Internship Project Part II: Our project on VLC

Second section of my internship project about data transfer with visible light. During remaining afternoons of second week we brainstorm about ideas to build. Later we made research and prepare an 5 minute elevator pitch. My project is data transfer using RGB LEDs, which I will explain better later. At Friday of second week everyone presented their projects and later that day everyone voted for 26 projects to be build. My project is selected second out of six selected projects to be build. After projects selected we formed teams to work with. My team has five members; 4 electronics engineering major and one computer science major. We spend remaining time of that day for preparing work plan and buying hardware for our project. At following paragraphs, first I will give details about my project and later about work plan.

At the end of the two week our project was able to send text messages from one computer to another without an error using visible lights. How ever this system is really slow (40 bits/ sec) due to hardware and the technique we used.

Components we used for one transceiver at our project is follows as:

Hardware

- Arduino Uno
- Adafruit TCS34725 RGB color sensor
- Breadboard
- Three 1w LEDs (one red, one green, one blue)
- Lots of jumpers and plane cables
- Three 45º lenses for LEDs

Software

- Arduino IDE
- Text Editor (for writing C/C++ libraries)
- Git

Equipment

- Soldering equipment (We used the ones at AirTies)
- Tape
- Wire cutter

Schematic of our system can be seen at figure 1. Figure 1 includes a legend for cable mapping. Photograph of system we build can be seen at figure 2. Also lenses can be seen at figure to, outside of LEDs. Final versions of all codes used in our project can be found at Appendix A.

