//            14: LED <= 16'b111111111111111;
//            15: LED <= 16'b1111111111111111;
//            default: LED <= 16'b1;
//           endcase
        end
        Address_in <= 8'h16;
      end
endmodule
```

adc_to_ram.vhd

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

ENTITY adc_to_ram IS
       PORT (
              clk : IN std_logic;
              reset : IN std_logic;
              enable_adc_to_ram : IN std_logic;
              ram_write_data : OUT std_logic_vector(63 DOWNTO 0);
              ram_write_addr : OUT std_logic_vector(13 DOWNTO 0);
              ram_address_b : OUT std_logic_vector(13 DOWNTO 0);
              ram_write_en : OUT std_logic_vector(0 DOWNTO 0);
              hann_ram_we : OUT std_logic_vector(0 DOWNTO 0);
              dram_read : IN std_logic_vector(63 DOWNTO 0);
              hann_rom_data : IN std_logic_vector(15 DOWNTO 0);
              hann_rom_addr : OUT std_logic_vector(8 DOWNTO 0);
              hann_ram_data : OUT std_logic_vector(15 DOWNTO 0);
              hann_ram_addr : OUT std_logic_vector(8 DOWNTO 0);
              adc_data : IN std_logic_vector(15 DOWNTO 0);
              adc_to_ram_done : OUT std_logic;
              -- FSM_DONE : IN STD_LOGIC;
              -- fft_done : in std_logic;

              frame_done : OUT std_logic := '0';
              s_hann_en : in std_logic;
              all_frames_done : OUT std_logic := '0'
       );
END adc_to_ram;

ARCHITECTURE Behavioral OF adc_to_ram IS
       SIGNAL count : INTEGER := 0;
       SIGNAL ram_addr : INTEGER := 0;
       SIGNAL operand1 : INTEGER := 0;
       SIGNAL operand1_temp : std_logic_vector(15 DOWNTO 0);
       SIGNAL operand2 : INTEGER := 0;
       SIGNAL mul_output : INTEGER := 0;
       SIGNAL mul_output_temp : std_logic_vector(31 DOWNTO 0);
       -- signal ram_part : integer := 0;
       SIGNAL adc_to_ram_done_sig : std_logic := '0';
```

```
        SIGNAL count2 : INTEGER := 0;
        SIGNAL count2addr : INTEGER := 0;
        SIGNAL count3 : INTEGER := 0;
        SIGNAL countromaddr : INTEGER := 0;

        SIGNAL int1 : INTEGER := 0;
        SIGNAL int2 : INTEGER := 0;

BEGIN
        PROCESS (clk)
        BEGIN
                IF reset = '1' THEN
                        count <= 0;
                        ram_addr <= 0;
                        ram_write_en <= "0";
                        -- adc_to_ram_done <= '0';
                        adc_to_ram_done_sig <= '0';
                ELSIF rising_edge(clk) THEN
                        IF enable_adc_to_ram = '1' THEN
                                IF count < 6250 THEN
                                        count <= count + 1;
                                ELSE
                                        IF ram_addr < 16384 - 1 THEN
                                                ram_write_en <= "1";
                                                ram_write_addr <=
std_logic_vector(to_unsigned(ram_addr, 14));

                                                -- ram_write_data(15 DOWNTO 0) <= adc_data;

                                                int1 <= to_integer(unsigned(adc_data));
                                                int2 <= int1 - 32768;
                                                ram_write_data(15 DOWNTO 0) <=
std_logic_vector(to_signed(int2, 16));
                                                ram_write_data(63 DOWNTO 16) <= (OTHERS => '0');

                                                count <= 0;
                                                ram_addr <= ram_addr + 1;
                                                -- adc_to_ram_done <= '0';
                                                adc_to_ram_done_sig <= '0';
                                        ELSE
                                                adc_to_ram_done_sig <= '1';
                                                -- adc_to_ram_done <= '1';
                                                ram_write_en <= "0";
                                        END IF;

