**D.3.1) Communication Subsystem**

The communication subsystem is where the data transmission between the vehicle and the terminals take place. Based on our requirements analysis, it should reach to a data transmission rate of 13 kbps (kilobits per second). Furthermore, it should be able to work under different lighting conditions. Also, it should have a very low bit error rate since we are transmitting the bytes of a compressed image and even a single bit error can be problematic while decompressing the image.

Considering these issues, the communication subsystem is designed. It consists of two parts, the software part and the hardware part.

***Software Part of the Communication Subsystem***

In order to successfully transmit the bits, the software design should include,

* Protocol establishment and handshaking
* Error checking
* Modulation

These three parts are investigated seperately below under their own headings.

1. ***Protocol Establishment and Handshaking***

During the transmission, the receiver and the transmitter should agree upon structure of the transmission so that they can understand each other. For example, they should know when the transmission starts or ends and they should understand how to infer the incoming waveforms. For this purpose, in the conceptual design report, the following method was proposed:

where r(t) is the received signal, rand(t) is some random meaningless bit stream before the actual content of the message, synch(t) is the synchronization signal, m(t) is the data signal and end(t) is the ending indicator. This structure is illustrated in the graph below.



*Figure* ***X:*** *The arriving bit stream as the time passes*

In Figure **X,** structure of the message signal can be observed. As the time passes, the bit stream moves to the right and the receiver first bypasses the random signal and then realizes that the actual data is coming after fully detecting the synch bits. By this, we mean that after the receiver detects the synch bits, it starts a timer in its microcontroller and samples the incoming bits accordingly. Lastly, it understands that the transmission of one data packet ended whenever it detects the end bits. In this configuration, the length of the synch bits were 6 bits long and the end bits were also 6 bits long. The lenght of the message signal was 150 bits. In its essence, we proposed a serial asynchronous communication method.

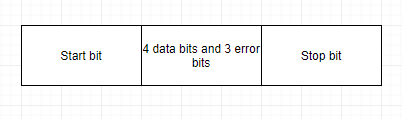
However, we have decided to modify this approach due to the fact that the timer inside our microcontroller is not counting the time passed correctly. As a result, after some time we were losing some bits or sampling the wrong bit at the wrong time. After realizing this, there were two options in front of us. We could either change the microcontroller or come up with a more suitable transmission scheme. We are using Arduino as a microcontroller we didn’t want to change it because it would create unnecessary cost. So, instead we have decied to modify our transmission scheme.

If investigated carefully, it can be understood that we were trying to develop a serial asynchronous communication method. We still didn’t give up on this idea in our modified solution, we have just changed its implementation. Because of these issues discussed above, we decided to switch to the UART protocol. UART stands for Universal Asynchronous Receiver-Transmitter. In this protocol, no clock signal is used to synchronize the receiver and transmitter. Instead, we control the flow of data by using start and stop bits. The data packet lies in between those bits and it is 7-bits long. If we refer to the **Equation K,** we can see that sync(t) becomes the start bit, end(t) becomes the stop bit and m(t) becomes the data bits. Therefore, by reducing the packet size and utilizing the UART chip inside the microcontrollers, we obtain stability and consistency during the transmission.

The data packet is used as 8-bits in standard so one can ask the reason we are using 7-bits instead of eight bits. The reason for that is to utilize error correction. Normally, without any error correction, to transmit 1 byte of data, we transmit one packet in UART. However, once we add error correction, we transmit two packets in order to transmit 1 byte of data. That is because in one packet, we insert four bits of data and three bits of error correction bits. Because of this, in order to transmit one byte, we send two bytes. This is a trade-off which we are willing to make to ensure the secure and correct transmission of data. The mechanism of error correction will be discussed in its heading in detail.

In UART communications, the receiver knows that the bit stream is coming when it detects the start bit and knows that one packet ended when it detects the stop bit. However, in this case, although the receiver knows when to detect data, the transmitter doesn’t know when to start sending it. This is where the handshaking comes in. When the car reaches to a terminal and is ready to take in data, it sends a special signal via UART and then the terminal starts sending data.