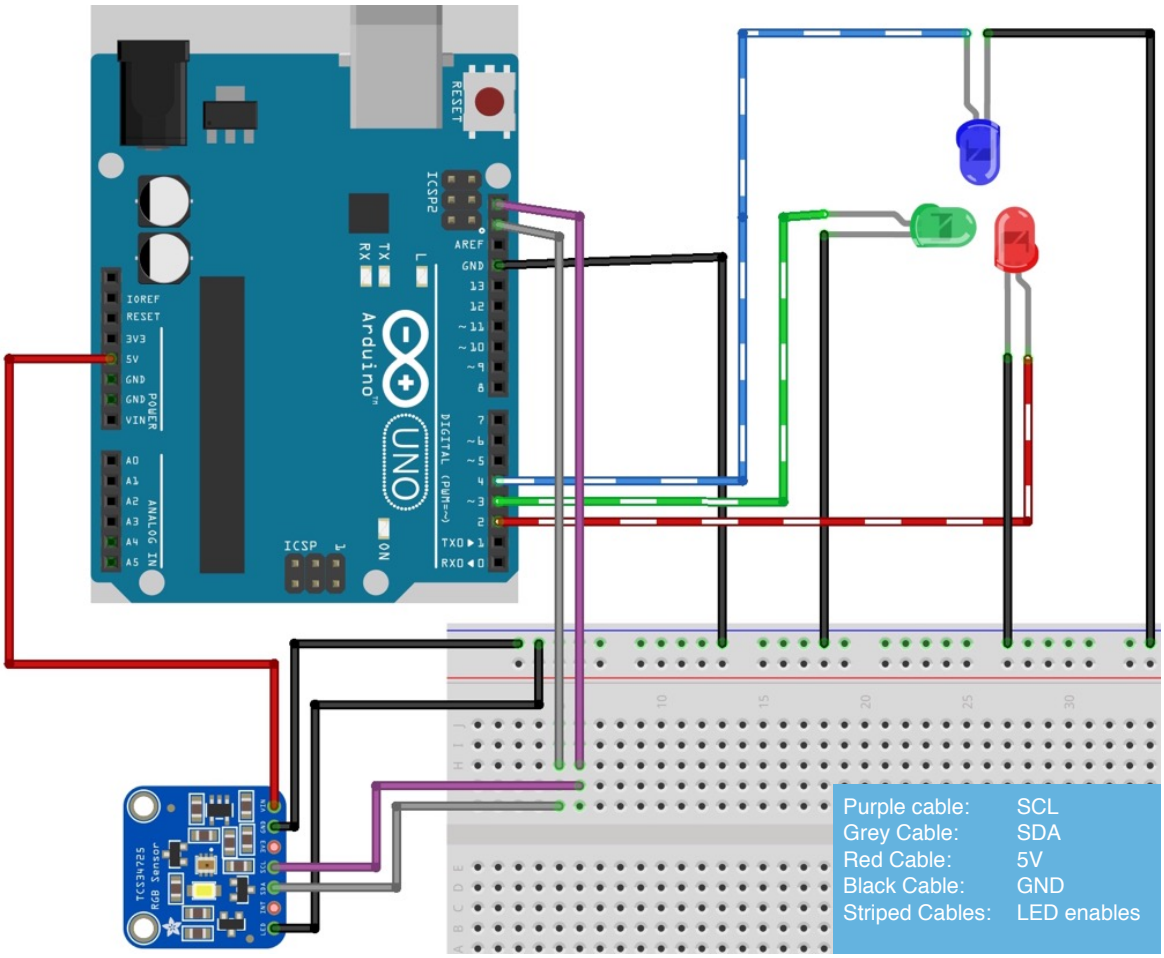

Figure 1: Schematic of System

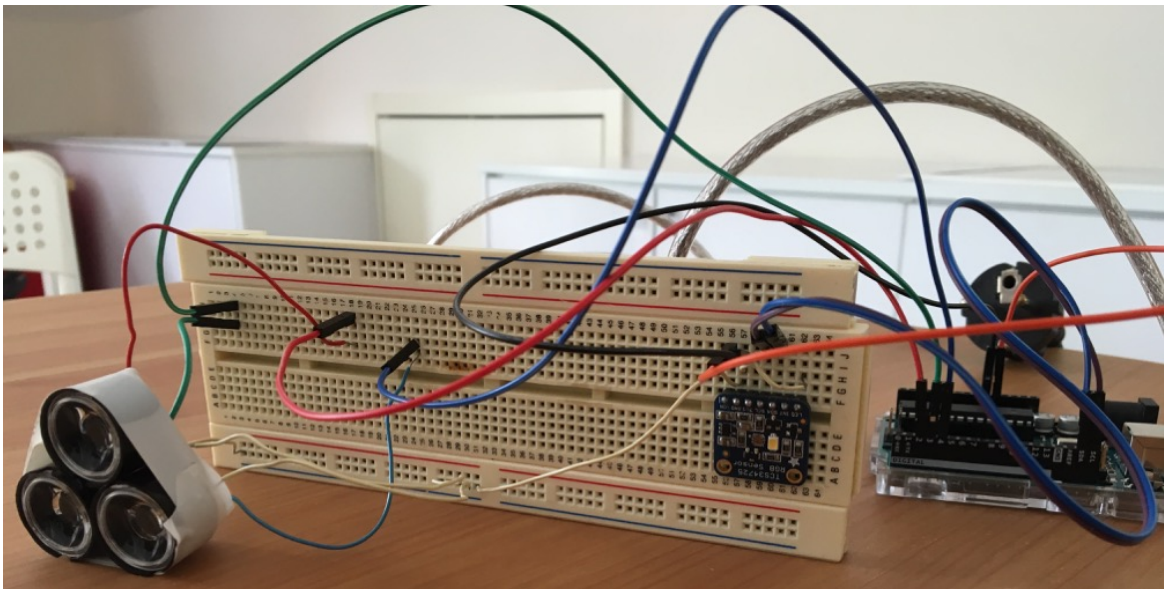

Figure 2: Physical System

Currently our system is in proof of a concept stage. It works on short range (~1.5m maximum tested), vulnerable for changes at the environment, slow (40bits/sec) and can only send ASCII text thought serial monitor at Arduino IDE.

Our system sends data char by char in 12 bit packages. We send data using three deferent channels (red, green and blue visible light) thus each char takes 4 cycle to send. Before discussing more detail we should observe the process. Assuming we have two people using this set Alice (A) and Bob (B). First Alice sent his message to Arduino A using serial monitor. Arduino A first encodes incoming char into binary data, then we apply a simple Hamming code (8-4) to create parity bits for error correction and create a 12 bit binary package. After creating package, Arduino A send this packages to Arduino B using the protocol we created. On receive Arduino B apply Hamming code for error correction and decode 8 data bits to a char and print it to serial monitor. This process works on all environments without an error as long as environment won’t change (increase or decrease at brightness). Only exception to previous statement is direct sun light. Our system won’t work on direct sunlight because sensor won’t work due to light intensity. We always observed

highest value from the sensor when we test it at outside under direct sunlight.

Our system has two protocols. One is for calibration, other one is for data transmission. Next two paragraphs I will explain our protocols.

