Khyle Calpe
405016683

EE113D: Digital Signal Processing Design

Lab 3: Fast Fourier Transform

OBJECTIVE

The objective of this lab is to perform Fourier Transforms on time signals at various shapes and frequencies. The signals are generated via a function generator and then measured via an oscilloscope through the AD2. The signals are fed into the STM32 for sampling and processing.

PART 1

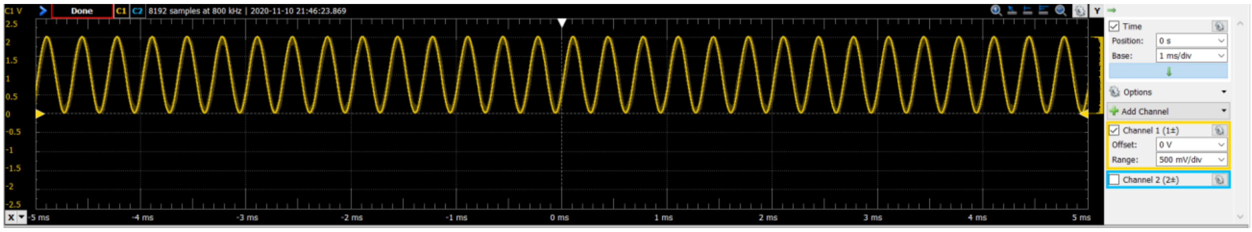

Figure 1.1: Oscilloscope reading of the input and output waveform.

CODE:

```
// Record samples
if(HAL_ADC_Start_DMA(&hadc1, (uint32_t *)&ADC_buff_1024, ADC_BUF_SIZE_1024) != HAL_OK) {
  Error_Handler();
}
while (1)
{
  if(record_done == 1){
    record_done = 0;
    break;
  }
}

// Print first 5 samples
for (int i=0; i<5; i++)
{
  printf("#%d: %d\n", i+1, ADC_buff_1024[i]);
  HAL_Delay(10);
}
// Plot sample
for (int i=0; i<ADC_BUF_SIZE_1024; i++)
{
  sample=ADC_buff_1024[i];
  HAL_Delay(10);
}

// Compute the array mean
for (int i=0; i<ADC_BUF_SIZE_1024; i++)
{
  mean += ADC_buff_1024[i];
}
mean /= ADC_BUF_SIZE_1024;
// Subtract the array mean from each element
for (int i=0; i<ADC_BUF_SIZE_1024; i++)
{
  ADC_buff_1024[i] -= mean;
}
```

PART 2

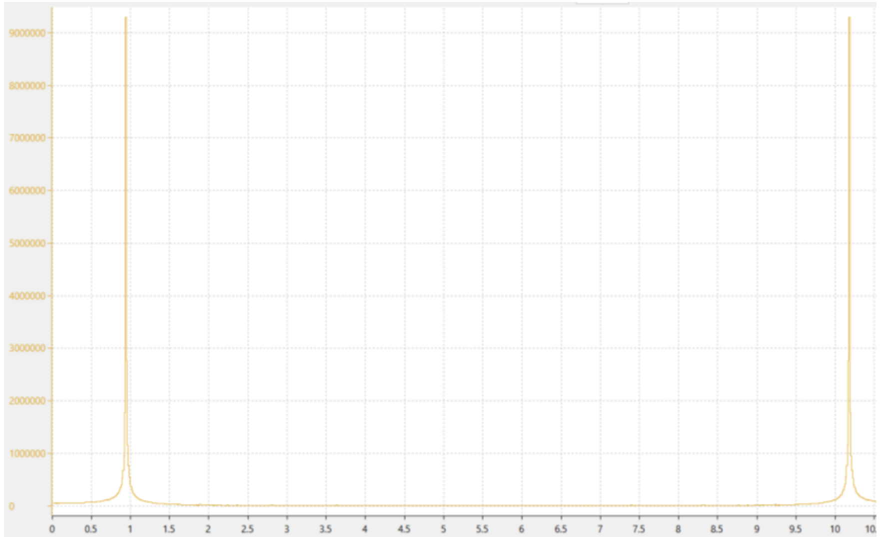

Figure 2.1: Screenshot of the FFT of a 3KHz sine wave.

