# COMP2300 A3: Networked Sequencer

Rachel Schroder, u6301105



## Introduction

For this assignment I have implemented a synchronous serial protocol that allows for changes in frequency across multiple channels.

The message format is loosely inspired by the MIDI protocol, but has been simplified significantly. Each message contains the channe for and the frequency of the note that should be played. As in MIDI, there are 16 channe ormat allows for all frequencies in the MIDI note range (8.18 - 12500 Hz) (Inspired Acousti te off, the frequency should be set to 0.

As in assignment 2, the user n (number of segments) and tempo (number of segments per second) of the song, and that should be played. At the start of every segment a message is sent for each cha this assignment was on developing a protocol, I decided not to spend time implementing additive synthesis to allow for one receiver to play multiple parts, so the receiver that I have implemented only listens to one channel, and multiple receivers would be needed to play multiple parts simultaneously. No additional pins are needed to connect additional receivers, but a breadboard would be needed to connect multiple boards to one pin.



I have configured my sequencer to play part of *I Am The Doctor* from *Doctor Who*, using an arrangement by Shelby Deal on Musescore (see Appendix, Figure 1). It is set to play the top part (channel 0), but there are 4 other channels to choose from. The channel can be changed using `receiver_channel`. I chose this song because it has a large range of frequencies (49 - 587.33 Hz), demonstrates my sequencer's ability to work with irregular time signatures, and is easy to split into channels.



## Protocol



Data transfer between the sender and receiver has been implemented as described for the simple serial protocol in the assignment specification.



Each message is 32 bits long, and is formatted as shown in the Appendix, Figure 2.

Bits 28-31 were chosen arbitrarily, and are used to help check that a message is valid.

I allocated 4 bits for the channel number, in order to reflect the 16 channels used in the MIDI protocol.



The frequency is sent in the same form that is used in part 1, so that the full accuracy of the given wave library could be utilised. The frequency of the highest MIDI note (127) is 12500 Hz. This would be transmitted as 1,250,000, which uses 21 binary bits. I originally chose to allocate 24 bits as I thought using a whole number of bytes would simplify the process of generating and interpreting messages, however since implementing this protocol I have realised that it would have been just as simple to use 21 bits.

## Implementation

I have divided code into sender and receiver sections using comments to help me keep the code separated, however there is nothing enforcing this split.

The priorities for the interrupts are in the following order (highest to lowest): clock line, control line, `TIM7_IRQHandler`. The clock and control lines are above `tim7` as they both need to be triggered while the messages are being generated and sent.

### Sender

At the start of every segment, the `TIM7_IRQHandler` first generates the appropriate message for each channel, adding them to the stack, then once they have all been generated sends them out. The generation and transmission of each message are separated to reduce the time between receivers receiving the message for their channel, so that they can better play in time with each other.

Each message is generated according to the format described above. They are sent according to the process described in the assignment spec.

There are three data structures used by the sender:

- `song_info`, an array used to control the tempo, duration and number of parts in the song.
- `channel_index`, used to track progress through the song.

- `channel_data`, an array that stores the notes for each channel. The format of this array is shown in the Appendix, Figure 3.

Because the data for each channel is only split by comments, there needs to be exactly one entry for every segment, otherwise the sender will not read the correct frequencies when generating the messages.

I chose not to include the duration of the note in the array because I prioritised implementing the protocol over recreating something that I had already demonstrated in assignment 2. However, given more time, including the duration would ents to my sequencer. This is further discussed in the improvements section below



## Receiver

The receiver also behaves as ignment spec.

On a rising edge on the contr interrupt is enabled, and on a falling edge it is disabled. The clock line interrupt triggers o

The only alteration to the described process is that when a falling edge is detected on the control line, the message is interpreted and enacted upon immediately, rather than storing it for later processing. From testing with 16 channels enabled at a tempo of four segments per second, the receiver seems to be able to process incoming messages fast enough that they do not need to be stored. This is further aided by the receiver disregarding any messages that are not intended for it before trying to interpret them.



Once it has been determined that the message is intended for the correct channel, the receiver checks that it is valid. To be valid, the message must start with `1100` and the message length counter must equal 32. If either of these conditions is not satisfied, the receiver exits the handler.



