EE445M Lab 4 Report

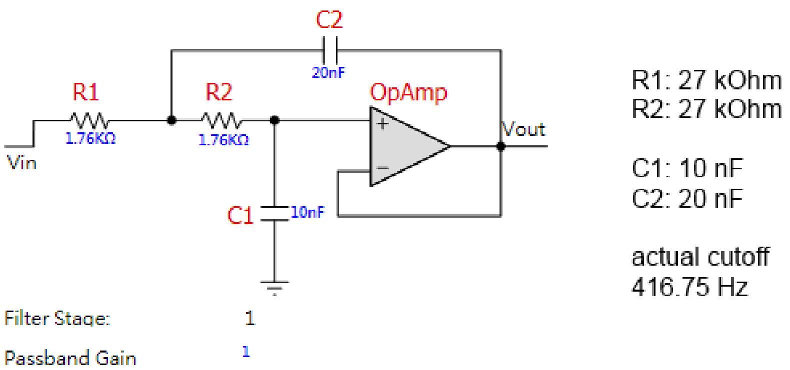
Yen-Kai Huang
Siavash Zanganeh Kamali

March 21, 2014

1 Objective

The goal of this lab is to learn the DSP theory of building a digital FIR filter. We also familiarize ourselves with theories behind and tools for building an analog filter. We will make use of the OS from Lab 3 and write a oscilloscope-spectrum analyser application with a LCD display.

2 Hardware Design



3 Software Design

Below is the code for sampling and acquiring the data.

```
// Consumer
// Foreground thread that takes in data from FIFO, apply filter, and
   record data
// If trigger Capture is set, it will perform a 64-point FFT on the
   recorded data
// and store result in fft_output[]
// Block when the FIFO is empty
unsigned long fft_output[FFT_LENGTH];

long recording[2*FFT_LENGTH];
unsigned char puti = 0;
unsigned char geti = 0;

#define SAMPLING_RATE 2000

```

```c
void Consumer(void) {
  ADC_Collect(0, SAMPLING_RATE, dummy, 64); // Timer triggered ADC sampling
  while (1) {
    // Get data, will block if FIFO is empty
    long data = OS_Fifo_Get();

    // Choosing whether to apply the filter
    if(Filter_Use == NOW_USING) data = Filter(data);
    else if(Filter_Use == STOP_USING) {
      Filter_Use = NOT_USING;
      Filter_ClearMemo();
    }

    // record data
    recording[puti] = recording[puti+FFT_LENGTH] = data;
    puti = (puti + 1) % FFT_LENGTH;

    if(puti == geti) { // a data point is just put in, so in this case it is overrun
      DataLost += 1;
      geti = (geti + 1) % FFT_LENGTH; // discard the oldest value
    } else {
      if (DisplayMode == TIME ) {
        OS_Signal(&Sema4DataAvailable);
      }
    }

    if(Capture == DO_CAPTURE) {
      short real; long imag;
      int k;
      Capture = NO_CAPTURE; // Acknowledge
      cr4_fft_64_stm32(fft_output, &recording[puti-FFT_LENGTH+1], FFT_LENGTH);
      for (k=0;k<FFT_LENGTH;k++) {
        real = (short) (fft_output[k] & 0x0000FFFF);
        imag= fft_output[k] >> 16;
        fft_output[k] = sqrt((long)(real)*(long)(real)+imag*imag);
      }
      if (DisplayMode == FREQUENCY)
        OS_Signal(&Sema4DataAvailable);
    }

    if(TriggerMode == CONTINUOUS) ++Capture;
  }
}


/****** Background Button Task ***/
// In Button Triggering mode, it trigger the Consumer to perform one fft
void ButtonTask(void) {
  if(TriggerMode == BUTTON) Capture = DO_CAPTURE;
}
```