                                END IF;
                        -- ELSE
                                -- adc_to_ram_done_sig <= '0';
                        END IF;

                        IF adc_to_ram_done_sig = '1' THEN

                                IF count3 < 63 THEN
                                        IF count2 < 1 THEN
                                                count2addr <= count3 * 256;
                                        END IF;

                    -- IF FSM_DONE = '1' THEN
                        -- frame_done <= '0';
                    -- END IF;
                                        -- IF count3 = 0 or frame_done = '0' THEN
                                        -- IF count3 = 0 THEN


                                        if s_hann_en = '0' then
                                           frame_done <= '0';

                                        else
                                                IF count2 < 512 * 7 THEN

                                                        -- frame_done <= '0';
                                                        hann_ram_we <= "1";
```

```
                                                    ram_address_b <=
std_logic_vector(to_unsigned(count2addr, 14));
                                                    hann_rom_addr <=
std_logic_vector(to_unsigned(countromaddr, 9));
                                                    operand1_temp <= dram_read(15 DOWNTO 0);
                                                    -- operand1 <=
to_integer(unsigned(operand1_temp));
                                                    operand1 <=
to_integer(signed(operand1_temp));
                                                    operand2 <=
to_integer(unsigned(hann_rom_data));
                                                    -- mul_output <= operand1 * operand2;
                                                    mul_output <= operand1 * operand2 / 2;

                                                    -- mul_output_temp <=
std_logic_vector(to_unsigned(mul_output, 32));
                                                    mul_output_temp <=
std_logic_vector(to_signed(mul_output, 32));

                                                    hann_ram_data <= mul_output_temp(31 DOWNTO
16);
                                                    hann_ram_addr <=
std_logic_vector(to_unsigned(countromaddr, 9));

                                                    IF (count2 MOD 7) = 6 THEN
                                                           count2addr <= count2addr + 1;
                                                           countromaddr <= countromaddr + 1;
                                                    END IF;
                                                    count2 <= count2 + 1;
                                             ELSE
                                                    frame_done <= '1';
                                                    count3 <= count3 + 1;
                                                    count2 <= 0;
                                                    countromaddr <= 0;
                                                    hann_ram_we <= "0";
                                             END IF;
                                      END IF;
                               ELSE
                                      all_frames_done <= '1';
                               END IF;
                        END IF;
                 END IF;
        END PROCESS;

        adc_to_ram_done <= adc_to_ram_done_sig;
END Behavioral;
```

design_2_wrapper.vhd

```
--Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------------------------------
--Tool Version: Vivado v.2020.1 (win64) Build 2902540 Wed May 27 19:54:49 MDT 2020
--Date        : Fri Dec 18 17:20:29 2020
--Host        : DESKTOP-MPQPSPI running 64-bit major release  (build 9200)
--Command     : generate_target design_2_wrapper.bd
--Design      : design_2_wrapper