Calibration protocol is simply gets environment values for each combinations for RGB values, by receiving them from other device. This process should be done for each device separately. This process starts when device enters calibration mode. At this mode device gets values for current environment and determines thresholds for that case. This process starts with the case all LEDs are LOW continues to the all LEDs are HIGH, when white light from our LEDs fall over sensor. Figure 3 shows the colour sequence with the

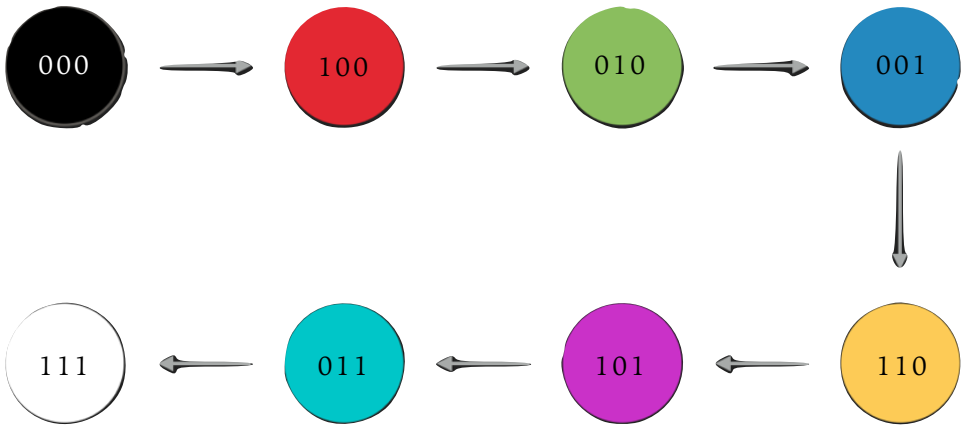

Figure 3: Synchronisation sequence

values. To determine threshold we develop a simple algorithm. Since the sequence is predetermined we know the order of colours. Device that is in calibration sequence (receiving device) first saves current environment values for all LEDs LOW then waits red LED to pass LOW threshold then constantly gets environment values until the HIGH LEDs are at maximum value (values for HIGH LEDs are stopped increasing). When we get highest values we multiply RGB values with coefficients to their states (HIGH or LOW) and save them. Then our device waits; first until HIGH LEDs of previous step pass down of LOW threshold, second until HIGH LEDs of next step pass up of LOW threshold. This second wait is same wait as first wait for red LED. LOW thresholds determined at first step of calibration sequence, when we get values for all LEDs LOW. Figure 4 shows the timeline of one step at calibration sequence. Up to this point I discussed only about receiving

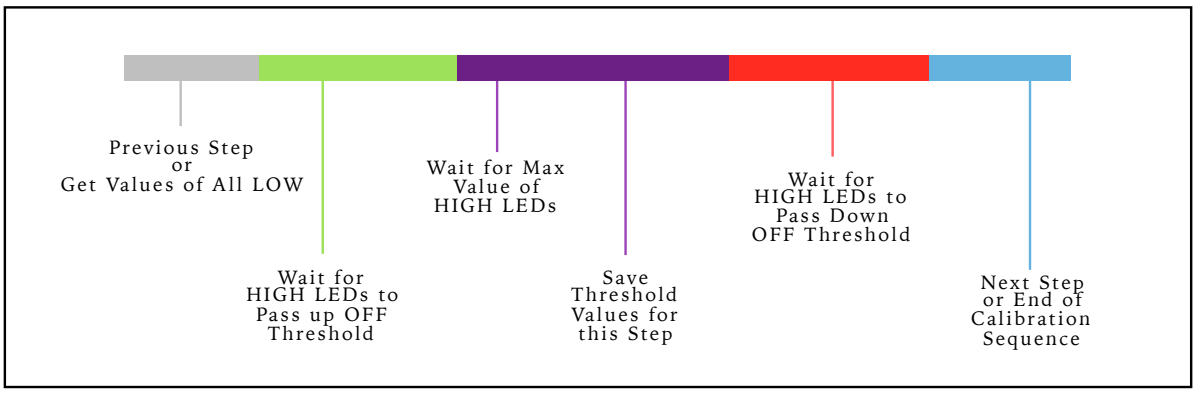

Figure 4: Calibration sequence timeline

side of Calibration protocol. That is because transmitting side is too much simple. We named it as Synchronisation

sequence. Synchronisation sequence follows the pattern at figure 3. At each step it turns on HIGH LEDs for that step, waits for initDelay (we set it to 30 ms but it can be changed easily), then turns off HIGH LEDs, waits for initDelay again and proceeds to next step. Timeline for synchronisation can be seen at figure 5.