Code/consumer.c

It uses the following filter code on input if specified to do so.

```
#define FILTER_LENGTH 51
const long ScaleFactor = 16384;
const long H[51]={-11,10,9,-5,1,0,-19,6,48,-12,-92,
      17,155,-20,-243,22,370,-24,-559,24,881,-24,-1584,24,4932,
      8578,4932,24,-1584,-24,881,24,-559,-24,370,22,-243,-20,155,
      17,-92,-12,48,6,-19,0,1,-5,9,10,-11};


long x[2*FILTER_LENGTH]; // this MACQ needs twice the size of FILTER_LENGTH
unsigned char n = FILTER_LENGTH-1; // to let pre-increment start at x[FILTER_LENGTH]

// Filter_ClearMemo
// this is called when one stops using the filter
// such that when filter is being used again the filter memory will be fresh
void Filter_ClearMemo(void) {
  unsigned char i;
  for(i = 0; i < 2*FILTER_LENGTH; ++i) x[i] = 0;
  n = FILTER_LENGTH-1; // to let pre-increment start at x[FILTER_LENGTH]
}

// Filter
// Digital FIR filter, assuming fs=1 Hz
// Coefficients generated with FIRdesign64.xls
//
// y[i]= (h[0]*x[i]+h[1]*x[i-1]++h[63]*x[i-63])/256;
long Filter(long data) {
  long y = 0;
  unsigned char i;

  if(++n == 2*FILTER_LENGTH) n = FILTER_LENGTH;
  x[n] = x[n-FILTER_LENGTH] = data;

  // Assuming there is no overflow
  for(i = 0; i < FILTER_LENGTH; ++i){
    y += H[i]*x[n-i];
  }
  y /= ScaleFactor;

  return y;
}
```

Code/fir.c

The result will be displayed on the LCD with the following thread.

```
char DisplayIsOn = ON;
void Display (void) {
  static char LastMode = FREQUENCY;
  while (1) {
    OS_Wait(&Sema4DataAvailable);
    if (DisplayMode == TIME) {     // DisplayMode is in time Domain
      if (LastMode != DisplayMode) {
```

```c
        ST7735_FillScreen(0); // set screen to black
        ST7735_DrawFastVLine(19,10,100,ST7735_Color565(255,255,41));
        ST7735_DrawFastHLine(19,110,140,ST7735_Color565(255,255,41));
        ST7735_PlotClear(0,4095);
      }
      if (DisplayIsOn){
        ST7735_PlotPoint(recording[geti]);
        ST7735_PlotNext();
      }
      geti = (geti + 1) % FFT_LENGTH;
      LastMode = TIME;
    } else if(DisplayMode == FREQUENCY) { // DisplayMode is in frequency Domain
      int i;
      /* Display the frequency */
      if (LastMode != FREQUENCY) {
        ST7735_FillScreen(0); // set screen to black
        ST7735_DrawFastVLine(19,10,100,ST7735_Color565(255,255,41));
        ST7735_DrawFastHLine(19,110,140,ST7735_Color565(255,255,41));
        ST7735_PlotClear(0,4095);
      }
       ST7735_FillRect(21, 10, 140, 99, ST7735_BLACK);
      if (DisplayIsOn){
        ST7735_PlotReset();
        for (i=0;i<FFT_LENGTH;i++) {
          ST7735_PlotBar(fft_output[i]);
          ST7735_PlotNext();
        }
      }
      LastMode = FREQUENCY;
    }
  }
}
```

