ESE 150 – Lab 08: Machine Level Language

LAB 08

In this lab we will gain an understanding of the instruction-level implementation of computation on a microprocessor by:

1. Using Arduino to perform the Fourier Transform on sampled data in the time domain, converting it to the frequency domain
2. Timing your Fourier Transform to see how long the operation takes to perform
3. Calculating number of cycles each instruction takes and calculating the time of execution using the assembly file.

Background:

Let us learn first what a .elf file and .hex file is.
ELF is an acronym for Executable Linking Format. Files that contain the .elf file format are system files that store executable programs, shared libraries and memory dumps.
A HEX file is a hexadecimal source file typically used by programmable logic devices.
In this lab you will be generating assembly code to observe how your code works at the processor instruction level. You will see which instructions get executed in what order and how many times. This will give you an idea of which instructions take longer time and how to reduce your execution time.

In previous labs we've performed DFT for various applications without thinking about the implementation. We have had programs which runs for several seconds and sometimes minutes!
Now we will look deeper at how the machine executes your program and how much time that takes.
From your lecture, you know what processor instruction are and how to read them. Let's work with them to understand the execution time of your program.

Prelab: Obtaining the Assembly code

- In this section, we'll compile our Arduino code and obtain the location of a temporary .hex file
- We'll convert the .hex file to assembly code.

Optimizing our code requires that we view the generated machine code (hex) in assembly language. This allows us to know how things work at the instruction level. When we review assembly code we understand how the computers hardware works and functions on a low-level. This allows us to calculate how many clock cycles each instruction takes and how to optimize our code/logic to achieve faster execution. Make sure to read the entire lab as a part of the pre-lab.

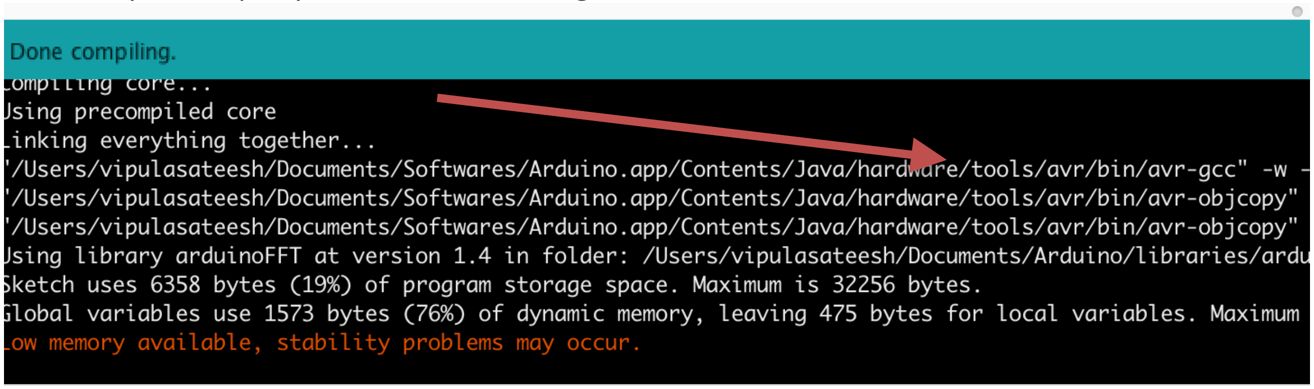
1. First let us obtain the location of our temporary folders created during the compilation of the Arduino code.

2. We will be compiling the code to compute the Fourier Transform of a discrete sample data and detect the peak frequency present in it. Go over the code and understand the math behind it.