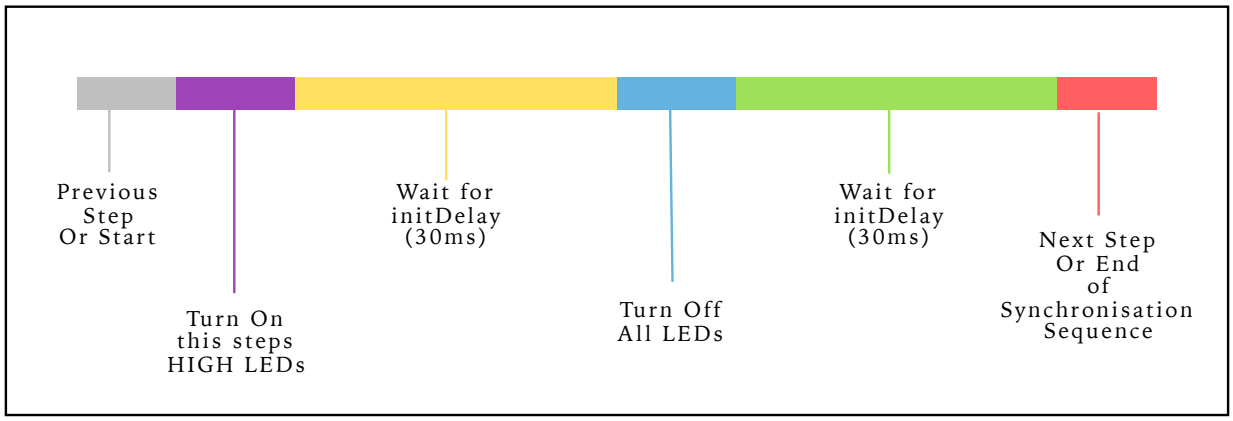

Figure 5: Synchronisation sequence timeline

Transmission protocol we used determines how we send messages. We send binary data directly, without any modulation. Using this protocol our devices will always work synchronously due to feedbacks provided. Transmission protocol consists two cycles, transmit cycle and receive cycle. Both cycles are can be seen at figure 6. I will start with transmit cycle, transmitting device lights green LED and waits receiving device's to respond by lighting green LED. When receiving device light green LED transmitting device turns off green LED and waits for receiving device to turn

its green LED off. Upon receiving device's green LED off, transmitting device sends first three bits of package by turning corresponding LEDs on, until blue LED of receiving device is lit. When that happens transmitting device turns all LEDs off and waits for blue LED of receiving device to turn off. Then send next three bits using same process until the last 3 bits. When last 3 bits send transmitting device waits red LED instead of blue LED. If transmitting device has another char to transmit it starts to wait for green LED of receiving device to first turn on then turn off. Transmitting device treat this green LED same as the first one, and continues to sending next package like previous one. Receiving cycle is the one which controls timeline. Timing at this cycle is controlled by two parameters fbTime and cycTime (feedback time and cycle complete time, both 25 ms). Feedback time states how long LEDs stay HIGH and cycle complete time states how long receiving device waits before doing processing. It starts upon receiving request (green LED) from transmitting cycle, and answers it with I am ready signal (green LED). Wait for cycTime then gets environment values, detects 3 bit and save them to a buffer. After receiving device finishes this step, it turns on and off blue LED (send me next 3 bit) with according to timing parameters. Receiving device continues this process until last 3 bits of package is

