

# **Embedded Systems 2.**

## **“Practice 2: Wireless speaker”**

# **Professor: Rodrigo Aldana López.**

# 

# Gustavo Araiza Obeso ie699209.

# Celis Tapia Daniel ie689705.

# 05/01/2018

## INTRODUCTION:

This practice shows the use of two different sockets, an UDP and TCP socket, and a DAC conversion task. The music is transmitted as raw data from a Python UDP socket, while the TCP menu is sent to a terminal on a cellphone to control the music reproduction. The DAC runs indefinitely, converting the received data on the buffers to a stream of outputs to an analog filter, which in turn is connected to a speaker.

The wireless aspect of the speaker comes from the need of a router to allow the communication between the audio source (UDP socket), the client (TCP socket), and the server (the K64 board), which in turn manages the listenable output.

## Theory aspects

Transmission Control Protocol accepts data from a data stream, divides it into chunks, and adds a TCP header creating a TCP segment. The TCP segment is then encapsulated into an Internet Protocol (IP) datagram, and exchanged with peers.

A TCP segment consists of a segment header and a data section. The TCP header contains 10 mandatory fields, and an optional extension field (Options, pink background in table).

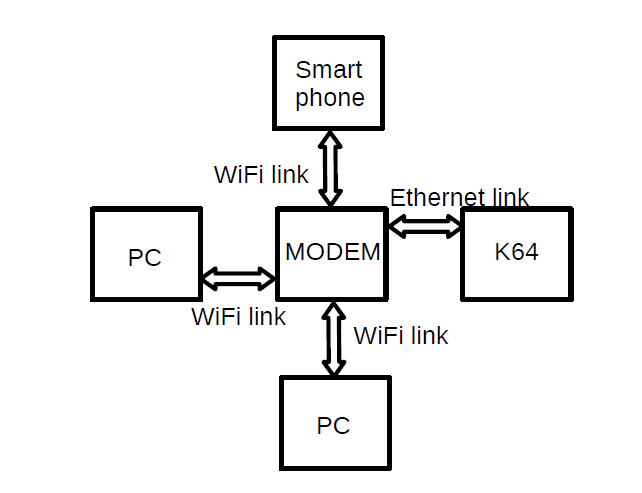
TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network. Major Internet applications such as the World Wide Web, email, remote administration, and file transfer rely on TCP. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.

UDP uses a simple connectionless communication model with a minimum of protocol mechanism. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network; There is no guarantee of delivery, ordering, or duplicate protection.

UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing in the protocol stack. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed due to retransmission, which may not be an option in a real-time system.

## Project planning:

As mentioned early, the project consists of three main modules to allow the wireless communication to a DAC output:



# Functional description

The project may be divided into four main stages:

*UDP transmission*: The UDP audio transmission is composed of two different stages which interact between them. The first stage is a socket in Python which converts, samples, and indefinitely sends the data to a port in the K64 board IP address. Since UDP does not require a handshake for the packages, the program may keep transmitting even if no one is listening. On the other hand, the K64 board has a two-part buffer which receives the data and allows the DAC to read the other buffer. The data is received once the port is assigned to be listened, then the data is processed out.

*TCP transmission*: The TCP socket helps the user control the project throughout a menu. Three options are offered: to pause or resume playing the current song, to switch the UDP port that is being listened to, to check the health of the UDP transmission. This socket may connect to an app on the user’s smartphone.

*DAC* *output*: The data received by the UDP buffers must be processed out at the sampling rate which was stablished on the transmission side. To do so, a PIT timer is enabled to take and process a sample every period. Once a buffer is processed completely, the next one is taken and the system is signaled that the previous buffer is available for data reception.

## Some numbers:

Our system can receive at a healthy 8.84 KHz sampling rate without any major hiccups. Each packet is sent with 4096 bytes, which represent 2048 samples, as each sample is composed of 12 bits. Every second, 8.61 packets are sent over UDP per the Python sockets. The played music must then be filtered out through a 5 KHz low-pass filter as the sampling rate creates a need to reduce the higher frequencies.

## Development:

No major elements were required that have not already been mentioned. The sockets were created from the SDK examples, as the startup sequences are the same in any case. The tasks that handle the data communication were modified to fulfill the needs for the system.

The TCP task includes now a menu which checks the input to allow the user the options. The play option changes a flag which is needed to allow the DAC to access the data inside the buffers. The port option changes the binded port in the UDP task to retrieve the data from the now listened port.

The UDP task takes the data onto one of the two buffers, while protecting the data copying through a series of flags. A buffer may be written if the previous one has been written and if the buffer is not being currently read. Once the buffer is full, the same conditions apply to retrieve data to the second buffer.

The DAC task is nested on the interrupt of a PIT timer, which in turn allows two bytes to be combined and sent as output throughout the channel. The DAC can only read data from a fully copied buffer, and only once the previous buffer has been read completely. These conditions keep data integrity and avoid some sync issues.

## Conclusions:

*Daniel*: This practice allowed us to understand the different uses of each of the internet protocols. UDP is used mostly to keep a steady flow of data, while TCP gives security and stability on the data received and sent. The timing on these tasks was critical, since the audio can fizzle out and the user may detect physically the issues that may arise. As such, it was important to create a robust system that, as it may not go very high on the sampling frequency, it can keep up properly with the sent data. Large buffers gave us the flexibility to manipulate the data without having long waits. This was a very complete exercise that gave me the context needed to understand the use of the protocols.

# Repository link:

The project may be found at the next link:

<https://github.com/GustavoAraizaO/Practica2SE2>

Please do mind that the last commit corresponds to the whole working project.