Real Time Digital Signal Processing

Lab 1: Introduction to The Lab Kit

Group 10: Marcos Delgado (GitHub ID: marcosdelgadorosa), Brett Mann (GitHub ID: BrettMann225)

November 4th, 2025

Table of Contents

Table of Figures

Introduction

This lab's purpose was to build familiarity with the hardware and software that will be utilized throughout the class. Additional basic experiments were run to verify equipment was set up properly and outlined topics briefly discussed during lecture such as signal sampling and reconstruction. The lab follows the function of the analog to digital converter (ADC) and the digital to analog converter (DAC) in the signal processing procedure.

Methods

This lab utilized a provided DSP kit consisting of a RPI4 board and a shielded PCB with a high-end DAC and ADC. An Analog Discovery 2 USB scope was used to monitor signals. The appropriate software was installed on all the hardware. An SSH connection was used to access the RPI4 and run necessary programs such as pipewire.

Results

Question 1:

Our first goal in this lab was to evaluate the performance of the RPI4 using the Passmark performance test. The test revealed that our hardware had the following metrics: A clock frequency of 1800MHz, a core count of 4 cores, 4526M Floating point operations per second, 1131.5M operations per core per second, and a memory bandwidth of 3003 MB/s. The direct output of our performance evaluation can be found in figure 1.

```
                    PassMark PerformanceTest Linux (11.0.1002)

Raspberry Pi 4 Model B Rev 1.5
Cortex-A72 (aarch64)
4 cores @ 1800 MHz   |  3.7 GiB RAM
Number of Processes: 4   |  Test Iterations: 1   |  Test Duration: Medium
--------------------------------------------------------------------------------
CPU Mark:                           634
  Integer Math                      11699 Million Operations/s
  Floating Point Math               4526 Million Operations/s
  Prime Numbers                     4.0 Million Primes/s
  Sorting                           3948 Thousand Strings/s
  Encryption                        78.5 MB/s
  Compression                       8678 KB/s
  CPU Single Threaded               487 Million Operations/s
  Physics                           91.8 Frames/s
  Extended Instructions (NEON)      781 Million Matrices/s

Memory Mark:                        715
  Database Operations               773 Thousand Operations/s
  Memory Read Cached                6088 MB/s
  Memory Read Uncached              3961 MB/s
  Memory Write                      3003 MB/s
  Available RAM                     3360 Megabytes
  Memory Latency                    90 Nanoseconds
  Memory Threaded                   4011 MB/s
--------------------------------------------------------------------------------

Results not submitted
Upload results to cpubenchmark.net? [Y/n]:

Use ESC or CTRL-C to exit
A: Run All Tests   C: Run CPU Tests   M: Run Memory Tests   U: Upload Test Results
```

Figure 1. Results of Passmark Performance Evaluation on RPi4

Question 2:

To determine how many DSP operations our hardware can achieve per second we can use the metrics in figure 1. Here we know we need two operations to happen per DSP operation. We also know that a single core can run 1131.5M operations per second, which means in theoretically we could achieve 565.75M DSP-operations per second per core. However, we must first verify if we have enough memory bandwidth to handle that quantity of DSP operations. The RPI4's memory bandwidth is 3003MB/s – each DSP operation will require 4 bytes to read and 4 bytes to write for a total of 8 bytes per DSP operation. With the memory bottleneck we can achieve 375.375M DSP-operations per second, and 93.8M DSP-operations per second per core.