Lastly, the structure of one packet of data can be seen in **Figure Y.**

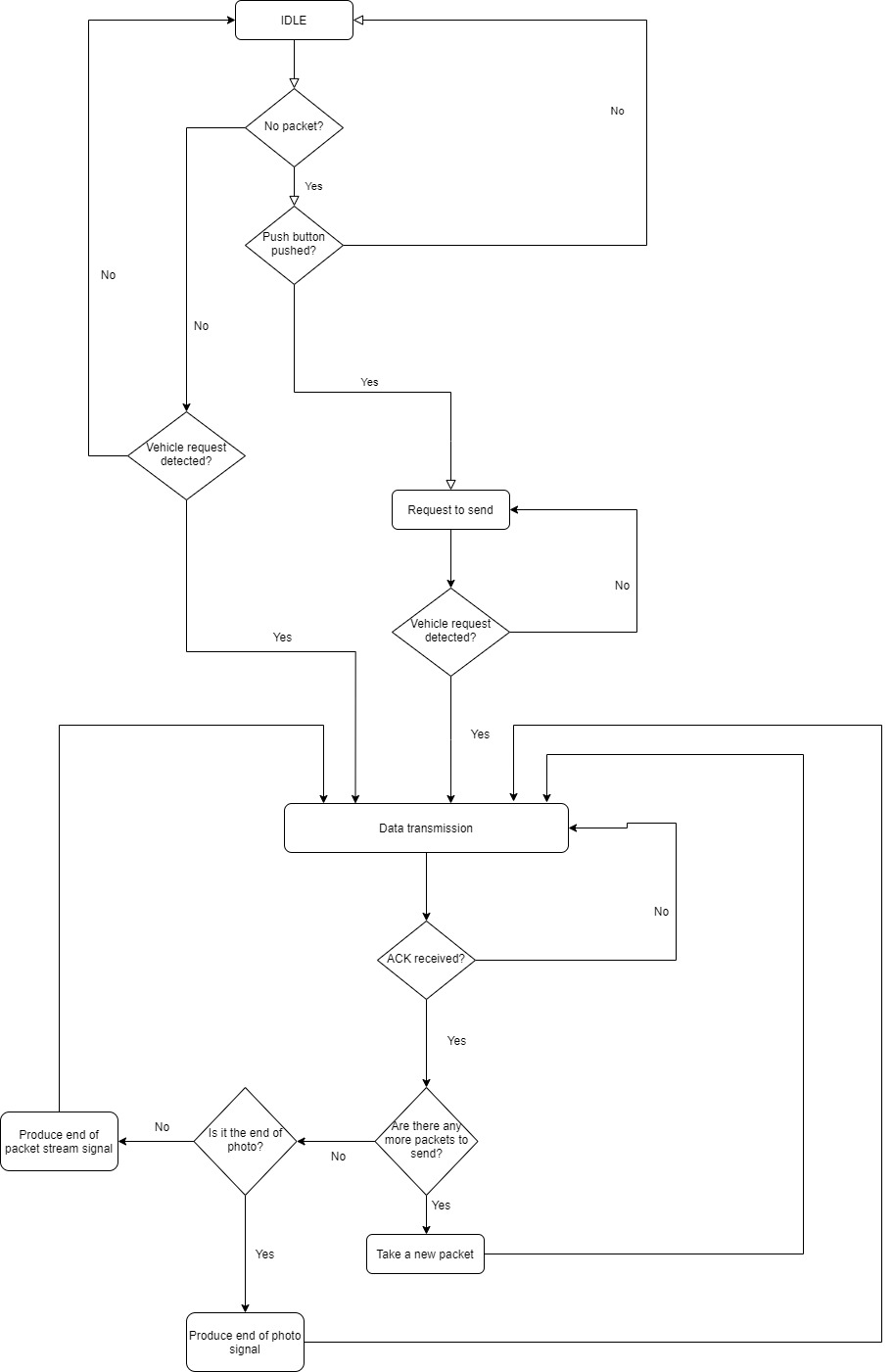


*Figure* ***Y:*** *Structure of one data packet in UART*

One final note to make here is that while the transmitter and the receiver are communicating with each other, they need some special signals to indicate that a special event is triggered. These signals are end of the photo signal, end of the packet stream signal and end of a packet chunk signal.