Code/display.c

The details setting of display mode, triggering mode of FFT (Once, Button-triggered, Continuous) and the use of the FIR filter can all be adjusted in real time with the following Interpreter commands.

```c
/************************ DSP commands *********************/
//Descriptiont: command for turning the filter on or off
// Expects parameter "on" or "off"
#define NOW_USING   1
#define STOP_USING  0
#define NOT_USING  -1
extern char Filter_Use;

static void adjustFilterCommand(void) {
  char *buffer;

  const char *msg = "Error: Enter 'on' or 'off'\r";

  if (buffer = strtok(NULL, " ")) {
    if(strcmp(buffer, "on") == 0) {
      Filter_Use =  NOW_USING;
    } else if(strcmp(buffer, "off") == 0) {
      Filter_Use = STOP_USING;
```

```
    } else {
      puts(msg);
    }
  } else {
    puts(msg);
  }
}


// Description: command for changing the fft sampling mode
// Available modes are Once, Button-trigger, and Continuous-trigger
extern char TriggerMode;
extern short Capture;
static void fftCommand(void) {
  char *buffer;

  const char *msg = "Error: Enter 'c' (continuous), 'b' (button), or 'o' (once)\r";

  if (buffer = strtok(NULL, " ")) {
    if(strcmp(buffer, "c") == 0) {
      TriggerMode = CONTINUOUS;
    } else if(strcmp(buffer, "b") == 0) {
      TriggerMode = BUTTON;
    } else if(strcmp(buffer, "o") == 0) {
      TriggerMode = NOW;
      Capture = 200;
    } else {
      puts(msg);
    }
  } else {
    puts(msg);
  }
}

// Command for performing a FFT once
extern long recording[];
extern long fft_output[];
static void captureCommand(void){
  char *buffer;

  const char *msg = "Error: Enter 'fft' or 'voltage' \r";


  if (buffer = strtok(NULL, " ")) {
    if(strcmp(buffer, "fft") == 0) {
      int i;
      for (i=0;i<64;i++){
        printf("%ld\r\n", fft_output[i]);
      }
    } else if(strcmp(buffer, "voltage") == 0) {
      char i, start;
      i = start = puti;
      do {
        printf("%ld\r\n", recording[i]);
        i = (i+1)%64;
```

```c
      } while (i != start);
    } else {
      puts(msg);
    }
  } else {
    puts(msg);
  }
}

//Description: command for changing display mode between Time mode and Frequency mode
//Will change the global variable displayMode
extern unsigned char puti;
extern unsigned char geti;

#define TIME      1
#define FREQUENCY 2
extern char DisplayMode;
#define ON 1
#define OFF 0
extern char DisplayIsOn;
static void parseDisplayCommand(void) {
  char *buffer;

  const char *msg = "Error: Enter 'time' or 'freq' or 'run' or 'stop'\r";

  if (buffer = strtok(NULL, " ")) {
    if(strcmp(buffer, "time") == 0) {
      if (DisplayMode == FREQUENCY){
        long sr;
        sr = StartCritical();
        puti = geti = 0;
        Sema4DataAvailable.Value = 0;
        DisplayMode =  TIME;
        EndCritical(sr);
      }
    } else if(strcmp(buffer, "freq") == 0) {
      DisplayMode = FREQUENCY;
    } else if(strcmp(buffer, "run") == 0) {
      DisplayIsOn = ON;
    } else if(strcmp(buffer, "stop") == 0) {
      DisplayIsOn = OFF;
    } else {
      puts(msg);
    }
  } else {
    puts(msg);
  }
}
```

Code/interpreter_cmds.c

4 Measurement

(a) Dynamic circuit performance (procedure 2)

Analog Filter Performance Measurement			
freq	input voltage	output voltage	gain
100	1.58	1.58	1
200	1.58	1.52	0.962025316
300	1.58	1.36	0.860759494
400	1.58	1.14	0.721518987
500	1.58	0.9	0.569620253
700	1.58	0.6	0.379746835
1000	1.58	0.38	0.240506329
1500	1.58	0.22	0.139240506
2000	1.58	0.18	0.113924051