A valid message is enacted upon by isolating the three bytes that contain the frequency, getting the amplitude from memory, then calling `wave_change` to alter the sound being played.



There are four variables in memory for the receiver:



- `amplitude` controls how loud the sound is and is currently set at 25000 (three quarters of the maximum amplitude).
- `current_message` stores the bits that have been received for the current message. It is cleared when the control line is set, read when the control line is cleared, and updated when a bit is received from a rising edge on the clock line.



- `receiver_channel` is used to configure which messages the receiver listens to.
- `message_length_counter` tracks how many bits have been received so far, and is used when checking if a message is valid. It is reset to 0 when a rising edge on the control line is triggered, and incremented when a bit is received.

## Improvements

The main improvement that I would make aims to reduce the amount of clicking sounds in notes that play for more than one segment. This could be done in two ways, one through the sender and the other through the receiver. The first would be to only send messages when the frequency changes between segments. This could be made easier by altering the structure of `channel_data` such that it also stores the duration of the note, similarly to the structure that I used for assignment 2 part 2. The second would be to store the current frequency, and only call `wave_change` when the frequency changes. The latter solution would be easier to add to my current implementation, however the former would reduce the time spent sending and generating messages.

The second improvement would either be to make the message format more compact, and thus spend less time sending messages, or to add extra information to the message, such as a change in amplitude or timbre, so that the 32 bits are being used more efficiently.

Another way to improve the efficiency of message dispatch would be to connect each receiver to a different data wire. Then, for each bit in the message, all of the wires could be written to, and then the clock line toggled once to send the data to all channels simultaneously. This has the advantage of reducing the amount of messages being received by each receiver and reducing the chance of a message being missed. However, it has the disadvantage of limiting the amount of receivers based on the amount of available pins.

## Reference List

Deal, S., n.d.. *Fandom Medley.* s.l.:Musescore.

Inspired Acoustics, n.d.. *MIDI note numbers and center frequencies.* [Online]
Available at: https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies
[Accessed 6 June 2020].

## Appendix

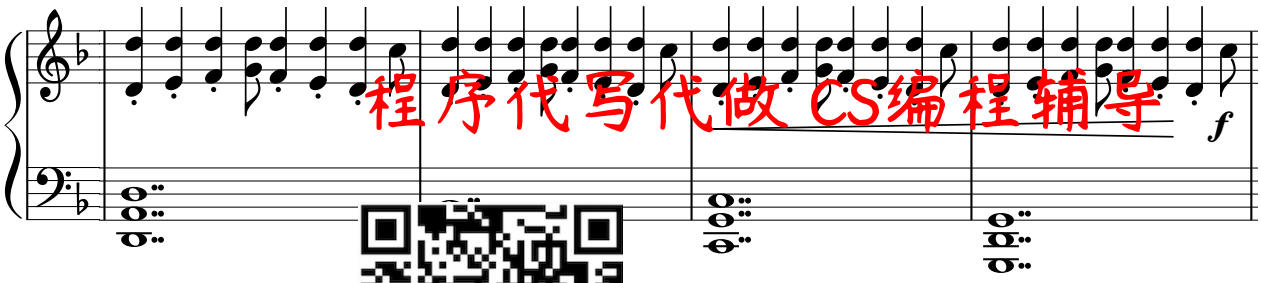

*Figure 1: I Am The Doctor from Doctor (n.d.)*

| Use | 1100 | Cl | Frequency (Hz) * 100 |
|---|---|---|---|
| Bits | 31-28 | | 23-0 |

*Figure 2: Message format*

| Channel number | Element (stored as a word) |
|---|---|
| 0 | 44000 |
| | 0 |
| | 44000 |
| | 0 |
| 1 | 88000 |
| | 0 |
| | 88000 |
| | 0 |
| 2 | 22000 |
| | 0 |
| | 22000 |
| | 0 |
| 3 | 11000 |
| | 0 |
| | 11000 |
| | 0 |

*Figure 3: channel_data example format for a song with 4 parts and a 4 segment duration*