End of the photo signal tells that there will be no more data anymore so that the image reconstruction can begin. End of the packet stream signal informs the car that it is fully loaded and good to go. End of the 1 kB signal gives the receiver a chance to detect if there is an error in the packet stream that it received. If there is not an error, then it acknowledges that to the transmitter so that it can send a new packet. If there is no acknowledgement, then the transmitter concludes that there is something wrong and resends the data.

In order to perform handshaking, the transmitter first sends a request to send signal to the receiver. The receiver replies back by sending a vehicle request. Once this occurs, the transmission begins. After the transmission finishes, the transmitter sends one of the special signals indicated above to trigger any special event. The detailed flowchart of the transmitter side can be seen in Figure **AQ.** Also, the detailed flowchart of the receiver can be seen in Figure **QQ.**

*Figure* ***AQ:*** *The flowchart of the transmitter*

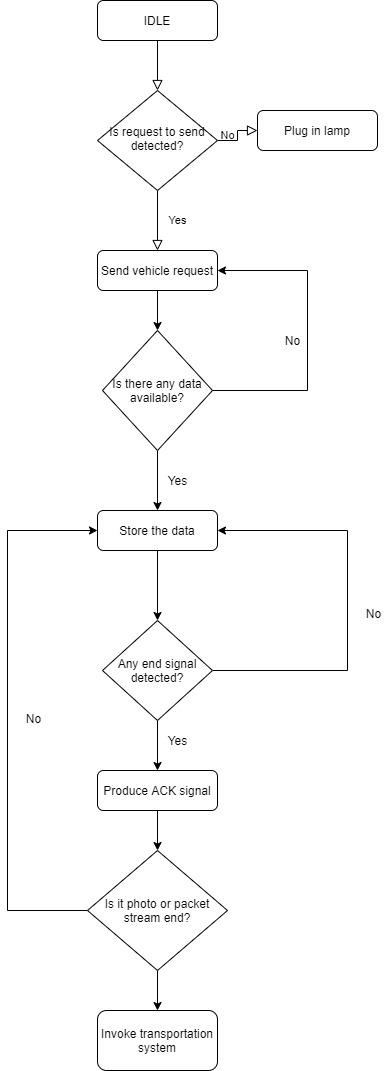
****

Figure **QQ:** The flowchart of the receiver

1. ***Error Checking***

In order to perform error correction Hamming coding is used. Hamming code is a mechanism which allow us to correct single bit errors in a given bit stream. It achieves that by adding some redundant bits to the actual data bits. In order to find the number of redundant data bits, the following formula is used.

where m is the number of data bits and r is the number of redundant bits. In our case, the number of data bits is equal to 4. Therefore, if we plug m = 4 into the **Equation L,** we find out that the minimum r value is equal to 3. Therefore, we need 3 extra bits to protect 4 bits. The extra bits are placed to the locations which correspond to the powers of two. It is illustrated in **Figure Z** below.



*Figure* ***Z:*** *The placement of bits in one data packet*

In Figure **Z,** each capital letter represents one bit. The bits A, B and D are the redundant bits and they are placed in the first, second and the fourth places. This placement is consistent with the idea of inserting them to the places which correspond to the powers of two. The rest of the letters C, E, F and G are data bits.

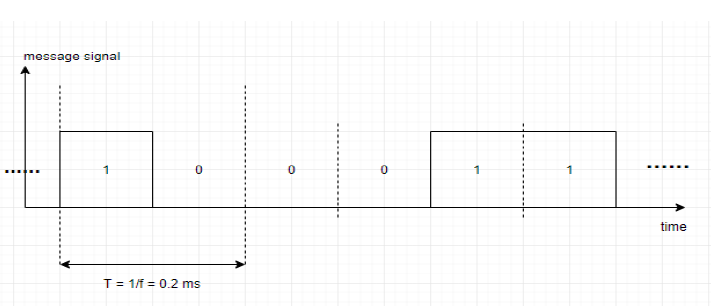
The values of A, B and D are found by using even parity. In our case, A covers the bits C, E and G. Therefore, if the total number of 1’s in C, E and G are odd, then A is 1, if not, then A is zero. B covers the bits C, F and G. D covers the bits E, F and G. Even parity is used for B and D, too.