Table 1. Analog Filter Performance Measurement

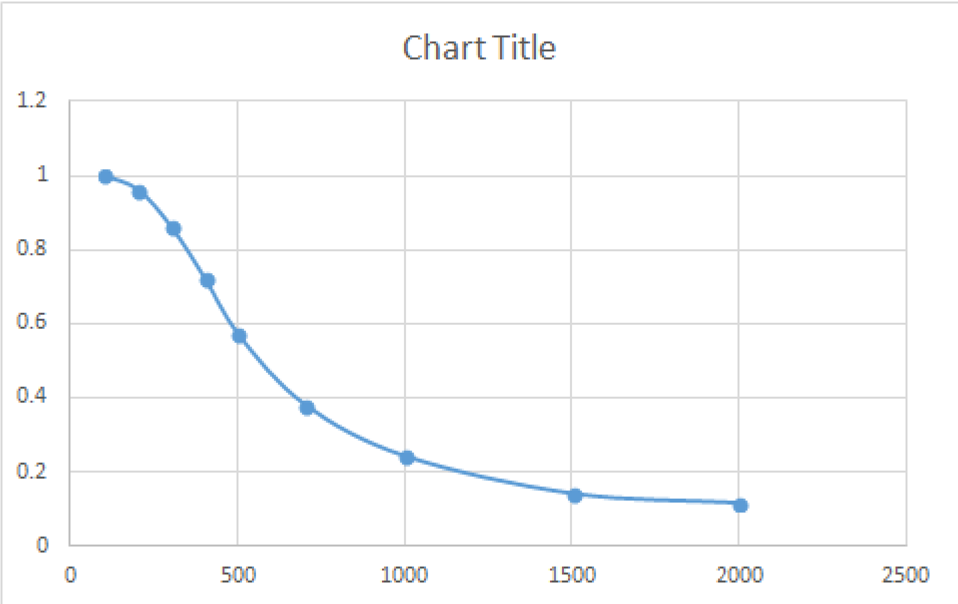


Figure 1: Plot of analog filter performance measurement

(b) Digital scope data for each of the three frequencies (three by three = 9 graphs) (procedure 3)

The result is shown Figure 2. The pictures of LCD screens have been edited for better clarity.

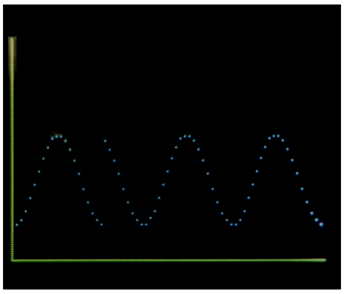
(c) Spectrum analyzer data (three by three = 9 graphs) (procedure 3)

See Figure 3

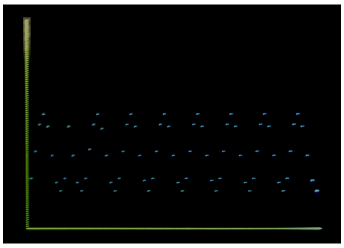
(d) FIR filter test data (procedure 4)

We imitated the turning of a robot by moving the sensor close to a wall surface and then move away. The data collected is shown.

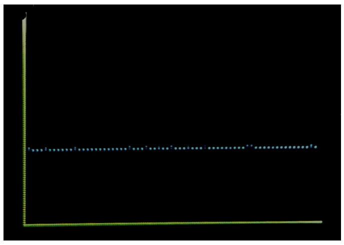
See Figure 4



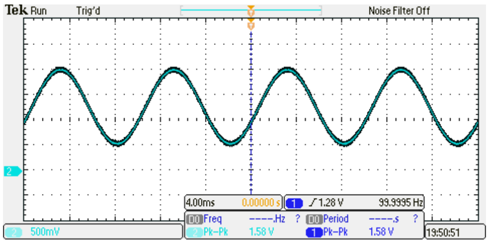
(a) Digital scope data frequency = 100 Hz



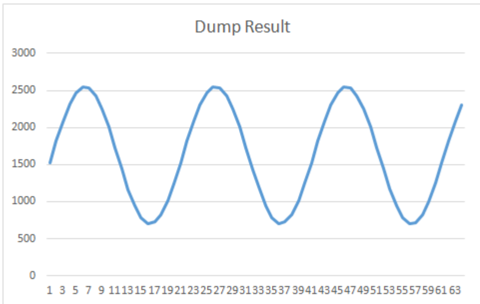
(b) Digital scope data frequency = 250 Hz