Figure 2.2: Screenshot of the FFT of a 4KHz sine wave. Δf is calculated to be 46 samples. The leftmost peak is calculated at 4125 Hz.

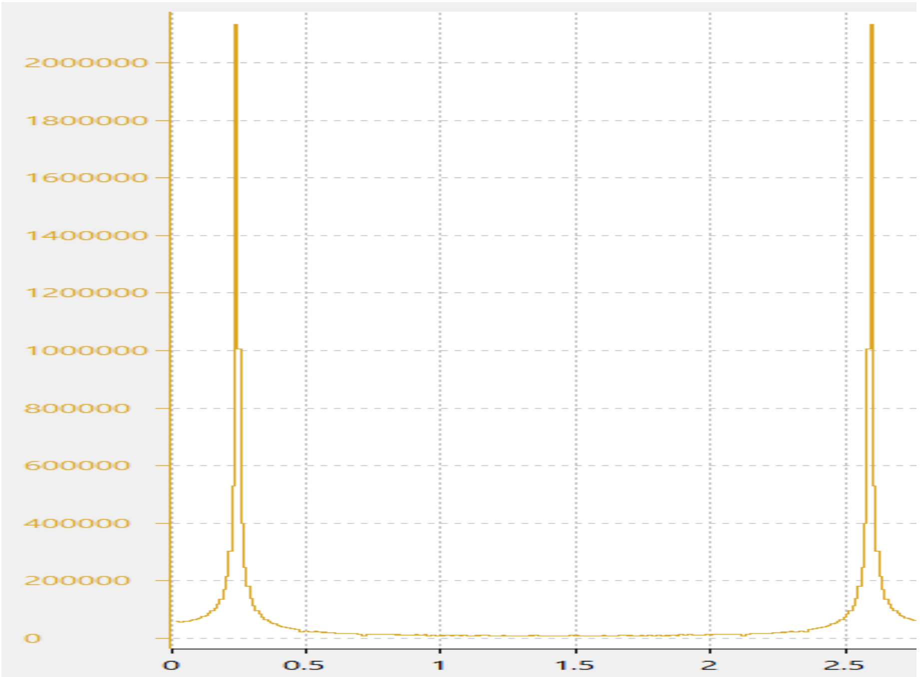

Figure 2.3: Screenshot of the same FFT for 256 samples. Δf is calculated to be 187 samples. The leftmost peak is calculated at 4125 Hz.

Comparing the FFT plots of the 4KHz sine wave at 1024 samples and 256 samples, the primary difference is the frequency resolution. Since the 1024-point FFT contains more samples than the 256-point FFT, the 1024-point FFT provides a denser plot of the frequency signal.