Whenever, the receiver obtains a packet like in **Figure Z,** it performs the logic operation XOR on them in a predefined manner. The exact operations that it performs are as follows.

After performing these logic operations, it concenates the results as RQP and it treats it as a binary number. If that number is something other than zero, for example 3, then it finds the bit at the third location and toggles it.

1. **Modulation**

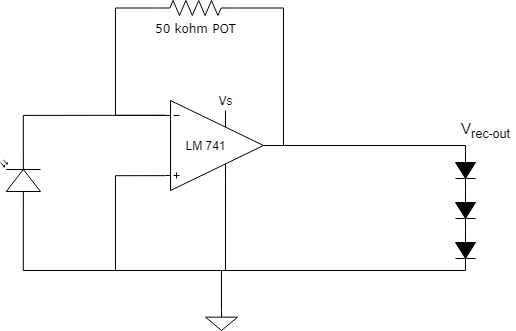
In order to represent the logic 1 and 0, we proposed to use on-off keying. In this modulation method, we assign LED state as ON for binary 1 and OFF for binary zero. The receiver detects these on and off states and interprets them as 1 or 0. For this purpose, we need the UART ports of the Raspberry Pi and Arduino to produce 5V for logic 1 and 0V for logic 0. They achieve this function by the use of serial.write() command in their libraries. In the hardware section, the circuitry for the LED driver will be discussed.

*Figure* ***T:*** *An example message waveform with OOK*

In Figure **T** it is possible to see an example message waveform. In this case, the frequency of the waveform is 5 kHz which corresponds to 10 kbps. By utilizing a better circuitry for the receiver terminal, we managed to increase its speed to 30 kbps. Considering the fact that our speed requirement is 13 kbps, we have met that requirement quite well.

***Hardware Part of the Communication Subsystem***

The previous receiver circuitry was constructed with a transresistance amplifier with LM 741 operational amplifier. The output of LM 741 was coming with 2 V DC bias due to fact that negative bias of the LM 741 opamp was connected to ground. The company planned to use one power source which was used as Vs as seen in Figure **O**. The serial connected diodes were used to provide voltage drop ≈ 2V so that voltage output of receiver circuit can have high and low level voltages which can be read by the RX pins of the Arduino or Raspberry Pi for serial communication purposes.



*Figure* ***O****: Previous receiver circuit*.

The circuit was operating in 5-10 cm range and output was very sensitive to external intense light sources. These problems are solved in the improved version of the receiver circuit which is seen in Figure **OBB**.

A high pass filter is connected next to transresistance amplifier. It filtrates the DC part of the output of the transresistance amplifier and an AC signal which oscillates around zero is obtained. This signal is then inputted to a comparator where the reference voltage is 0V.

The resultant circuitry has much better distance and external light insensitivity. Additionaly, it operates at 6V supply which results in less power consumption. In our previous design, we used to operate at 9V level.