--Purpose     : IP block netlist
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity design_2_wrapper is
  port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_dout : out STD_LOGIC_VECTOR ( 15 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC
  );
end design_2_wrapper;
```

```
architecture STRUCTURE of design_2_wrapper is
  component design_2 is
  port (
    BRAM_PORTA_0_addr : in STD_LOGIC_VECTOR ( 8 downto 0 );
    BRAM_PORTA_0_clk : in STD_LOGIC;
    BRAM_PORTA_0_dout : out STD_LOGIC_VECTOR ( 15 downto 0 );
    BRAM_PORTA_0_en : in STD_LOGIC
  );
  end component design_2;
begin
design_2_i: component design_2
     port map (
      BRAM_PORTA_0_addr(8 downto 0) => BRAM_PORTA_0_addr(8 downto 0),
      BRAM_PORTA_0_clk => BRAM_PORTA_0_clk,
      BRAM_PORTA_0_dout(15 downto 0) => BRAM_PORTA_0_dout(15 downto 0),
      BRAM_PORTA_0_en => BRAM_PORTA_0_en
    );
end STRUCTURE;
```

8.2 MATLAB Codes

lab_debug_matlab.m

```
clc;
% clear all;
close all; delete(instrfind);
s = serial('COM6'); % Modify COM according to your COM port
nofElem=16384*2; %Number of receive samples
%nofElem=512*2*4;
decdata=uint8(nofElem); % integer data
set(s,'BaudRate',9600); % MODIFY FOR YOUR BAUDRATE
set(s,'InputBufferSize',1); %number of bytes
set(s,'OutputBufferSize',1); % number of bytes
get(s) % Properties of your serial port


k=1;
fopen(s);
%fread(s,1);
%fread(s,1);
while(1)
decdata(k) = fread(s,1);
if k==nofElem
break;
end
k=k+1;
disp(k);
end
get(s);
fclose(s);

data = reshape(decdata, [2, length(decdata) / 2]);

decimals = zeros(1, length(decdata) / 2);
for i = 1:(length(decdata) / 2)
     a = zeros(2, 8);
     a = de2bi(transpose(data(:, i)));
     decimals(i) = bi2de(flip(reshape(transpose(flip(de2bi(transpose(data(:,
i)), 8), 2)), [1, 16])));
```

```
end

figure;
plot(decimals);
%
% V1 = 2^16;
% digital = decimals / V1;
% f = 16*10^3;
% f1 = 100;
%
% t = 0:1/f:1000/f;
% t = t(1:end-1);
% analog = 0.084 + 0.700*(1 + sin(2*pi*f1*t))/2;
```

mel_cs.m

```
function cs = mel_cs( sig )

L = 512;
N = 63;

window = floor( hann(512)' .* (2^16));

A = [5 13 26 44 67 95 128];
B = [8 13 18 23 28 33 38];
C = [235 217 194 166 133 95 52];

for i = 1:7
    filt(i, :) = [zeros(1, A(i)) linspace(0, 1, B(i)) linspace(1, 0, B(i))
zeros(1, C(i))];
end

j = 1;
for i=1:N
    frame(i, :) = sig(j:j+L-1);
    j = j + L / 2;
end

for i=1:N
    frame_win = frame(i,:) .* window;
    frame_fft = fft(frame_win);
    frame_fft_abs(i, :) = abs(floor(frame_fft ./ 2^16));
end

for i=1:N
    energy = [];
    for t=1:7
        fft_filt = frame_fft_abs(i, 1:256) .* filt(t, :).^2;
        energy1 = sum(floor(fft_filt));
        energy = [energy energy1];
    end
    frame_E(i,:) = energy;
end

dct2 = [];
for i=1:N
```

```
    E = frame_E(i,:);
    log_E = floor(log10(E + 1) .* 128);
    log_E_dct = dct(log_E);
    log_E_dct(1) = 0;
    dct1 = floor(log_E_dct);
    dct2 = [dct2 dct1];
end

cs = dct2;
end
```

guess.m

```
% clear all;
close all;
cutoff = 100;
rec_length = 3;
Fs = 16000;
load("samplesound3.mat")
load("data.mat");
decimals = zeros(1, 16384);
% decimals1 = decimals - 2^15;
decimals1 = decimals - mean(decimals);
decimals2 = decimals1 / (32767 * 5);
sig = decimals2;
sig_filt = sig(1:end-1) - sig(2:end);


sig_filt1 = sig_filt .* (2^13) + 2^13;
sig_int = floor(sig_filt1);

sig_int_mean = mean(sig_int);
sig_cut = ones(1, 2^14) .* sig_int_mean;

j = 0;
k = 1;
cut_en = false;
for i=1:length(sig_int)
    if cut_en
        if k < 2^14 + 1
            sig_cut(k) = sig_int(i);
            k = k + 1;
        else
            break;
        end
    else
        if abs(sig_int(i) - sig_int_mean) > cutoff
            j = j + 1;
        end

        if j > 10
            cut_en = true;
        end
    end
end
```

```
figure;
plot(sig_cut);

% sig_cut = samplesound2(24, :);
soundsc(sig_cut,Fs);
c_vector = mel_cs(sig_cut);


% for i=1:50
% for i=1:20
    % train_cs(i,:) = mel_cs(samplesound3(i,:));
% end

% for i=1:50
for i=1:20
    % mses(i) = sum( (train_cs(i,:) - c_vector).^2 );