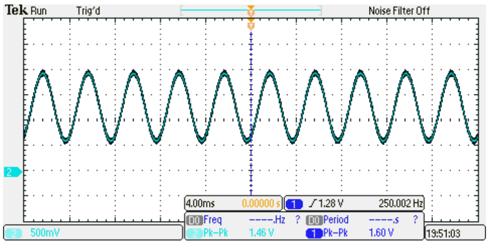
(c) Digital scope data frequency = 2 kHz



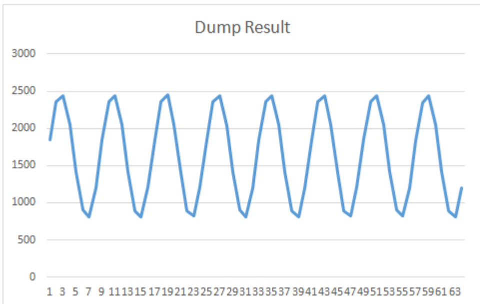
(d) Spectrum analyzer data



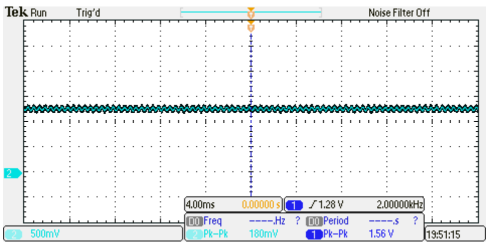
(e) print data



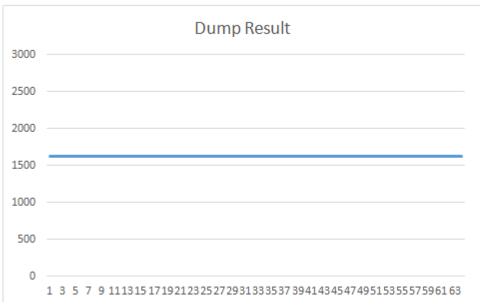
(f) Spectrum analyzer data



(g) print data

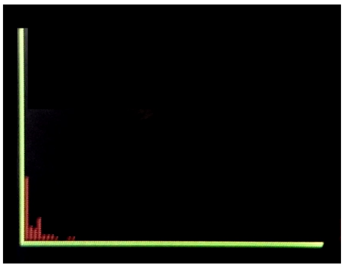


(h) Spectrum analyzer data

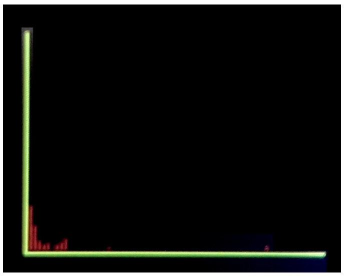


(i) print data

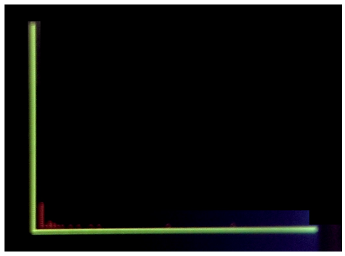
Figure 2: Plots and scope pictures



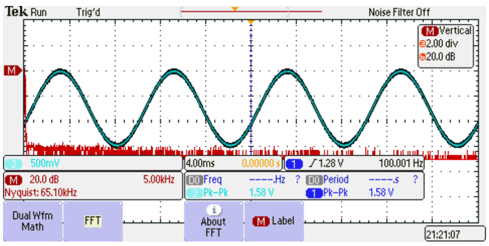
(a) Digital scope data frequency = 100 Hz



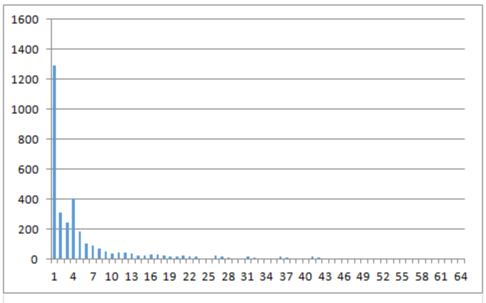
(b) Digital scope data frequency = 250 Hz