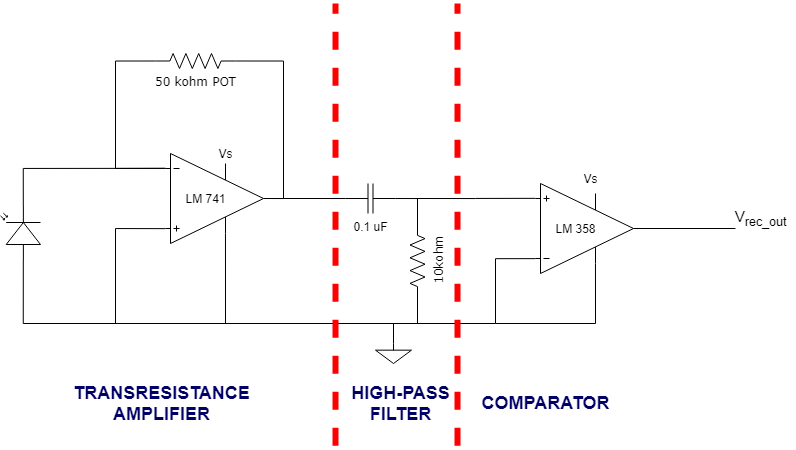
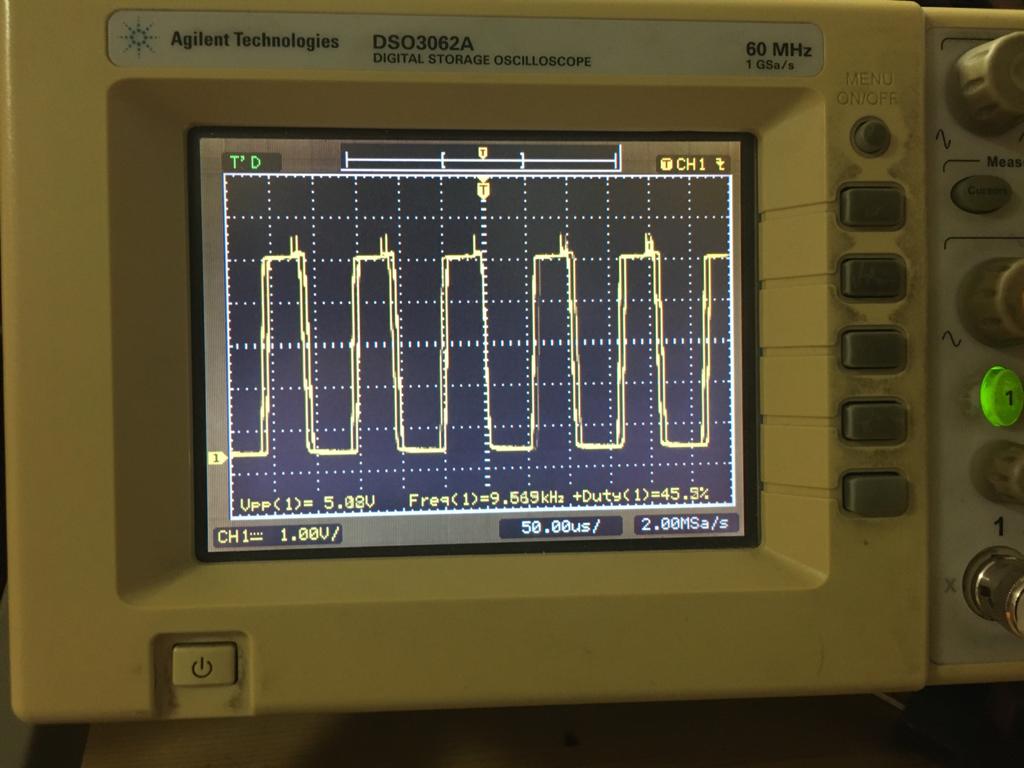


Figure **OBB**: Improved receiver circuit.

An example output of the receiver circuit where recever and transmitter distance is 30 cm, input is 10 kHz square and the photodiode is under intense light is seen in Figure **KQ**.



*Figure* ***KQ****: An example output of receiver circuit*.

Another part to discuss is the transmitter circuitry. The transmitter circuitry basically consists of an LED driver. The circuit can be seen in Figure **OZBOY**

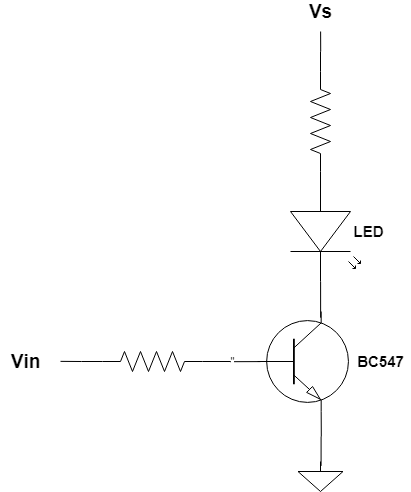
****

Figure **OZBOY:** The transmitter circuit.

The Vin in Figure **OZBOY** is connected to the TX pin of the Raspberry Pi. Whenever the TX pin is HIGH, the LED is ON and whenever it is LOW, the LED is OFF. This way, we can physically realize the OOK modulation.