CODE:

```c
// FFT Instances
arm_cfft_instance_f32 fft_handler_1024;
arm_cfft_instance_f32 fft_handler_256;
arm_cfft_init_f32(&fft_handler_1024, 1024);
arm_cfft_init_f32(&fft_handler_256, 256);

// Complex FFT of 3KHz sample
convert_to_complex(ADC_buff_1024, CMPLX_buff_1024, 1024);
// FFT of 3KHz sample
arm_cfft_f32(&fft_handler_1024, CMPLX_buff_1024, 0, 1);
// Print magnitude of FFT
set_magnitude(CMPLX_buff_1024, MGNTD_buff_1024, 1024);
print_magnitude(MGNTD_buff_1024, 1024);
// Record sample of 4KHz sine wave
if(HAL_ADC_Start_DMA(&hadc1, (uint32_t *)&ADC_buff_1024, ADC_BUF_SIZE_1024) != HAL_OK) {
  Error_Handler();
}
while (1)
{
  if(record_done == 1){
    record_done = 0;
    break;
  }
}
// Complex FFT of 4KHz sample
convert_to_complex(ADC_buff_1024, CMPLX_buff_1024, 1024);
// FFT of 4KHz sample
arm_cfft_f32(&fft_handler_1024, CMPLX_buff_1024, 0, 1);
// Print magnitude of FFT
set_magnitude(CMPLX_buff_1024, MGNTD_buff_1024, 1024);
print_magnitude(MGNTD_buff_1024, 1024);

// Record sample of 4KHz sine wave for 256 samples
if(HAL_ADC_Start_DMA(&hadc1, (uint32_t *)&ADC_buff_256, ADC_BUF_SIZE_256) != HAL_OK) {
  Error_Handler();
}
while (1)
{
  if(record_done == 1){
    record_done = 0;
    break;
  }
}

// Complex FFT of 4KHz sample for 256 samples
convert_to_complex(ADC_buff_256, CMPLX_buff_256, 256);
// FFT of 4KHz sample for 256 samples
arm_cfft_f32(&fft_handler_256, CMPLX_buff_256, 0, 1);
// Print magnitude of FFT
set_magnitude(CMPLX_buff_256, MGNTD_buff_256, 256);
print_magnitude(MGNTD_buff_256, 256);
// Calculate delta f
printf("delta f for %d samples: %d\n", 1024, get_d_f(1024, 48e3));
printf("delta f for %d samples: %d\n", 1024, get_d_f(256, 48e3));
```

PART 3

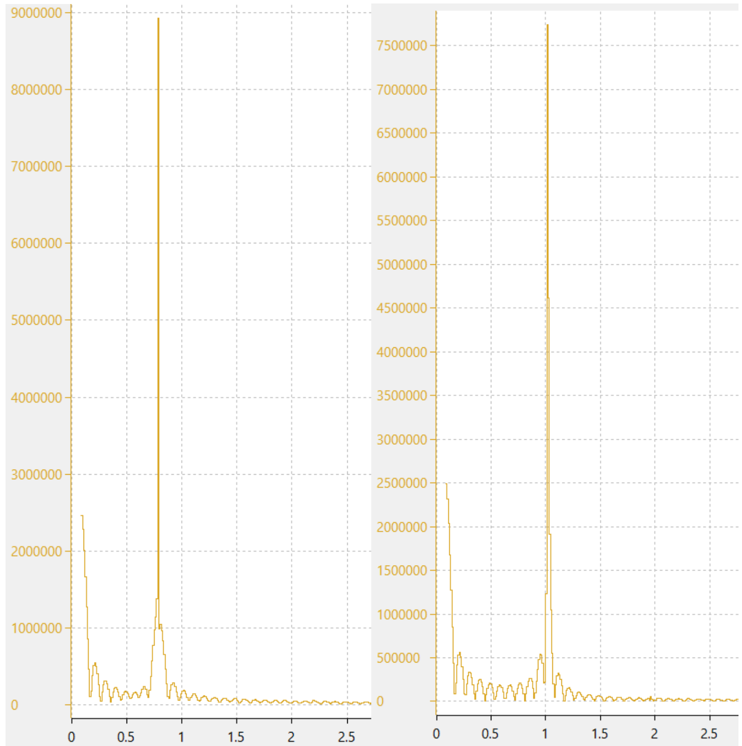

Figure 3.1: Leftmost peaks of the FFTs of a 3KHz and 4KHz sine waves.

Comparing the frequency plots of part 2 and part 3, the primary difference is the amount of side bands around the carrier frequency. The FFT plots of part 3 contain significantly more side bands than the FFT plots of part 2. The cause of this ripple effect is the zero padding where the multiplication of a rectangular function results in the convolution of a sinc function in the frequency domain. The effect is amplified in the 4KHz frequency signal due to the energy leakage around the carrier frequency, which is a result of the partial sample cycles of the input sequence. For example, the 4KHz signal contains 64 cycles while the 3KHz signal contains 85.33 cycles.