3. Paste the following code in the Arduino IDE and name your file FT_PeakDetection.

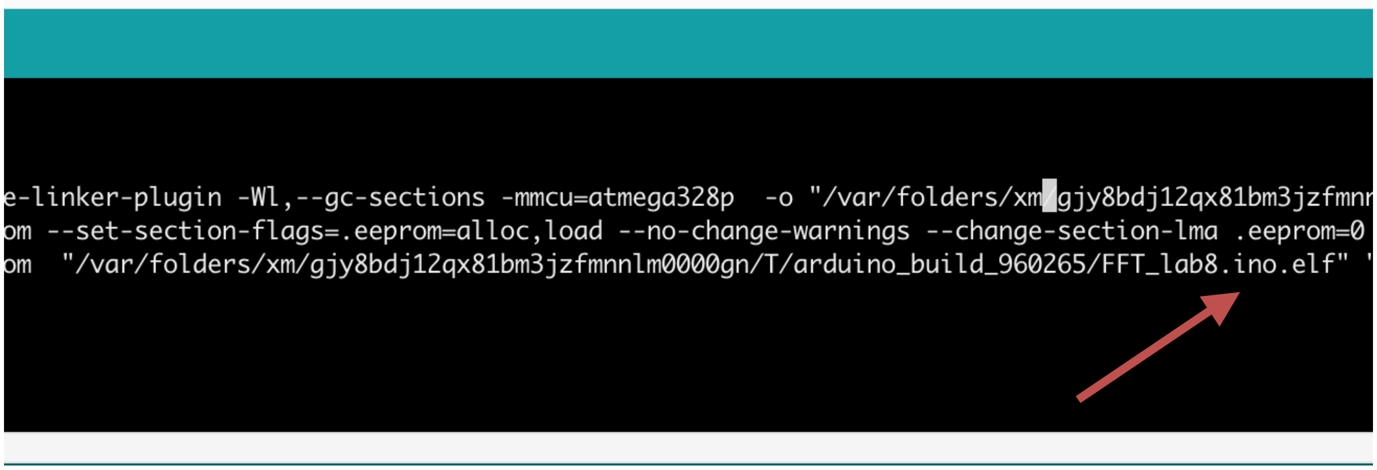
```
#define MAX_SAMPLES 128    //make it a power of 2
#define SAMPLING_TIME 0.0002 //Hz, must be less than 10000 due to ADC
#define pi 3.1416
#define scale_factor 100
int samples[MAX_SAMPLES] ;
boolean samplesReceived = false ;
int sineref[MAX_SAMPLES];
int cosref[MAX_SAMPLES];
long ask;
long ack;
double freq[MAX_SAMPLES];
double samp_freq;
int k = 0 ;
int i ;
double fft_abs;

void setup() {
    Serial.begin(9600);  // setup serial monitor speed
    samp_freq = 1/(SAMPLING_TIME + 0.000125);     //Analog read  delay
}

long dotproduct(int length, int *a, int *b) {
    long sum=0L;      // Initializing long
    for(int i=0;i<length;i++)
    sum+=(long)a[i]*(long)b[i]; //type casting
    return(sum);
}

void loop() {
  for(int i = 0; i < MAX_SAMPLES; i++) {
    samples[i] = analogRead(A0); // read input from A0
    delayMicroseconds(200);
    samplesReceived = true ;
  }


  if (samplesReceived){
    for(int i = 0; i < MAX_SAMPLES; i++) {
    Serial.println(samples[i]);
  }
   for(k =0;k<MAX_SAMPLES;k++){
      for (i=0;i<MAX_SAMPLES;i++){
        sineref[i]=(int)(sin(2*pi*k*i/MAX_SAMPLES)*(1<<10));  //sine and
cosine samples will be between -1 and 1, so multiply those by 2^10 and
round
        cosref[i]=(int)(cos(2*pi*k*i/MAX_SAMPLES)*(1<<10));
      }

ask=(dotproduct(MAX_SAMPLES,samples,sineref))/(MAX_SAMPLES*scale_factor);
// scaling it

ack=dotproduct(MAX_SAMPLES,samples,cosref)/(MAX_SAMPLES*scale_factor);
      freq[k] = k*samp_freq/(MAX_SAMPLES-1);
     /* Serial.print("For :");
       Serial.println(freq[k]);
       Serial.println(fft_abs);    printing the amplitude for each freq */
       fft_abs = abs(ask)+abs(ack);   // abs returns the modulous

      if (fft_abs>1000){
        Serial.println("Peak  found at ");
          Serial.println(freq[k]);
        }
    }
    while(1);       //Run code once
  }
}
```

4. Now, press SHIFT+ compile to compile the code to see the output files (ELF, BIN, HEX files).
5. If you're using Windows PC, jump to step 10. If you're using MAC OS then continue with step 6.
6. Once you compile you will see something like this:



Done compiling.

```
compiling core...
Using precompiled core
Linking everything together...
"/Users/vipulasateesh/Documents/Softwares/Arduino.app/Contents/Java/hardware/tools/avr/bin/avr-gcc" -w -
"/Users/vipulasateesh/Documents/Softwares/Arduino.app/Contents/Java/hardware/tools/avr/bin/avr-objcopy"
"/Users/vipulasateesh/Documents/Softwares/Arduino.app/Contents/Java/hardware/tools/avr/bin/avr-objcopy"
Using library arduinoFFT at version 1.4 in folder: /Users/vipulasateesh/Documents/Arduino/libraries/ardu
Sketch uses 6358 bytes (19%) of program storage space. Maximum is 32256 bytes.
Global variables use 1573 bytes (76%) of dynamic memory, leaving 475 bytes for local variables. Maximum
Low memory available, stability problems may occur.
```

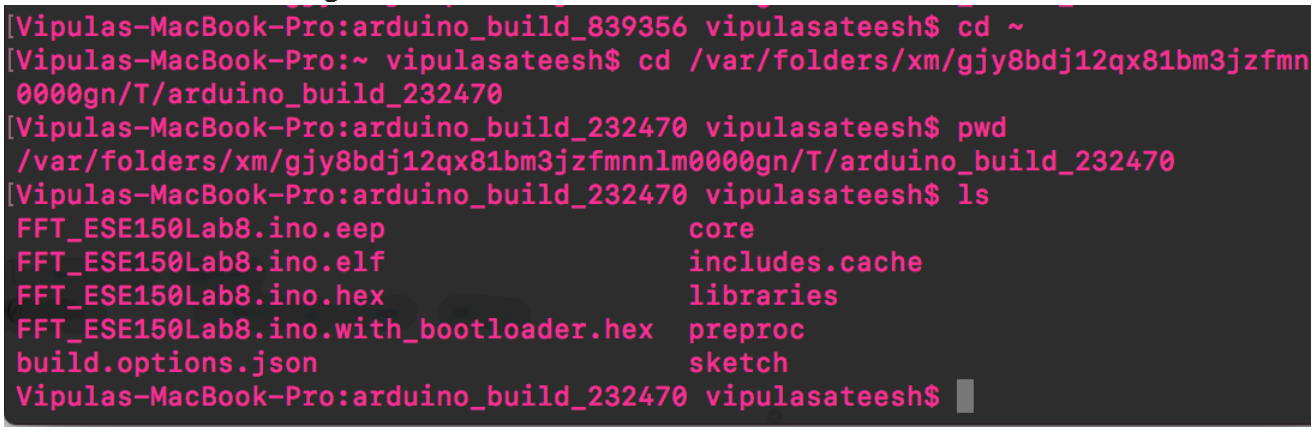


```
e-linker-plugin -Wl,--gc-sections -mmcu=atmega328p  -o "/var/folders/xm/gjy8bdj12qx81bm3jzfmnr
om --set-section-flags=.eeprom=alloc,load --no-change-warnings --change-section-lma .eeprom=0
om  "/var/folders/xm/gjy8bdj12qx81bm3jzfmnnlm0000gn/T/arduino_build_960265/FFT_lab8.ino.elf" '
```

The directory /Users/vipulasateesh/Documents/Softwares/Arduino.app/Contents/Java/hardware/tools/avr/bin contains a file called avr-objdump which will be used in step 5. Scroll left to see the .elf file's location.

7. Now let's generate assembly code from the .elf file generated. Open your terminal and type the following.
   a. `cd /var/folders/xm/gjy8bdj12qx81bm3jzfmnnlm0000gn/T/arduino_build_232470`
      replace the specify path with the one you found above
   b. `pwd`
   c. `ls`

   You should see something like:



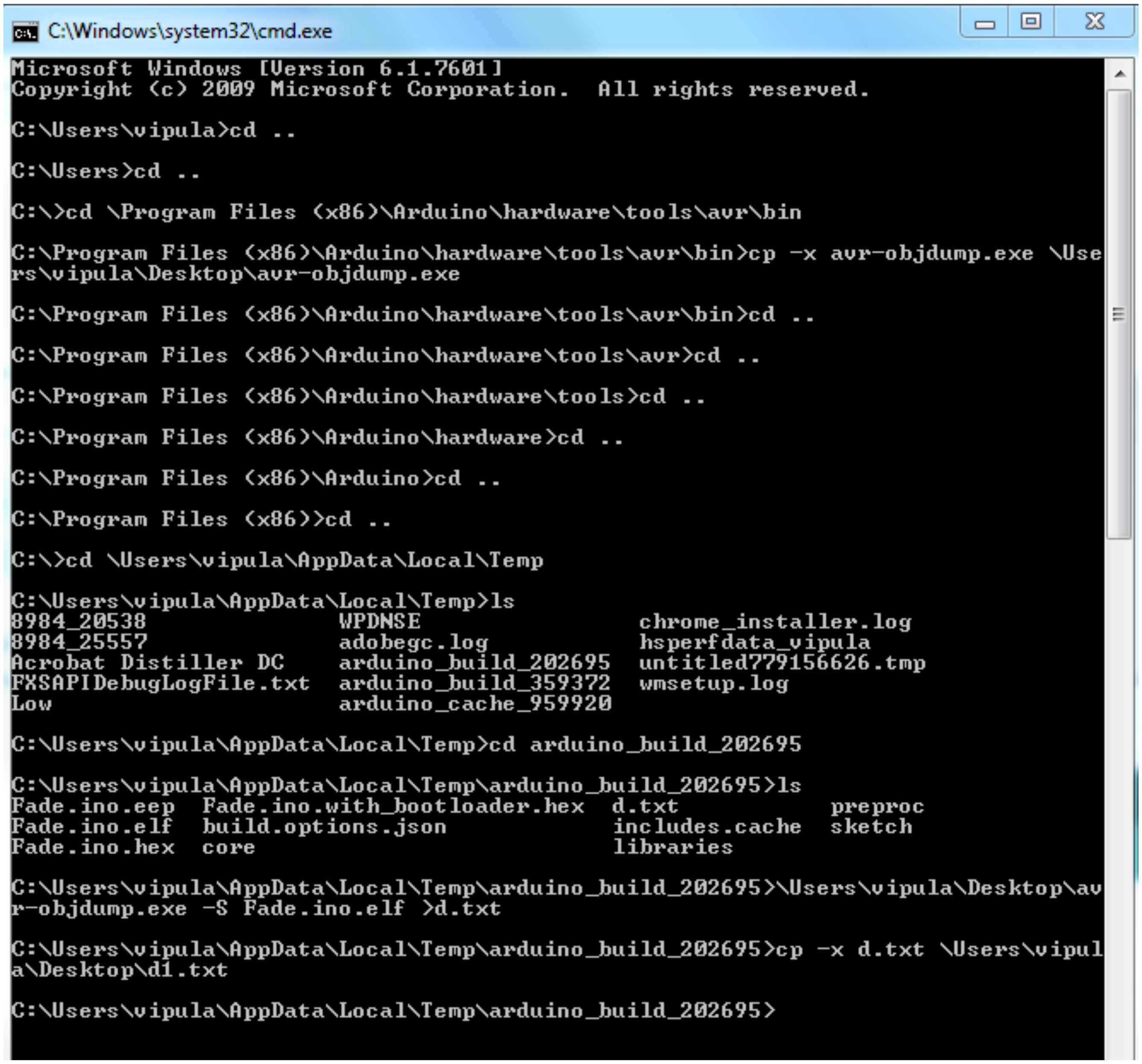
```
[Vipulas-MacBook-Pro:arduino_build_839356 vipulasateesh$ cd ~
[Vipulas-MacBook-Pro:~ vipulasateesh$ cd /var/folders/xm/gjy8bdj12qx81bm3jzfmn
0000gn/T/arduino_build_232470
[Vipulas-MacBook-Pro:arduino_build_232470 vipulasateesh$ pwd
/var/folders/xm/gjy8bdj12qx81bm3jzfmnnlm0000gn/T/arduino_build_232470
[Vipulas-MacBook-Pro:arduino_build_232470 vipulasateesh$ ls
FFT_ESE150Lab8.ino.eep                  core
FFT_ESE150Lab8.ino.elf                  includes.cache
FFT_ESE150Lab8.ino.hex                  libraries
FFT_ESE150Lab8.ino.with_bootloader.hex  preproc
build.options.json                      sketch
Vipulas-MacBook-Pro:arduino_build_232470 vipulasateesh$
```

   The command cd is used to open the directory which is mentioned after the command. The command ls is used to obtain all the files and folders in the current directory.
   Note the FFT_ESE150Lab8.ino.elf file, which is used in our next step.

