Summary Document

Group 6

Bar Peretz
University of Sydney
e: bper2590@uni.sydney.edu.au

Kai Lian Leong
University Of Sydney
e: kleo6049@uni.sydney.edu.au

Brendan Guan
University Of Sydney
e: bgua5258@uni.sydney.edu.au

I. Introduction

In this document, the design of a software defined radio (SDR) for the purpose of decoding WSPR signals. Given a prototype "SDR 1.0", improvements were made such that it would be more suitable as a WSPR decoder, as well as built using the Raspberry Pi microprocessor (as opposed to the beaglebone). Open-Source code which decodes signals provided in a "wav" file was also modified such that the program can decode a live signal detected by the microphone using pulse audio.

II. WSPR Decoding software (Bar Peretz & Brendan Guan)

What was developed? (Changes made):

Using a provided template for a program, written in C, several files needed to have been modified such that it could decode a signal which is fed directly through the microphone of the device, wait two minutes (on each even minutes) and decode another signal and so on. The files edited were labeled "parec.c", "wsprd.c", "wsprwait" as well as the "Makefile", which runs the code. Firstly, the "parec.c" file needed to be completed, this code can read the live data through pulseaudio, next the "wsprd.c" file (which handles the decoding of the signals) needed to be edited such that it can handle live inputs, simply by creating a case where it calls the function written in "parec.c".

The wsprwait file is responsible for ensuring that the program is run only on each even minute (sometimes a cycle is skipped due to the time it takes for the program to be run). It does so by checking the time (linked to UTC) to see when an even minute is hit, and then running the program (this continues an indefinite loop)

Finally, the Makefile was edited to enter the pa-wsprcan directory and run "wsprd.c" with the live input from pulseaudio turned on.

How was this tested?

The changes to the code were tested with a real time WSPR signal played out loud while running the program using the provided "m2-strean" and "m2-test" files. The former detects the microphone input, and the latter processes the signal detected. The code working correctly should display the decoding to the console

What problems occurred?

When the code is initially run, a lot of warnings appear on the console before the signal is decoded. Although they dot create a problem for our particular case as the program ran correctly, It demonstrates that the code could have been rewritten into a more concise and easier to read form, which may avoid issues for other applications of the program.

III. PCB changes (Kai Lian Leong, Bar Peretz)

Kicad – Schematic

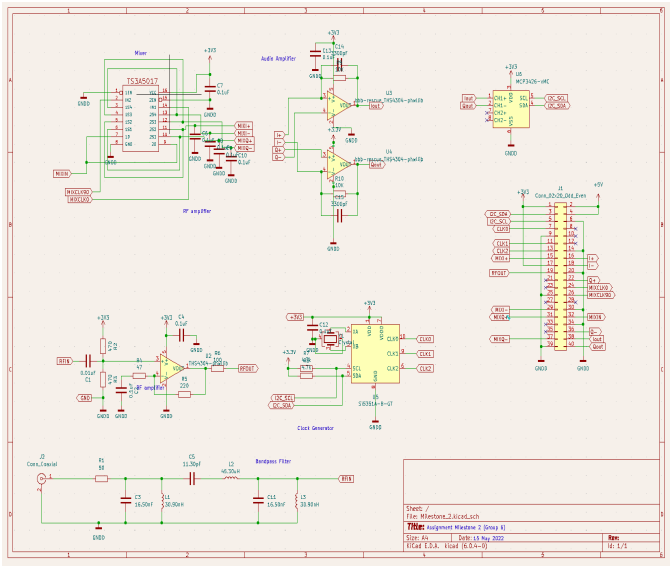

Fig. 1. SDR Schematic

SDR has seven main components: Bandpass Filter, RF amplifier, Clock Generator, Mixer, Audio Amplifier, Rasberry Pi pinout (40 pins), and ADC.

In terms of hardware, the PCB will first receive the WSPR signal pass it through the bandpass filter as we are only interested in the frequencies the WSPR signal is at. It will then be amplified before going into the mixer which will give us two analog signals, one which is called the "I" signal which is the in-phase signal, and the other is the "Q" signal which is the quadrature signal. These two analog signals

will be amplified and will then be converted into a digital signal for our computer to process it as computers are only able to process discrete values.

When running the electrical rules checker, just look through each error and warning and fix it accordingly. However, at the end of our schematic, we have a few warnings telling us that our pins are not driven by other pins. This is because Kicad does not know out intended use, so we could add a power flag to solve these warnings.

Kicad – PCB editor

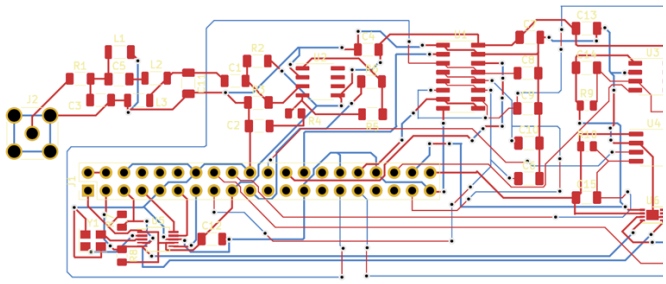

Fig. 2. PCB Editor

Once we update PCB from Schematics, we can begin organizing the components such that we will be able to connect the components with the tracks. For instance, we may encounter such an issue when there is an insufficient amount of space to put down tracks to connect the components. Lastly, we run a Design Rule Checker which would tell us what error and warning Kicad has detected. In our case, we did not have any warnings or error, which meant that our design was ready to move onto the printing stage.

IV. LT-SPICE SIMULATIONS (BRENDAN GUAN, KAI LIAN LEONG)

The SDR 2.0 was implemented into LTSpice and simulations were run accordingly. In our schematic, we connected the bandpass filter, RF amplifier, Tayloe Detector and Audio Amplifier into a working circuit. The ideal component for the RF amplifier was replaced with THS4304 model component acquired from Texas Instruments. The ideal component for op-amp in the audio amplifier was replaced with TL972 model components also acquired from Texas Instruments.

We ran into issues implementing the Mixer in the form of TS3A5017, where output signals were not what we had expected. To ensure a working simulation, the ideal component for the Tayloe Detector was left in. Compared to the ideal simulations from Lab 3, q_out lags by 90 degrees as expected. The observed peak to peak for both outputs in the non-ideal simulation (0.738V-2.36V) are larger than the peak to peak observed in the ideal simulations (1.193V-2.1V), however the waveform itself is consistent between both simulations.