CODE:

```
    // Record sample of 3/4KHz sine wave
    if(HAL_ADC_Start_DMA(&hadc1, (uint32_t *)&ADC_buff_1024, ADC_BUF_SIZE_1024) != HAL_OK) {
      Error_Handler();
    }
    while (1)
    {
      if(record_done == 1){
        record_done = 0;
        break;
      }
    }

    // Zero out the last 124 complex points of the ADC_buff
    for (int i=0; i<124; i++)
    {
      ADC_buff_1024[1023-i] = 0;
    }
    // Complex FFT of 3/4KHz sample with zero padding
    convert_to_complex(ADC_buff_1024, CMPLX_buff_1024, 1024);
    // FFT of 3/4KHz sample with zero padding
    arm_cfft_f32(&fft_handler_1024, CMPLX_buff_1024, 0, 1);
    // Print magnitude of FFT
    set_magnitude(CMPLX_buff_1024, MGNTD_buff_1024, 1024);
    print_magnitude(MGNTD_buff_1024, 1024);
```

PART 4

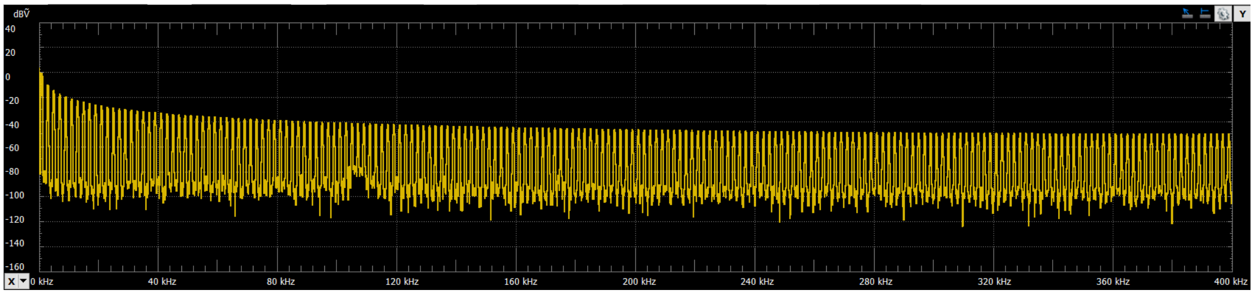

Figure 4.1: Screenshot of the FFT of a 1KHz sine wave through the AD2.