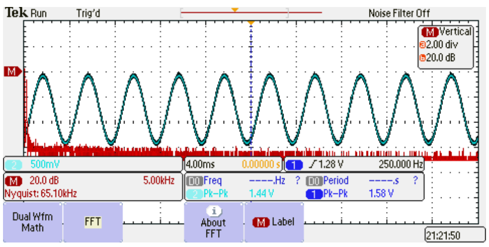
(c) Digital scope data frequency = 2 kHz



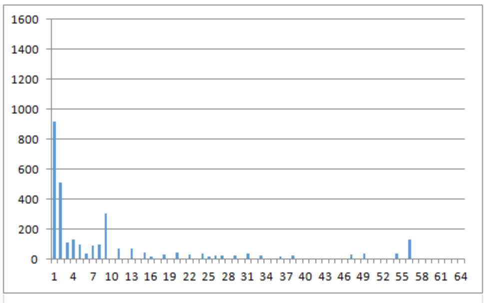
(d) Spectrum analyzer data



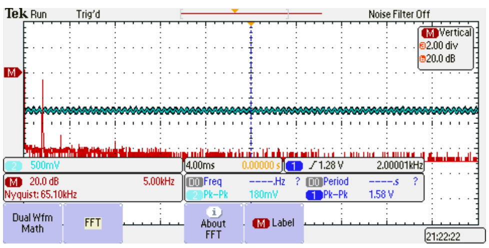
(e) print data



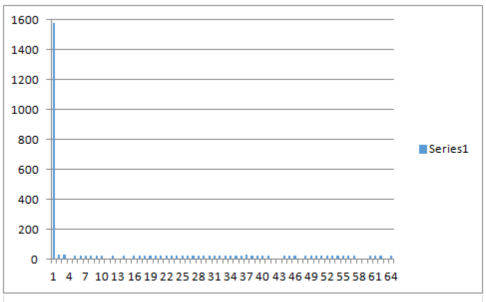
(f) Spectrum analyzer data



(g) print data

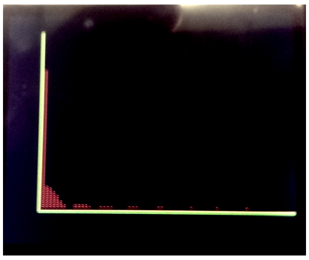


(h) Spectrum analyzer data

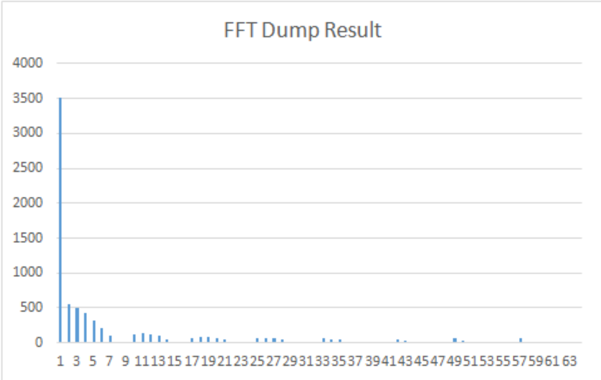


(i) print data

Figure 3: Plots and scope pictures



(a) FFT plot result



(b) FFT dump result

Figure 4: FIR filter test data

5 Analysis and Discussion

Give the calculations (equations and results) you used to estimate the cutoff frequencies for your HPF and LPF (Preparation 1)? How did the measured frequency response compare to the estimations (procedure 2)?

We wanted the LPF's cutoff to be $400\,Hz$ since we set the sampling rate to $2\,kHz$. Although cutoff is much lower than the maximum allowed frequency according to Nyquist's theorem, we chose such a low cutoff to be improve performance and prevent aliasing.