8. Now, generate the assembly file. Here again you should replace the location with the location you obtained from the Arduino console. Also observe here, d.txt is the txt file you're storing the assembly code in.
   /Users/vipulasateesh/Documents/Softwares/Arduino.app/Contents/Java/hardware/tools/avr/bin/avr-objdump -S FFT_ ESE150Lab8.ino.elf >d.txt

Note: There are other ways to generate the assembly code and some applications do it for us by taking in the elf file. For more information on this you can refer to the link below.
https://sourceforge.net/projects/arduino-to-assembly-converter/

9. Copy paste the txt file onto another txt file which is more accessible. To do that type in the following command and replace the destination location with the directory of your choice.
   ```
   cp -X d.txt /Users/vipulasateesh/Documents/Penn/d1.txt
   ```
   You can now skip to step 12.
10. If you're not using a MAC OS, you can obtain the output files using the command terminal.
    (It is highly recommended you do your prelab in either Detkin or Klab; these instructions below works smoothly in detkin or klab.)
11. Once you compile your code by using SHIFT+ compile, check if you have the path of your temporary output files in the console. If not, open your command terminal. Please keep in mind to replace the user name here 'vipula' to your name and change any other file names if required.
    - Check your current directory, it should be C:\ (Root Directory). If it is not, type
      ```
      > cd ..
      ```
    - This command takes you back to your parent directory. Do this until you are in C:\ directory.
    - Next type the following to obtain the file avr-objdump.exe to your desktop
      ```
      > cd \Program Files (x86)\Arduino\hardware\tools\avr\bin
      > cp -x avr-objdump.exe \Users\vipula\Desktop\avr-objdump.exe
      ```
    - Now type `cd ..` to go back to your Root directory
    - Next we will navigate to your output file location. Type the following
      ```
      > cd \Users\vipula\AppData\Local\Temp
      ```
    - Now enter `ls` to see all your temporary files. One of the will be named arduino_build_202695 (With some change in the name). Cd into this folder by typing this. You can
      ```
      > cd arduino_build_202695
      ```
    - Now type ls to see all the contents of the folder. You should be able to see your ELF, HEX and BIN files now. Type the following
      ```
      > \Users\vipula\Desktop\avr-objdump.exe -S FFT_ ESE150Lab8.ino.elf > d.txt
      > cp -x d.txt \Users\vipula\Desktop\d1.txt
      ```
    - With this you have your assembly code file in your desktop!
    - You can refer to this image below for better clarity.


C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\vipula>cd ..

C:\Users>cd ..

C:\>cd \Program Files (x86)\Arduino\hardware\tools\avr\bin

C:\Program Files (x86)\Arduino\hardware\tools\avr\bin>cp -x avr-objdump.exe \Use
rs\vipula\Desktop\avr-objdump.exe

C:\Program Files (x86)\Arduino\hardware\tools\avr\bin>cd ..

C:\Program Files (x86)\Arduino\hardware\tools\avr>cd ..

C:\Program Files (x86)\Arduino\hardware\tools>cd ..

C:\Program Files (x86)\Arduino\hardware>cd ..

C:\Program Files (x86)\Arduino>cd ..

C:\Program Files (x86)>cd ..

C:\>cd \Users\vipula\AppData\Local\Temp

C:\Users\vipula\AppData\Local\Temp>ls
8984_20538              WPDNSE                 chrome_installer.log
8984_25557              adobegc.log            hsperfdata_vipula
Acrobat Distiller DC    arduino_build_202695   untitled779156626.tmp
FXSAPIDebugLogFile.txt  arduino_build_359372   wmsetup.log
Low                     arduino_cache_959920

C:\Users\vipula\AppData\Local\Temp>cd arduino_build_202695

C:\Users\vipula\AppData\Local\Temp\arduino_build_202695>ls
Fade.ino.eep  Fade.ino.with_bootloader.hex  d.txt           preproc
Fade.ino.elf  build.options.json            includes.cache  sketch
Fade.ino.hex  core                          libraries

C:\Users\vipula\AppData\Local\Temp\arduino_build_202695>\Users\vipula\Desktop\av
r-objdump.exe -S Fade.ino.elf >d.txt