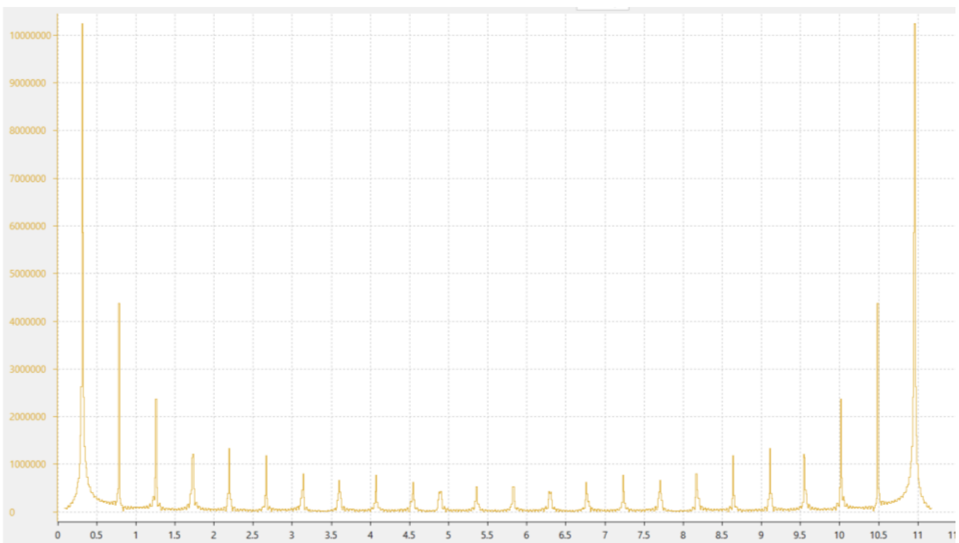

Figure 4.2: Screenshot of the FFT of a 1KHz sine wave through the STM32 Cube IDE

The STM32 Cube IDE FFT and the AD2 oscilloscope FFT display the frequency signal with some distinct similarities and differences. On one hand, both FFTs have the same general shape in which the highest frequency peak repeats over some integer multiple of the initial frequency with decreasing amplitude. On the other hand, the amplitude scaling, frequency axis, and the FFT window are different. While the AD2 oscilloscope displays the amplitude in log scale, the STM32 Cube IDE displays the energy of the frequency signal without any scaling. Additionally, while the AD2 oscilloscope displays the frequency signal until 400 KHz, the STM32 Cube IDE only displays the frequency signal until 48 KHz. Finally, while the AD2 oscilloscope displays only half of the FFT window, the STM32 Cube IDE displays the entire FFT window as seen by the mirrored plot across the midpoint.

CODE:

```
// Record sample of 1KHz square wave
if(HAL_ADC_Start_DMA(&hadc1, (uint32_t *)&ADC_buff_1024, ADC_BUF_SIZE_1024) != HAL_OK) {
  Error_Handler();
}
while (1)
{
  if(record_done == 1){
    record_done = 0;
    break;
  }
}

// Complex FFT of 1KHz square wave
convert_to_complex(ADC_buff_1024, CMPLX_buff_1024, 1024);
// FFT of 1KHz square wave
arm_cfft_f32(&fft_handler_1024, CMPLX_buff_1024, 0, 1);
// Print magnitude of FFT
set_magnitude(CMPLX_buff_1024, MGNTD_buff_1024, 1024);
print_magnitude(MGNTD_buff_1024, 1024);
```