Question 3

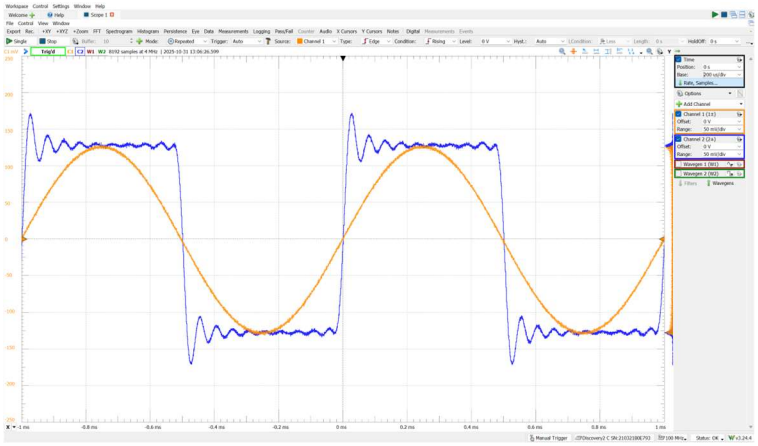

Figure 2. Scope output shows clean sine wave and ringing square wave

Sending a sine and a square wave through our device reveals a very clean sine wave and a ringing square wave, as depicted in figure 2. The cause of the imperfection in the square wave is likely to due to the reconstruction filter present in the DAC which filters out the high frequency harmonics of the square wave causing an imperfect analog signal.

Question 4

To connect the board's inputs with code, a function built into pipewire must be used called pw-link. To connect the left and right the following lines were run:

pw-link alsa_input.platform-soc_sound.stereo-fallback:capture_FL alsa_output.platform-soc_sound.stereo-fallback:playback_FL

pw-link alsa_input.platform-soc_sound.stereo-fallback:capture_FR alsa_output.platform-soc_sound.stereo-fallback:playback_FR

Question 5

Figure 3. FFT spectrum of square wave.

Further investigation of the square wave reveals an unexpected Fourier transform. The expected Fourier transform of a square wave is a smoothly attenuating set of ripples, however as seen in figure 3, the Fourier transform of our signal yields a ripple that stops suddenly and then experiences another, attenuated band of ripples at higher frequencies. The reason for the sudden cutoff is that the DAC has a lowpass filter as it reconstructs the signal, attenuating the higher

frequency ripples. The second image at ~370kHz is due to the sampling which generates smaller image bands at periodic intervals.

Question 6)

To further explore the imperfection of the square wave reconstruction we modified the code to print out the maximum and minimum values of the left and right channels as well as the frame size that pipewire was using. As shown in figure 4, the frame size is 1024 samples and the left channel goes from -0.883578V to 0.882495V and the right channel goes from -0.667349V to 0.665873V.

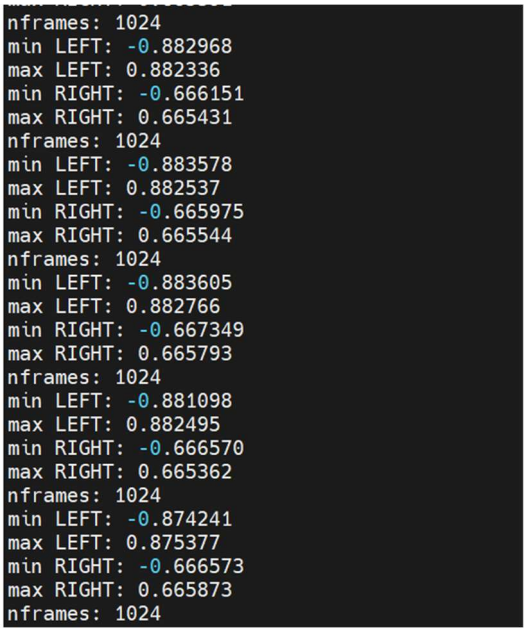

Figure 4. Outputs depicting the log of minimum, maximums and frame size. Sine wave in left channel and square wave in right channel.

Question 7

Figure 4 shows that the maximum and minimum values of the square wave and the sine wave are not the same with the right channel (square wave) being of significantly lower amplitude. The square wave has an infinite Fourier transform as it is a sum of infinite harmonics, however most of the higher frequencies will be filtered out by the ADC in the anti-aliasing filter and again in the DAC at the reconstruction filter, leaving the square wave with a much more attenuated signal compared to the sine wave. The sine wave has most of the frequency information focused at one point and therefore is less affected by the filters.