Equations: For a Butterworth 2 Pole Sallen-key LPF, $C_1 = 141.4\,\mu F$ and $C_2 = 70.7\,\mu F$ and $R = 10\,k\Omega$, $f_c = 1\,Hz$. To change fc to $400\,Hz$, the new capacitor values can be calculated by $C_1' = \frac{C_1}{2\pi f_c} = 0.0563\,\mu F$ and $C_2' = C_1'/2 = 0.0281\,\mu F$. If we multiply the capacitors by 0.344484 to set $C_1 = 20\,nF$ and $C_2 = 10\,nF$, to keep f_c the same as $400\,Hz$, we should also divide the resistance by 0.344484 that results in $R = 28.131k\Omega$. Finally, we chose the resistance as $27\,k\Omega$ which resulted in a f_c of $416.75\,Hz$.

After doing procedure do we find the cutoff frequency to be around $410\,Hz$. (frequency at which gain was 0.707) This difference is probably due to inaccuracy of resistance and capacitance of the filter components.

Explain how you measured maximum bandwidth (procedure 5). What was the limiting factor affecting bandwidth?

We added a variable that starts by 0 and is incremented everytime ADC fails to put the sampled data into the FIFO. We also added a separated variable that starts by 0 and is incremented everytime Producer fails has to disregard an old data that is still not displayed by the Display thread. Both variables should be 0 through the test. We started by a low Sampling rate by $2000\,Hz$ and continuously increased it, and read both DataLost variables with or without FIR filter being on. As soon as there was a DataLost in either variable, the bandwidth was determined as the previous step with no DataLost.

Regardless of FIR filter being on or off, liming factor was the display thread as only that variable was incremented as soon as we chose a sampling rate more than the bandwidth. We were able to increase bandwidth by optimizing display functions. (But the system was still limited by the display thread)

We also tested the system with display off and the bandwidth was much higher than before.

What is the expected FFT output if the input is a squarewave?

Squarewave is the sum of sinewaves with frequencies of $(2n+1)f$ for all $n = 0, 1, 2, 3, \cdots$ where f is the primary frequency of squarewave.

Therefore, the FFT result in the frequency domain should be a impulse train of harmonious frequencies with decreasing strength.

Look at the noise in your digital samples when it is very quiet? What type of noise is it?

The noise when we don't move the distance sensor is sudden spikes at random times. (we used distance sensor, not microphone)

We made a big fuss over jitter in labs 2 and 3. Can you estimate the jitter in your ADC samples?

Jitter is 0, since we are using a timer to trigger ADC capture.

Prove your FIR implementation can not overflow.

largest coefficient of the FIR filter $= 8578 < 2^14$
The number of points $= 51 < 2^6$
Maximum number of ADC result $= 4095 < 2^12$

Maximum total size $< 2^6 \cdot 2^14 \cdot 2^12 = 2^32$
Therefore, the sum fits in a 32-bit unsigned long variable

Look at the symmetry in the $h[51]$ coefficients in the example FIR design. How could you rewrite the following filter equation to reduce the number of multiplies from 51 to 26?

We could rewrite the filter equation to simplify the calculation as follows

```
y[i]= (h[0]*x[i]+h[1]*x[i-1]+h[2]*x[i-2]+...+h[50]*x[i-50])/256;
    = (h[0]*(x[i]+x[i-50]) + h[1]*(x[i-1]+x[i-49] + ... h[25]*(x[i-24]+
   x[i-26]) + h[26]*x[i-26] ) / 256;
```

If your system executed the FIR filter using the multiply and accumulate instruction (MLA), you can skip this question. Explain how the MLA instruction could have made your filter execute faster? If you were to have used the MLA instruction, would it have been more accurate?

The compiler used MLA in the filter implementation,.

MLA is faster than using consecutive MUL and ADD instruction. So, it can potentially speed up the FIlter execution time by a factor of 2. However, since we have a for loop in the system, the effect of the otther instructions cannot be neglected, so execution time will practically increase, but not increase by a factor of 2.