Part 5

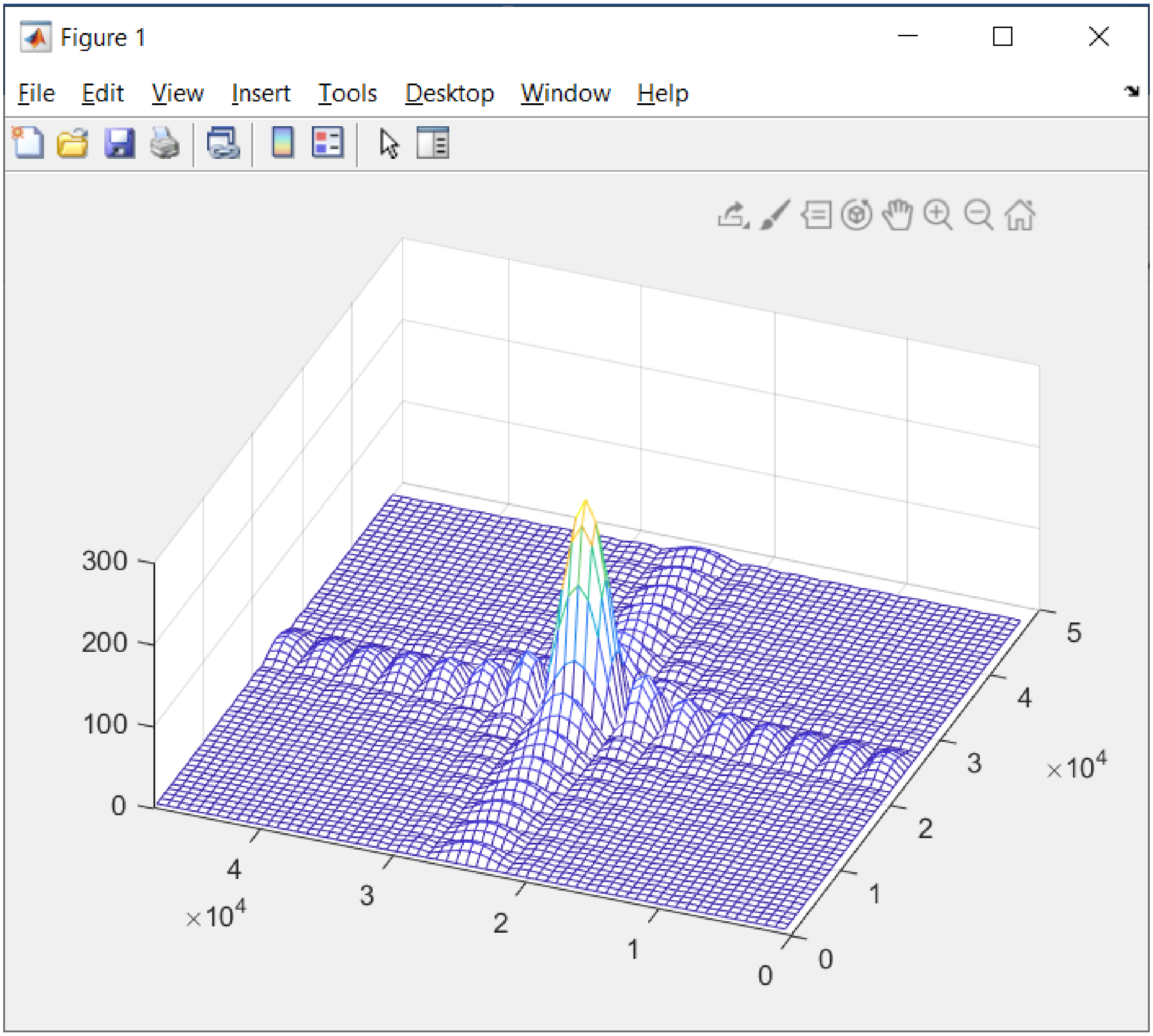

Figure 5.1: 2D FFT of a 64x64 input with a 16x16 center with values of 1.

CODE:

```
// Image
float img[64][64];
// fill the image center (16x16) with ones
for (int row=0; row<64; row++)
  for (int col=0; col<64; col++)
    img[row][col] = ((row > 23 && row < 40) && (col > 23 && col < 40)) ? 1 : 0;

// FFT Instance
arm_cfft_instance_f32 fft_handler_64;
arm_cfft_init_f32(&fft_handler_64, 64);

for (int row=0; row<64; row++)
{
  // Perform row FFT
  convert_to_complex(img[row], CMPLX_buff_64[row], 64);
  arm_cfft_f32(&fft_handler_64, CMPLX_buff_64[row], 0, 1);
}
float32_t NEW[64][128];
// Perform col FFT
for (int col=0; col<64; col++)
{
  float32_t arr[128];
  for (int i=0; i<64;i++)
  {
    arr[2*i]=CMPLX_buff_64[i][2*col];
    arr[2*i+1]=CMPLX_buff_64[i][2*col+1];
  }
  // Perform col FFT
  arm_cfft_f32(&fft_handler_64, arr, 0, 1);
  for (int i=0; i<64;i++)
  {
    NEW[col][2*i]=arr[2*i];
    NEW[col][2*i+1]=arr[2*  i+1];
  }
}
// Assign magnitude values
for (int row=0; row<64; row++)
{
  // Set magnitude of FFT
  set_magnitude(NEW[row], MGNTD_buff_64[row], 64);
}
```

DISCUSSION

The time signals and their respective FFTs followed the expected waveforms. Issues with the waveforms typically resulted from the sampling of the signal. For instance, the fillets around the carrier frequencies in any of the waveforms resulted from partial sample cycles. Apart from the cases covered in the prior portions of the report, the results did not deviate from expectations.