Gerard Huang (u6080257)

Design Document

Intro

This document will provide insights into the methodology and implementation used to write an extended version of the P2300 protocol (implementation by *Ben Swift, 2017)* using assembly which involves the addition of a secondary data line used to send out a 16-bit message. The protocol implementation utilizes functions, GPIO interrupts and memory based data structures. Testing the protocol was done by playing the bassline to 2pac's "Do for Love" using square waves sampling at a 192 kHz rate. This implementation was done using the STM32L47b Disco Board.

1 Design Overview

Within this section the design for the extended P2300 protocol will be covered. The extended P2300 program is designed to send two 16-bit messages through GPIO pins using jumper cables. The aim was to translate the messages into note frequency and amplitude. Since the hearing range of the human ear is 20 Hz – 20,000 Hz (*Smith, 1997)*. Only 16 bits per message would suffice since any frequency represented larger than 16-bits would not be audible to the human ear.

1.1 Protocol Structure

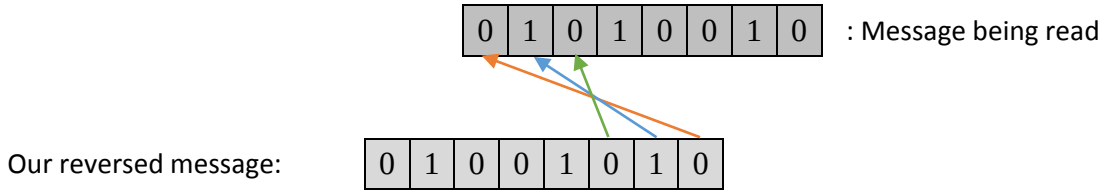
The protocol utilizes 4 wires which consist of a control, clock and 2 data lines. The wiring details will be listed below:

- Control line: **PE12** (sender) to **PH0** (receiver)
- Clock line: **PE13** (sender) to **PH1** (receiver)
- Data lines:
    - Frequency: **PE14** (sender) to **PE11** (receiver)
    - Amplitude: **PE15** (sender) to **PE10** (receiver)

Contrary to the normal P2300 protocol, the extended implementation involves the addition of an extra data line used to transmit the amplitude value while the other data line sends the frequency, both messages are sent as 16-bit messages. Having a 16-bit message representing amplitude is appropriate given that the maximum (semi) amplitude range of 0x7FFF does not exceed any value above 16-bits.

1.2 Data Structure

The messages that were sent over the data lines were stored in memory were frequency were stored in a sequential fashion such that each successive note in the song is one word apart. The very first entry stored in memory is the amplitude which never changes throughout the song therefore it did not require any duplicates within the data structure. All the entries inside the data structure are represented in binary however a caveat is that they are the *reversed* binary representation of their literal decimal equivalent. For example a frequency for the E (3rd Octave) note is represented as 01010010 (82 Hz), however when stored in memory the bit string is reversed: 01001010. This is due to the implementation of the program where the read function reads from the least significant bit to the most. If the regular binary representation were to be stored then the final message would be reversed. E.g.



1.3 Reading and Processing

When the message is being sent one bit at a time a concurrent process is also run where the sent bit is also read inside the same sub-routine. This comes in the form of nested interrupts where the EXTI0 interrupt handler triggers the send functions and within the send function the EXTI1 interrupt handler is triggered where inside, it reads a bit and sends it into a register and to be processed.

Processing the message involves taking the recently read messages and shifting the registers. These values are then passed into registers 0 and 1 to then be queued using the macro *change_square* then played using another macro *play_sqaure*.

Alternate methods such as a more macro centric program was considered but was disregarded. The extended protocol requires many registers (especially with two data lines). With constant overwriting of the registers, management of the registers can become cumbersome. Having a more function based approach means isolation and temporary reservation of registers within processes become much simpler given that each function can run its task and not have any side effects due to register values being incorrect or overwritten during runtime of the program as a whole. The inclusion of a data structure where the messages are stored in memory allows for scalability and was deemed an appropriate choice should the need of playing extraordinarily large songs is required.

2 Results

This section will discuss the results of the program when reading two 16 bit messages through two data lines to play 2Pac's "Do for Love" bassline using an extended version of the P2300 protocol written in assembly using the STM32L47b Disco Board.

The final messages being sent through the data lines were all read and processed with slight discrepancies with the given amplitude. The amplitude contained an extra bit which was not specified within the data structure. With the expected amplitude value being 011111111111, the actual value being read was 111111111111. A possible culprit to the bug was likely a logical error with the initial truncation of bits during the start of the reading/writing. The solution was during the processing phase the amplitude bit string was logically shifted to the right by 1 extra bit. And as a result the program was successfully able to read, process and play the given messages.

3 Conclusion

In this document the methodology and implementation in designing an extended P2300 with two data lines was discussed. The design revolved around functions, GPIO interrupt handlers and data structures. When tested the program was successfully able to read 16-bit messages on both its data lines as well as translate the messages to play the bassline to 2Pac's "Do for Love". A logical error occurred the reading process resulted in a misplaced bit inside the amplitude message, this was corrected by shifting an extra bit to truncate the unwanted bit. The main takeaways from this assignment includes control flow, concurrent interrupts and protocols. Any extensions to assembly program that has been discussed in this document would include more abstraction with the reading and sending functions to reduce verbosity. In addition, more wires would be added to include more messages to describe attributes such as note and rest duration.

References

- Swift, B. (2017). *Assignment 3: Networked instrument - Computer Organisation & Program Execution*. [online] Cs.anu.edu.au. Available at: https://cs.anu.edu.au/courses/comp2300/assignments/03-networked-instrument/ [Accessed 21 May 2017].
- Smith, S. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing's Table of Content*. [online] Dspguide.com. Available at: http://www.dspguide.com/pdfbook.htm [Accessed 28 May 2017].