Design Document of Assignment 3

COMP2300 – Danny Feng (u6611178)

28 May 2019

1 Overview

Introducing DF19, a half-duplex network protocol with error checking and disaster recovering. It is a brand-new serial protocol that only uses four wires to achieve a 2-way communication. The protocol allows sending arbitrary length data with each element of 16 bits long. In this assignment, DF19 is used to send note frequency. When the receiver fully received the message and confirmed the integrity of data, it will start playing the received notes until a new message comes in.

2 Design Decision

2.1 DF19 Protocol explained

The protocol uses four lines to establish two-way communication. The receiver will always check the control line to know if it needs to record the coming data. Sender pluses the clock when a new bit of data is shipped to the data line and the receiver will then pick it up. After sending each 16-bit data, another 16 bit XOR checksum calculated by the sender will be sent immediately. The receiver will also calculate the checksum and compare with the one sender generated. If the receiver finds out the checksum does not match to previous 16 bits data, a resend request for will be fired.

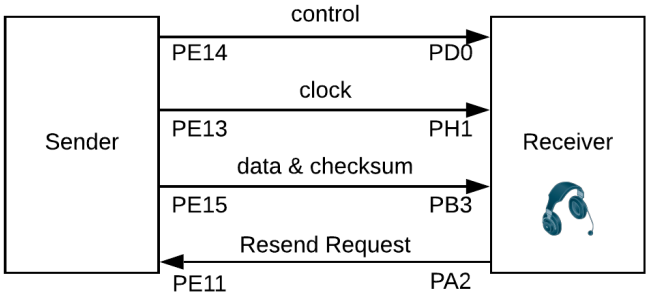

Fig.1 DF19 communication lines

A DF19 packet is made of three components. The first element is always the length of the data, followed by the second and third components, which are the actual data and the data XOR checksum. The decision of making each element halfword (i.e. 16 bits) long is because DF19 is designed to send frequency notes. The human hearing range is between 20 Hz to 20000 Hz. The maximum unsigned integer that DF19 can represent is $2^{16} = 65536$. So, it has the full capability to cover the whole range.

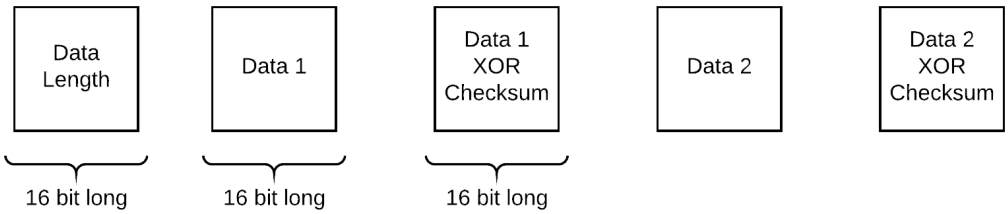

Fig.2 DF19 protocol packet example

2.2 Advantages, Disadvantages and Alternative Solution Analysis

By knowing the length, the receiver can easily allocate necessary memory spaces and data structures beforehand. It also explicitly informs the receiver how many digits should it listen to. Sixteen bits long XOR checksum is used to check the data integrity. However, errors could occur not only in actual data transmitting but also in checksum sending. It might be very inefficient that the data need to be resent again and again. Also, sending such long checksum data, which cutting the transmit speed by half. Using only one parity bit would double the speed, but half of the result could be false positive. The best alternative solution is to implement Hamming (7,4) to detect and correct the data efficiently.

3 Code Implementation

3.1 Architecture

Sender and Receiver's code is separated. Sender's code is in `main.s` while receiver part is in `tim7.s`. This structure offers readability that developers can easily maintain different part of the code and avoid coupling like registers clash. It is not possible to implement higher-level design patterns in assembly, but functions have been used to improve the code reuse. Common helper functions like `checksum_xor, pointer_outof_bound` is shared between sender and receiver. For storing arbitrary length data, a 12KB heap pre-allocated. Most of the variables like array pointer are stored in `.data` section. Stack pointer was used but in order to support one board running mode, and future parallel data sending, function re-entrance is later on added as a requirement. In addition, other functions also need to access that array pointer for checksum calculation. Therefore, a stack is no longer suitable to use.

3.2 Algorithms

Core algorithms to implement DF19 are to read bits and send bits by looping. Bit reading is fast, only takes two CPU cycles to do left shift, then does a "logic and" with 1 to get that bit. The data sending process is queue based (FIFO), first read, first send. The least significant bit is read and send first. When the receiver picks it, it will temporarily store it until all the data is transmitted. Then, it will assemble all those 16 bits into one register and store it as one value. By predefining the endianness of data, receiver and sender have an agreement so that it will not mess up the order. Space and time complicity are roughly equal to bit length. Besides, there are two flags in the memory that indicates the program state that writes by interrupts. Therefore, no time-consuming operations in interrupts and they finish quickly as well. Instead of doing modular arithmetic to bring array counter back when out of bound, the program simply reset it to 0. This is much better since the counter could go to maximum integer and get reset by CPU. In that case, the counter will point to an unexpected location.

3.3 Robustness and Extensibility

For implementing a network protocol, the most vital requirement is to recover from errors. The program has the ability to detect the error from the data line and then reset the transmission state to resend the data. The program is robust. It only plays music when all the data is received and confirmed. Playing sound while receiving notes might lead to wrong pitch or long delay when data is lost and resend is required. Meanwhile, the program keeps high extensibility, and it can transmit any kinds of 16 bits long data. It loops for 16 times when sending because only the least significant 16 bit is needed. The next generation of DF19 could extend to support 64 bits of data. All programmers need to do is to change the constant number accordingly.

Appendix

Demo of Transmit Error Recovery

https://youtu.be/jwwZ5KxIxIk

Notes

- In my program, the flushing red LED means data is sending. Green LED on with red LED off means data is sent successfully. If you see green and red LED flushing three times together, it means an error was been caught during current 16 bits transmission. Both sender and receiver will realize the error at the same time and automatically resend the data.

- The transition speed can be adjusted by changing the tim7 tick time, it was meant to be at least 0.0125 second per interrupt, which gives the speed $\frac{1}{0.0125 \times 2\ bytes\ per\ interrupt} = 40\ bytes/s$. Due to demonstration purpose, timer initialise code above is commented out, leave with a much slower timer that gives the speed 2 byte/s so that people can see the data sending process from flushing LED.