    mses(i) = dtw(train_cs(i,:), c_vector);
end


[~, index] = min(mses);
% [~, index] = max(mses);

% num = ceil(index / 5);
num = mod(index - 1, 10);
% num = index;
disp("Guess:");
disp(num);
```

adc_analysis2.m

```
f1 = 1000;
% f = 1000;
t = 0:1/f:1000/f;
t = t(1:end-1);
analog = 0.116 + 0.800*(1 + sin(2*pi*f1*t))/2;

digital = decimals;
% remove phase difference
i1 = find((digital / 2^16) > 0.5, 1);
i2 = find(analog > 0.5, 1);
digital_1 = digital(i1:end);
analog_1 = analog(i2:end);

s1 = length(digital_1);
s2 = length(analog_1);
s3 = min(s1, s2);

digital_2 = digital_1(1:s3);
analog_2 = analog_1(1:s3);




figure;
hold on;
```

```
plot(digital_2 / 2^16);
plot(analog_2);
legend("Digital", "Analog");
xlabel("Time")
ylabel("Voltage")


a = inldnl(analog_2(200:280), digital_2(200:280), [0, 1], "ADC");

digital_3 = digital_2 / 2^16;

noise = abs(analog_2 - (digital_3));
b = snr(analog_2, noise);
disp("Signal to noise ratio (dB):");
disp(b);
dist = thd(digital_3);
disp("Total harmonic distortion (percentage):");
disp(100*(10^(dist/20)));
disp("Spurious free dynamic range (dB):");
disp(sfdr(digital_3));

sinad1 = sinad(digital_3, 16000);
enob = (sinad1 - 1.76) / 6.02;
disp("Effective number of bits:");
disp(enob);

disp("Full power bandwidth:");
disp(powerbw(digital_3, 16000));

disp("Offset error:")
disp(a.OffsetError)

disp("Gain error:")
disp(a.GainError)

figure;
plot(a.INL);
title("INL")

figure;
plot(a.DNL);
title("DNL");
```

debug_fft.m

```
clc;clear all; close all; delete(instrfind);
s = serial('COM8'); % Modify COM according to your COM port
%nofElem=16384*8; %Number of receive samples
% nofElem=512*2*63;
frame_count = 16;
nofElem=512*2*frame_count;
decdata=uint8(nofElem); % integer data
set(s,'BaudRate',9600); % MODIFY FOR YOUR BAUDRATE
set(s,'InputBufferSize',1); %number of bytes
set(s,'OutputBufferSize',1); % number of bytes
get(s) % Properties of your serial port
```

```
Fs = 16000;

k=1;
fopen(s);
%fread(s,1);
%fread(s,1);
while(1)
    decdata(k) = fread(s,1);
    if k==nofElem
        break;
    end
    k=k+1;
    disp(k);
end
get(s);
fclose(s);

data = reshape(decdata, [2, length(decdata) / 2]);

decimals = zeros(1, length(decdata) / 2);
for i = 1:(length(decdata) / 2)
    decimals(i) = bi2de(flip(reshape(transpose(flip(de2bi(transpose(data(:,
i)), 8), 2)), [1, 16])));
end

% figure;
% plot(decimals);


for k = 0:(frame_count-1)
    a = decimals(1+(512*k):512+(512*k));
    B = de2bi(a);
    B_real = B(:, 1:8);
    B_imag = B(:, 9:16);
    C_real = zeros(512, 8);
    C_imag = zeros(512, 8);
    for i = 1:512
        if B_real(i, 1) == 1
            C_real(i, :) = (B_real(i, :) == 0);
        else
            C_real(i, :) = B_real(i, :);
        end
        if B_imag(i, 1) == 1
            C_imag(i, :) = (B_imag(i, :) == 0);
        else
            C_imag(i, :) = B_imag(i, :);
        end
    end
    re = bi2de(C_real);
    im = bi2de(C_imag);

    re2 = re .* re;
    im2 = im .* im;

    total = re2 + im2;
    mag = sqrt(total);
    figure;
    n = 512;
    dF = Fs/n;
    f = -Fs/2:dF:Fs/2;
    mag1 = fftshift(mag);
    plot(f(1:512), mag1/n);
end
```

8.3 Hamming Window Coefficients (.coe file)

```
memory_initialization_radix=2;

memory_initialization_vector=0000000000000000 0000000000000010 0000000000001010
0000000000010110 0000000000101000 0000000000111110 0000000001011001 0000000001111001
0000000010011110 0000000011001000 0000000011110111 0000000100101011 0000000101100100
0000000110100010 0000000111100100 0000001000101100 0000001001111000 0000001011001001
0000001100011111 0000001101111010 0000001111011010 0000010000111110 0000010010101000
0000010100010110 0000010110001000 0000011000000000 0000011001111100 0000011011111101
0000011110000011 0000100000001101 0000100010011100 0000100100110000 0000100111001000
0000101001100101 0000101100000110 0000101110101100 0000110001010110 0000110100000101
0000110110111000 0000111001110000 0000111100101100 0000111111101101 0001000010110001