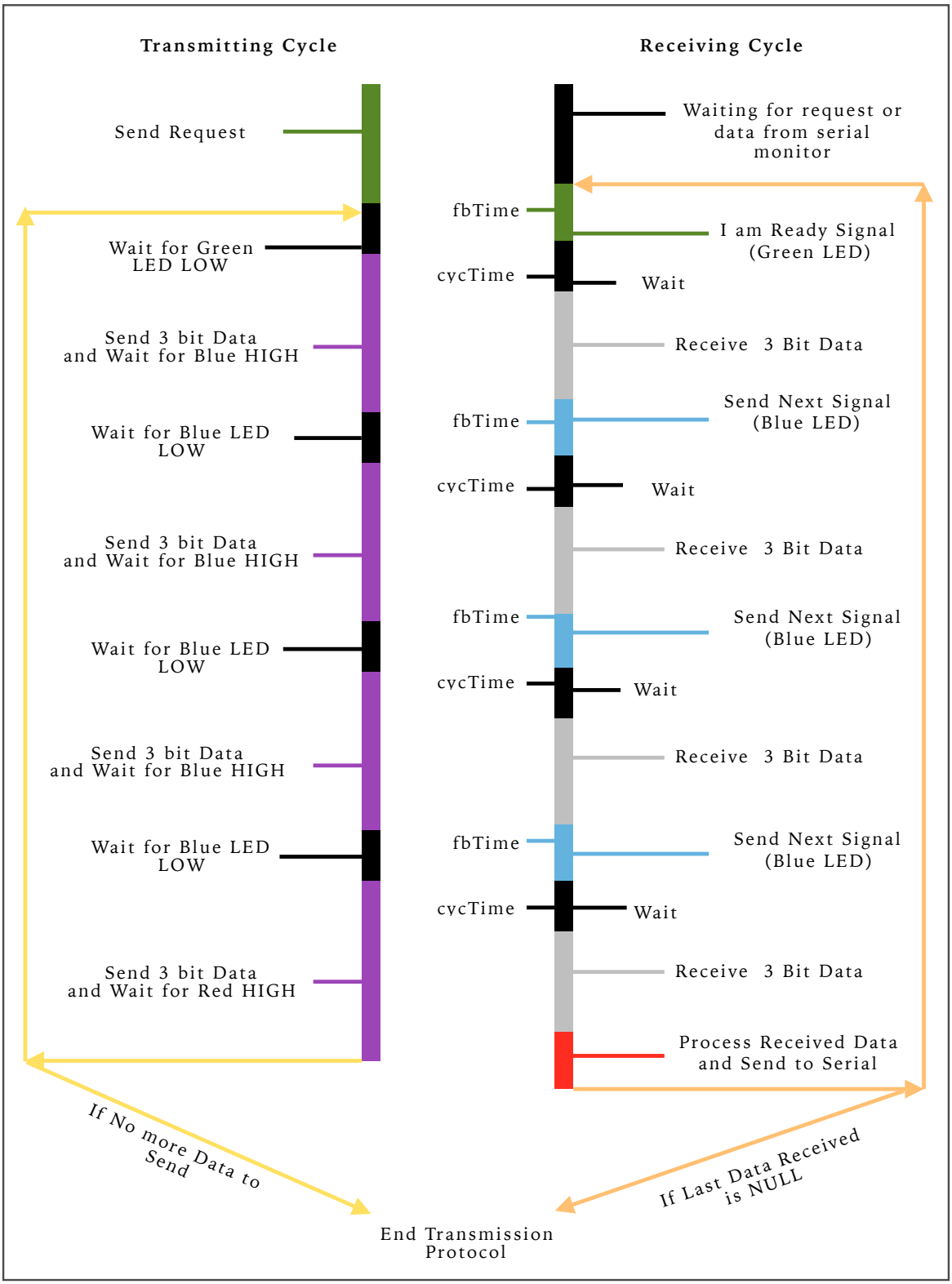

Figure 6: Transmission protocol timeline

received and saved to the buffer. At this point receiving device turns on red LED (I get the package) instead of blue one. Then receiving device does error correction and decoding. After sending received char to serial monitor, receiving device turns off red LED (Package delivered) turns on green LED (gives I am ready signal) and continues cycle from that point until it receives a NULL char. When NULL received transmission protocol for receiving device ends. Summarised ASM chart of device, that includes all operations, can be found at Appendix B.

We were a team with a five members thus we had more than enough time to build our project. First a two days we spend our time on research about error coding and visible light communication; as well as we prepare the hardware. Next three days we do the coding and testing. At the end of first week we had an early prototype that works. But this version had many errors. System was unstable, we observed many errors during transmission. Also we had a separate code for receiver and transmitter thus our system can send data on single direction. Next Monday we start with improving error rate. First we improved our RGB detection algorithm a little bit and set a calibration protocol similar to the one we currently use. These two improvements improved our error rate

a lot. And to stabilise our system we set the transmission protocol we currently use, which includes lots of feedbacks. Next day we combined transmitter and receiver codes into Master.ino file afterwards we did more testing. Wednesday morning we start with improving our calibration protocol to version we currently use and afternoon we continue our testing. Thursday we prepared presentation about our project, did more testing (We send long articles from internet) and looked for a way to send file. Friday we presented our project to others and listen other presentations.

Even though our prototype works with well (we were able to send an article with approx. 700 characters from approx. 1m distance without an error) it is not complete yet. We can add a driver and an application to send data instead of serial monitor. This addition will allows us to send non-unicode chars and files, as well as increase data transfer speed. Also we can design a custom hardware for this system, which will increase our data transfer rate by itself and also by allowing us to use modulation. Last improvement we planed to do in future is add an autonomous calibration process, which will protect our system from changes at environment.