Question 8)

```c
#include <string.h>

#include <jack/jack.h>

jack_port_t *input_port_left;
jack_port_t *output_port_left;
jack_port_t *input_port_right;
jack_port_t *output_port_right;
jack_client_t *client;

int indexleft = 0;
int indexright = 0;

int process (jack_nframes_t nframes, void *arg) {
  jack_default_audio_sample_t *outleft, *outright;

  outleft  = jack_port_get_buffer (output_port_left, nframes);
  outright = jack_port_get_buffer (output_port_right, nframes);


for (int i = 0; i < nframes; i++) {
 float phaseL = (float)indexleft / (float)(48); //48 samples per cycle, 24 samples for rising, 24 for falling. goes from 0-1 every 48 samples

 if (indexleft < 24) {
        outleft[i] = 2*phaseL - 0.5f;
 }  else {
        outleft[i] = (1-2*phaseL)+0.5f;
        }
 indexleft = (indexleft+1)%48;
}

for (int i = 0; i < nframes; i++) {
  float phaseR = (float)indexright / (float)(24);

  if (indexright < 12) {
        outright[i] = 2*phaseR - 0.5f;
 }  else {
        outright[i] = (1-2*phaseR)+0.5f;
        }
  indexright = (indexright+1)%24;
}

  // define a function that writes nframes of samples
  // to the left and right buffer according to
  // the specifications of the lab

  return 0;
}
```

Figure 5. Triangle generator code

Figure 5 depicts the code used to generate two triangle waveforms. This is done by calculating the phase and setting it to the corresponding values for each frequency. As shown in Figure 6, the left channel generates a 1kHz triangle wave and the right channel generates a 2kHz triangle wave.

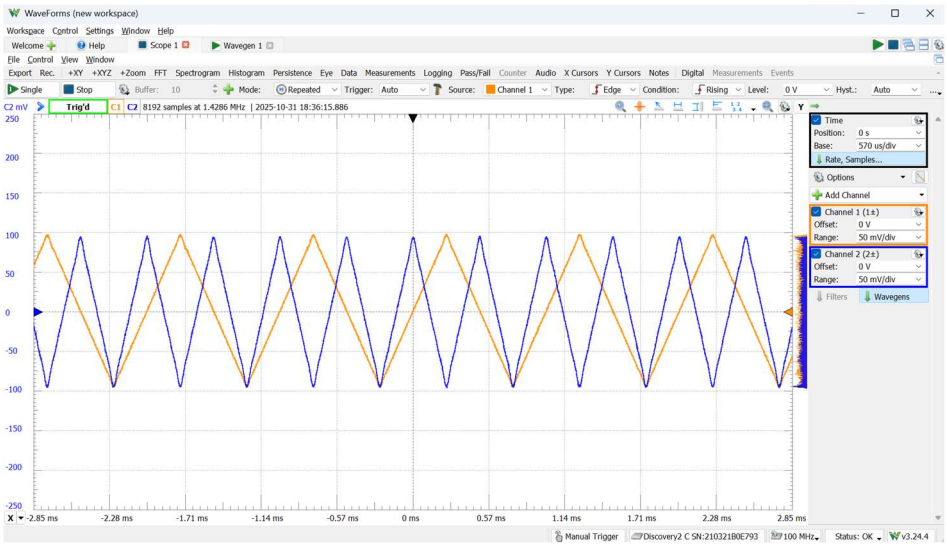

Figure 6. Output of the triangle generator. Left channel (orange) 1kHz triangle wave and right channel (blue) 2kHz triangle wave.

Conclusion

This lab assignment provided great insight into the lab kit we will be using throughout the rest of this course. It required us to familiarize ourselves with new hardware and software through the writing of algorithms to both visualize hardware imperfections and generate our own signals.

References

The use of external references was not required for the completion of this lab assignment.