C:\Users\vipula\AppData\Local\Temp\arduino_build_202695>cp -x d.txt \Users\vipul
a\Desktop\d1.txt

C:\Users\vipula\AppData\Local\Temp\arduino_build_202695>
```

12. Now you have created d1.txt with your assembly code in it!
    Familiarize yourself with the assembly code, the more you look through it, the easier the lab will be.
13. Go over the code and identify the sections associated with a line of Arduino C code.
    a) Search for (e.g. in the text editor Press CTRL+F and enter)
       `sineref[i]=(int)(sin(2*pi*k*i/MAX_SAMPLES)*(1<<10));`
       to see how this command is executed.
    b) You will obtain something like this:

```
     70e: 41 01          movw r8, r2
     710: 03 2c          mov  r0, r3
     712: 00 0c          add  r0, r0
     714: aa 08          sbc  r10, r10
     716: bb 08          sbc  r11, r11
     718: c5 01          movw r24, r10
     71a: b4 01          movw r22, r8
     71c: 0e 94 f2 04    call 0x9e4      ; 0x9e4 <__floatsisf>
```

    c) Recall we looked at instructions like add in lecture. For example the `call 0x9e4` instruction on the address `71c` means the control is now being passed on to the instruction by the address `0xc9e <__floatsisf>`. To see descriptions of these instructions, open the datasheet of the ATmega328 by clicking on the link below, and go to Section 36 Instruction Set Summary.
       http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

    d) Look up the dotproduct

       `int dotproduct(int length, int *a, int *b)`

       This is actually a bit tricky. The compiler copied the code for dotproduct into the loop() routine at the places where it was called, so you will actually see that signature line 4 times. To confuse matters further, it shows it twice for each time it is copied into the code (once for computing ask, once for computing ack). Starting from the first occurrence, find where it performs the multiply (hint: `__mulhisi3`), where it does the accumulating addition, and where it performs a branch back to the top of the loop. Notice that a divide follows the branch at the end of the loop. This divide (`__divmodsi4`) corresponds to the division you see in loop() following the dotproduct() call in the C code. Following that you can see where it stores the result into the `ask` variable. From these landmarks, identify and extract the code for the dotproduct.

       Look further down and identify the dotproduct code for the computation of the `ack` variable. Note that it is the same sequence of instructions.

14. Extract the code for `__mulhisi3`. Include all the routines that it calls or jumps to.
15. You will need these in Section 3 (and your report). Show your extracted code to your TA for prelab checkoff.

For more information on this refer to Section 3- 2a

Lab Procedure:

Lab – Section 1: Fourier Transform using an Arduino

- In this section we'll use Arduino to compute the Fourier Transform of a signal and recheck the most prominent frequency.

1. We will set up the function generator we used in previous labs
    a. Set the output to high-Z.
    b. Select a 300Hz sine wave with 2Vpp and 1V offset.
    c. Turn on the output, attach a BNC to BNC cable, and plug it into Channel 1 on the oscope.
2. Use your Arduino to sample the 300 Hz sinusoid (see lab 1 for help!)
3. Run the code from the prelab to compute the Fourier Transform and print the most prominent frequencies.
4. Identify which of the frequencies printed is the frequency of your signal and what the others could be. (Note: we haven't performed any calibration calculations to identify which frequency corresponds to each k.)

Lab – Section2: Timing the Dot Product

- In this section, you'll time the dotproduct using the in-built micros() command.
- Finally, you'll calculate different size dotproducts and time taken for each.

1. Take a look at the micros() function in the Arduino library by clicking on the link below.
   https://www.arduino.cc/reference/en/language/functions/time/micros/
2. Initialize two variables that will be used to calculate the time in micro seconds at two different points in your program to calculate how much time the Arduino takes to execute that section.

```
unsigned long microseconds1;
unsigned long microseconds2;
unsigned long time_elapsed;
```

3. Call the micros() function before and after the calculation of the dot product as shown below.

```
long dotproduct(int length, int *a, int *b) {
microseconds1 = micros();

     long sum=0L;       // Initializing long
     for(int i=0;i<length;i++)
     sum+=(long)a[i]*(long)b[i]; //type casting
   microseconds2 = micros();
   time_elapsed = (microseconds2 - microseconds1);
   Serial.println(time_elapsed);
     return(sum);
}
```

4. The variable `time_elapsed` will contain the execution time in micro seconds.
5. Now let's repeat the process for different size dotproducts and measure the execution time.
6. Replace the value of `MAX_SAMPLES` with different sizes (in powers of 2, like 64, 32,…,2) as shown below.