0001000101111010 0001001001001000 0001001100011001 0001001111101111 0001010011001001
0001010110100111 0001011010001010 0001011101110000 0001100001011010 0001100101001001
0001101000111011 0001101100110010 0001110000101100 0001110100101010 0001111000101100
0001111100110010 0010000000111011 0010000101001000 0010001001011001 0010001101101110
0010010010000110 0010010110100001 0010011011000000 0010011111100011 0010100100001001
0010101000110010 0010101101011111 0010110010001111 0010110111000010 0010111011111000
0011000000110001 0011000101101110 0011001010101110 0011001111110000 0011010100110110
0011011001111110 0011011111001001 0011100100010111 0011101001101000 0011101110111100
0011110100010010 0011111001101011 0011111111000110 0100000100100100 0100001010000100
0100001111100110 0100010101001011 0100011010110010 0100100000011100 0100100110000111
0100101011110101 0100110001100101 0100110111010110 0100111101001010 0101000011000000
0101001000110111 0101001110110000 0101010100101011 0101011010100111 0101100000100110
0101100110100101 0101101100100110 0101110010101001 0101111000101101 0101111110110010
0110000100111000 0110001011000000 0110010001001001 0110010111010011 0110011101011110
0110100011101010 0110101001110110 0110110000000100 0110110110010010 0110111100100001
0111000010110001 0111001001000001 0111001111010010 0111010101100011 0111011011110101
0111100010000111 0111101000011010 0111101110101100 0111110100111111 0111111011010010
1000000001100101 1000000111111000 1000001110001010 1000010100011101 1000011010110000
1000100001000010 1000100111010100 1000101101100101 1000110011110110 1000111010000111
1001000000010111 1001000110100110 1001001100110101 1001010011000011 1001011001010000
1001011111011100 1001100101101000 1001101011110010 1001110001111100 1001111000000100
1001111110001011 1010000100010001 1010001010010101 1010010000011001 1010010110011010
1010011100011011 1010100010011010 1010101000010111 1010101110010011 1010110100001101
1010111010000101 1010111111111011 1011000101110000 1011001011100011 1011010001010011
1011010111000010 1011011100101111 1011100010011001 1011101000000001 1011101101100111
1011110011001011 1011111000101101 1011111110001100 1100000011101000 1100001001000010
```

```
1100001110011010 1100010011101110 1100011001000000 1100011110010000 1100100011011101
1100101000100110 1100101101101101 1100110010110001 1100110111110011 1100111100110001
1101000001101100 1101000110100100 1101001011011000 1101010000001010 1101010100111000
1101011001100011 1101011110001011 1101100010101111 1101100111010000 1101101011101101
1101110000000111 1101110100011101 1101111000110000 1101111100111111 1110000001001010
1110000101010010 1110001001010110 1110001101010110 1110010001010010 1110010101001010
1110011000111110 1110011100101111 1110100000011011 1110100100000100 1110100111101000
1110101011001000 1110101110100100 1110110001111100 1110110101010000 1110111000011111
1110111011101011 1110111110110010 1111000001110100 1111000100110011 1111000111101100
1111001010100010 1111001101010011 1111010000000000 1111010010101000 1111010101001011
1111010111101010 1111011010000101 1111011100011011 1111011110101100 1111100000111001
1111100011000000 1111100101000100 1111100111000010 1111101000111100 1111101010110010
1111101100100010 1111101110001110 1111101111110101 1111110001010111 1111110010110100
1111110100001100 1111110101100000 1111110110101111 1111110111111001 1111111000111110
1111111001111110 1111111010111001 1111111011101111 1111111100100001 1111111101001101
1111111101110101 1111111110010111 1111111110110101 1111111111001110 1111111111100010
1111111111110001 1111111111111010 1111111111111111 1111111111111111 1111111111111010
1111111111110001 1111111111100010 1111111111001110 1111111110110101 1111111110010111
1111111101110101 1111111101001101 1111111100100001 1111111011101111 1111111010111001
1111111001111110 1111111000111110 1111110111111001 1111110110101111 1111110101100000
1111110100001100 1111110010110100 1111110001010111 1111101111110101 1111101110001110
