Harriet Phillips [redacted]  9th June 2020

# COMP2300 Networked Instrument Part-2 – Simple Serial Protocol

## Overview

My part 2 of Networked Instrument is an implementation of a simple serial protocol which uses 3 wires – a control wire, a clock wire and a data wire.
My main goal when choosing what to create for part 2 was to implement a protocol with enough flexibility to enable the user to change what song was played and the characteristics of that song easily. This lead to the decision to use a serial protocol to send the data required for the song over the lines enabling many possibilities.
I decided to focus on being able to change the frequency of the wave first, and then added the ability to change the amplitude. I chose these properties as they would enable the ability to play a harmony using additive synthesis, which could be implemented in the future.

## Implementation

### Storage of note sequence

The data for the song is stored as 23 bit numbers in an array. The first 15 bits determine the amplitude and the last 8 bits determine the frequency. These values were chosen as 0x7FFF (the maximum amplitude) in binary is fifteen consecutive 1s, and each of the frequencies I needed to play are under 255, the max number for 8 bits. These numbers could easily be modified in the future to allow higher frequencies/a higher amplitude if need be.

### Loop

**Receiver:** The receiver part of loop plays the most recently received note.
**Sender:** The sender part of the loop checks the state of the control line. If it is on, it starts sending the data for the next note.

### Send

Send handles the sending of data across the data line and the toggling of the clock line so the receiver knows to read in the next bit. It loops through the current message to be sent and sets the data line to be high for a 1 or low for a 0 and then toggles the clock line until it has sent all the required bits. I chose 1 to be high and 0 to be low as 1 is a larger number than 0, although they are easily interchangeable. It also means that when the clock/data wire is disconnected it only plays silence, as it only receives 0s (data) or nothing (clock).

### Finish Sending

Finished sending clears the clock line so the sender and receiver know that the message is finished. It also updates the variables required to send next message correctly.

### Timer

The timer is triggered every 0.25 seconds and turns on the control line, triggering the next message to be send an the wave to be updated. The value of 0.25 seconds was chosen as it's the minimum time between notes in the song played, although it could be easily altered.

### Interrupts

**Control Line Receiver Interrupt**

If the control line was triggered by a rising edge, then the receiver knows a message is about to be sent. If it was a falling edge, the receiver knows the message is finished, and therefore it updates the wave using the message stored in memory. The first 15 bits of the message is set as the new amplitude, and the following 8 bits are set as the new frequency.

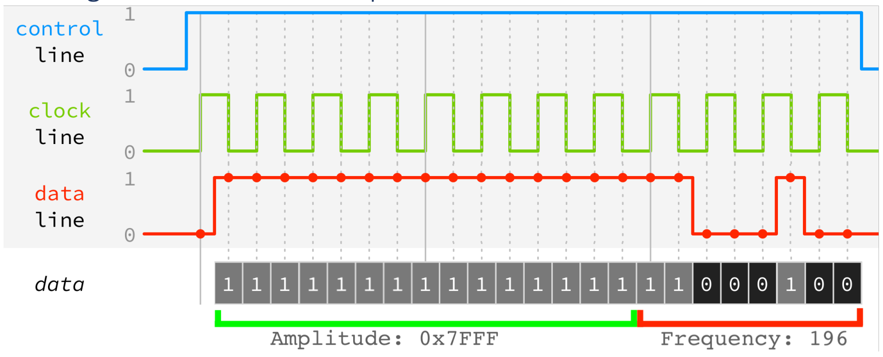
**Data Line Receiver Interrupt**

When a data line interrupt happens, the current data line value is updated in memory. If it was triggered by a rising edge a 1 is stored. If it was triggered by a falling edge a 0 is stored.

**Clock Line Receiver Interrupt**

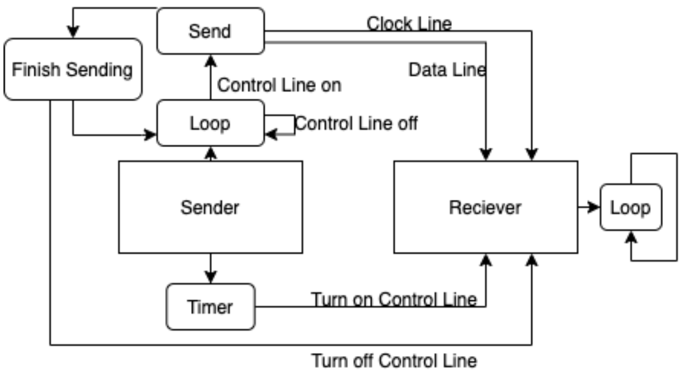
When a clock line receiver interrupt is triggered the current data line value is read in from memory and is appended to the end of what has already been read in.

## Message Transmission Example



This is an visualisation of how the first note in the sequence is sent over the wires, edited from the image at [1].

## Diagram



This is basic diagram that shows how the different parts of the program connected. From it you can see how the sender and receiver parts are separated. It also shows what parts of the program they interact through, and what controls those interactions. For example, it can be seen that the sender's timer controls when the control line is turned on, and finish sending controls when it is turned off.

## Reflection

**Limitations and improvements:**

As mentioned above, the current implementation is limited to a frequency of 255 Hz and an amplitude of 0x7fff. While these values work well for the song I chose to play, many songs have notes above 255 Hz. A simple improvement to my design would be to allow amplitudes up to around 4186 Hz, which is the highest note on a piano [2]. This would require 13 bits instead of the current 8.

There are also many other improvements that could be made by altering the messages sent and how the receiver interprets them. One of these is allowing different note shapes. For example, you could send 00 for a sawtooth wave, 01 for a triangle wave and 10 for a square wave. Allowing each note to have a custom duration gives a lot more flexibility with what can be played. Another probably more useful improvement would be to allow custom lengths for each note. This could be achieved by re-initializing the timer with the new time, much like initializing a new wave with each frequency/amplitude read in.

**Difficulties**:

The main difficulty I had was with getting the receiver to read in the next note while continuing to play the current note correctly. This caused strange artefacts to occur in the sound, so I decided to load the note after the previous note finished. This isn't an issue with the song played, as there isn't a noticeable gap between notes ending and beginning. However, if the song had shorter notes it might become more apparent.

Overall, I would say that I successfully met my goal of implementing a flexible protocol that enables users to easily customise the song played.

## Testing Instructions:

if you disconnect the control line then the program will continue to play the current note forever, as the message is never updated. Disconnecting the other lines after the control line will also result in the current note being played forever as well, as the message will still not be updated.

If you disconnect the data line/clock line the message being sent/received will remain 0, resulting in silence.  Disconnecting either the data or clock lines followed by the other lines will also result in silence. Again, this is because the message is never updated.

## References:

Message transmission original diagram:
[1] https://cs.anu.edu.au/courses/comp2300/deliverables/03-networked-instrument/

Highest note on a piano:
[2] https://www.audiology.org/sites/default/files/ChasinConversionChart.pdf