```
        #define MAX_SAMPLES 64
```

7. Try increasing the number of samples to 264 and run the program. Report the console message and reason behind it.
8. Tabulate and plot the values (time vs MAX_SAMPLES). Show the TA the plot before you continue.
9. Review the plot and identify an equation that approximates the curve.

Lab – Section 3: Computing the runtime for the Dot Product

- In this section we will go through the assembly code generated and check the number of cycles each instruction takes
- Finally, you'll calculate the runtime of the dotproduct

1. Refer to the dotproduct Assembly code generated and extracted in the pre-lab.
2. Now refer to the Instruction set of the Arduino's Micro-controller to see how many clock cycles each instruction takes (For example, add, mul, ld, mov)
   a. Open the datasheet of the ATmega328 by clicking on the link below. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf
   b. Go to Section 36 Instruction Set Summary
3. Next calculate clock cycles.
   a. Total Cycles is defined by the following

$$\Sigma_i (IC_i)(CC_i)$$

Where IC_I is the number of instructions for a given instruction type i, CC_i is the clock-cycles for that instruction type. The summation sums over all instruction types for a given benchmarking process.

   b. Make a spreadsheet table with one line for each instruction in `dotproduct` and `__umulhisi3` (and all the routines that `__mulhsi3` calls).
      i. Add column to the table for cycles for each instructions
      ii. Add column for number of times it is called.
      Hint: the number of times some instructions are called will depend on the value of MAX_SAMPLES, so actually create two columns here
         - Number of multiples of MAX_SAMPLES the instruction is called
         - Number of times called that is not a multiple of MAX_SAMPLES

      So, together, you can express every instruction as a linear equation:
      a×MAX_SAMPLES+b
   c. Put MAX_SAMPLES in a spreadsheet cell.
   d. Add an equation that uses the table values to calculate total cycles
      Make this use the MAX_SAMPLES cell so you can get it to compute a Total Cycles estimate for each of the different MAX_SAMPLES values you previously measured.

4. Add an equation using the above to calculate the Execution time using the formula below:

   Execution Time = $\textit{Total Cycles} \times \textit{clock time}$

Here the clock frequency is 16Mhz

5. Compare the time obtained in this vs the time obtained from section 2.
6. Show the TAs the value and explain your discrepancy. This is your exit check off.

Postlab

1. Reviewing the Instruction Set table of ATmega328.
   - Why does the MUL takes more instruction than ADD?
2. If we were to design an enhanced Arduino (ATmega328) that included a hardware 16b×16b multiplier that was supported by a new instruction that could multiply two unsigned 16b integers in one cycle, how much faster would the dotproduct run on this enhanced Arduino compared to the Arduino you used in lab and evaluated in Section 3?
   - This instruction will replace the call to __umulhisi3. So, all the cycles that you calculate for making the call to __mulhisi3 and the cycles for the call will be replaced by a single cycle.
3. For what MAX_SAMPLE window sizes can you sample data and compute the Fourier Transform in real time?
   - For our 5000 samples per second rate, calculate how many milliseconds the Arduino has to compute between samples. Call this ctime.
   - From the lab, you know how many milliseconds it takes to compute the dotproduct for a particular values of MAX_SAMPLES. Call this function dp_time(MAX_SAMPLES).
   - Further, you know it takes roughly 2×MAX_SAMPLES×dp_time(MAX_SAMPLES) to compute the Fourier Transform.
   - When is 2×MAX_SAMPLES×dp_time(MAX_SAMPLES) < ctime×MAX_SAMPLES?
   - How does the maximum MAX_SAMPLES window size change if we sample at 1000 samples per second?

HOW TO TURN IN THE LAB

- Each student should assemble an individual writeup and upload as a PDF document to canvas, containing:
  - dotproduct and __mulhisi3 extracted code from prelab
  - Arduino C code time calculations
  - Table and plot of different FFT MAX_SAMPLES vs time taken for execution.
  - Calculation of your Total Cycles with spreadsheet table and any assumptions made
  - Calculation of Execution time and inference from the comparison
  - Answers to all questions in the lab
  - Answers to post-lab