1111101100100010 1111101010110010 1111101000111100 1111100111000010 1111100101000100
1111100011000000 1111100000111001 1111011110101100 1111011100011011 1111011010000101
1111010111101010 1111010101001011 1111010010101000 1111010000000000 1111001101010011
1111001010100010 1111000111101100 1111000100110011 1111000001110100 1110111110110010
1110111011101011 1110111000011111 1110110101010000 1110110001111100 1110101110100100
1110101011001000 1110100111101000 1110100100000100 1110100000011011 1110011100101111
1110011000111110 1110010101001010 1110010001010010 1110001101010110 1110001001010110
1110000101010010 1110000001001010 1101111100111111 1101111000110000 1101110100011101
1101110000000111 1101101011101101 1101100111010000 1101100010101111 1101011110001011
1101011001100011 1101010100111000 1101010000001010 1101001011011000 1101000110100100
1101000001101100 1100111100110001 1100110111110011 1100110010110001 1100101101101101
1100101000100110 1100100011011101 1100011110010000 1100011001000000 1100010011101110
1100001110011010 1100001001000010 1100000011101000 1011111110001100 1011111000101101
1011110011001011 1011101101100111 1011101000000001 1011100010011001 1011011100101111
1011010111000010 1011010001010011 1011001011100011 1011000101110000 1010111111111011
1010111010000101 1010110100001101 1010101110010011 1010101000010111 1010100010011010
1010011100011011 1010010110011010 1010010000011001 1010001010010101 1010000100010001
1001111110001011 1001111000000100 1001110001111100 1001101011110010 1001100101101000
1001011111011100 1001011001010000 1001010011000011 1001001100110101 1001000110100110
1001000000010111 1000111010000111 1000110011110110 1000101101100101 1000100111010100
1000100001000010 1000011010110000 1000010100011101 1000001110001010 1000000111111000
1000000001100101 0111111011010010 0111110100111111 0111101110101100 0111101000011010
0111100010000111 0111011011110101 0111010101100011 0111001111010010 0111001001000001
0111000010110001 0110111100100001 0110110110010010 0110110000000100 0110101001110110
```

```
0110100011101010 0110011101011110 0110010111010011 0110010001001001 0110001011000000
0110000100111000 0101111110110010 0101111000101101 0101110010101001 0101101100100110
0101100110100101 0101100000100110 0101011010100111 0101010100101011 0101001110110000
0101001000110111 0101000011000000 0100111101001010 0100110111010110 0100110001100101
0100101011110101 0100100110000111 0100100000011100 0100011010110010 0100010101001011
0100001111100110 0100001010000100 0100000100100100 0011111111000110 0011111001101011
0011110100010010 0011101110111100 0011101001101000 0011100100010111 0011011111001001
0011011001111110 0011010100110110 0011001111110000 0011001010101110 0011000101101110
0011000000110001 0010111011111000 0010110111000010 0010110010001111 0010101101011111
0010101000110010 0010100100001001 0010011111100011 0010011011000000 0010010110100001
0010010010000110 0010001101101110 0010001001011001 0010000101001000 0010000000111011
0001111100110010 0001111000101100 0001110100101010 0001110000101100 0001101100110010
0001101000111011 0001100101001001 0001100001011010 0001011101110000 0001011010001010
0001010110100111 0001010011001001 0001001111101111 0001001100011001 0001001001001000
0001000101111010 0001000010110001 0000111111101101 0000111100101100 0000111001110000
0000110110111000 0000110100000101 0000110001010110 0000101110101100 0000101100000110
0000101001100101 0000100111001000 0000100100110000 0000100010011100 0000100000001101
0000011110000011 0000011011111101 0000011001111100 0000011000000000 0000010110001000
0000010100010110 0000010010101000 0000010000111110 0000001111011010 0000001101111010
0000001100011111 0000001011001001 0000001001111000 0000001000101100 0000000111100100
0000000110100010 0000000101100100 0000000100101011 0000000011110111 0000000011001000
0000000010011110 0000000001111001 0000000001011001 0000000000111110 0000000000101000
0000000000010110 0000000000001010 0000000000000010 0000000000000000;
```