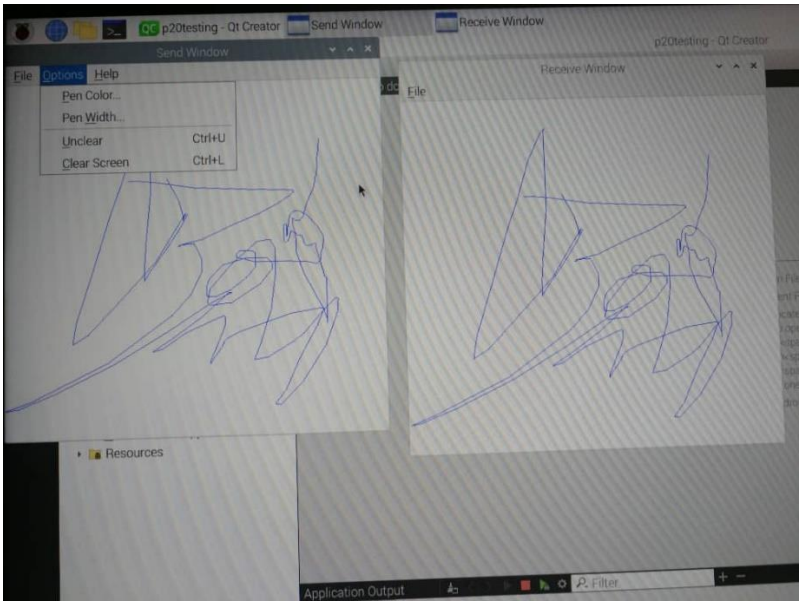
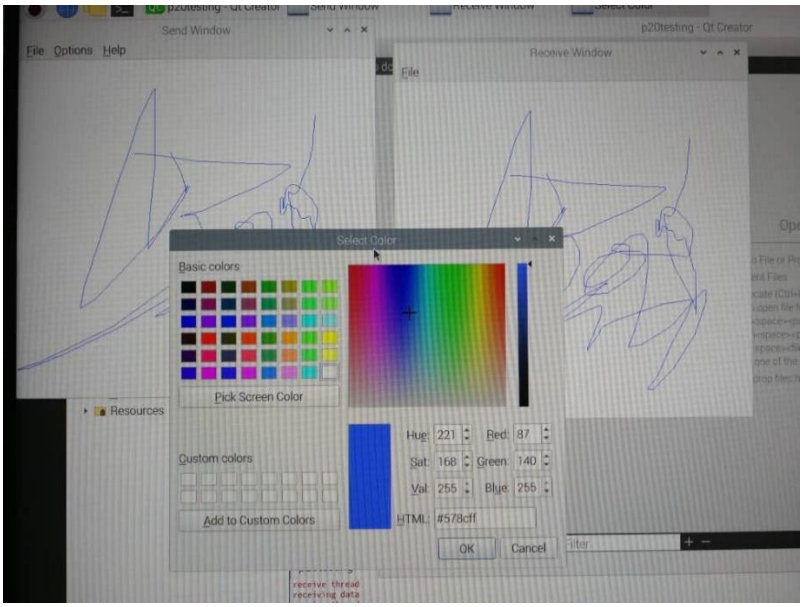
Lab P20 Logbook

By Vernon Koo Hern Xhen 31541933

Section 3.1 The GUI send and receive windows

- Two windows are created. One is the sender window while the other one is the receiver window.
- The application is designed to have several functions such as pen color, pen width, clear screen and unclear screen. For example, users can select the desired pen color from a color box as shown in the diagram below.
- QImage class is used to provide a hardware-independent image representation that allows direct access to the pixel data, and can be used as a paint device.
- There is a receive window that is called when the user sends the drawing to perform specific paint event. The painting operations are implemented by using QPainter class on the paint device.





Section 3.2 Serialize and deserialize drawing-commands

- As the gpio are only able to read and write binary data (0 and 1), serialisation of data into array of 0 and 1 is needed.

- The first three bits of my dataframe represent the opcode which are the operation and command to be performed such as startDrawCommand, drawLineCommand, penWidthCommand, penColorCommand, clearCommand and unclearCommand.

- From the third bit onwards, the bits represent the information to be transmitted such as the xy coordinates of the drawing position, pen colour and pen width.

- The last bit is the parity bit. Even parity is implemented in this case. This is used for error detection when passing the data from sender to receiver. The complete dataframe is as shown in the diagram below.

Opcode	Information bit	Parity bit

- Deserialization of data is implemented in the receiver side to deserialise the data received from the sender side. The receiver side will perform the necessary deserialization and operation based on the first three bits of the dataframe which is the opcode. An example of the code implementation is as shown below.

```
QString bit1=binarr[0]==1?"1":"0"; //LSB
QString bit2=binarr[1]==1?"1":"0";
QString bit3=binarr[2]==1?"1":"0";

QString decodedCommand=bit3+bit2+bit1;

if(decodedCommand=="001"){
        //qDebug()<<"entering draw line command";
        int j=0;
        for(int i=3;i<13;i++){
            pt_x+=binarr[i]*pow(2,j);
           // qDebug()<<"pt_x "<<pt_x;
            j++;
        }

        j=0;
        for(int i=13;i<23;i++){
            pt_y+=binarr[i]*pow(2,j);
           // qDebug()<<"pt_y "<<pt_y;
            j++;
        }
         QPoint endPoint = QPoint(pt_x,pt_y);
         drawLineTo(endPoint);
        // qDebug()<<pt_x;
        // qDebug()<<pt_y;
    }


    if(decodedCommand=="010"){
       // qDebug()<<"Entering pen width command";
        int newPenWidth=0;
        int j=0;
```

```
    for(int i=3;i<9;i++){
        newPenWidth+=binarr[i]*pow(2,j);
       // qDebug()<<"pen width is "<<newPenWidth;
        j++;
    }
    setPenWidth(newPenWidth);
}
```

Section 3.3 Implement send- and receive- threads

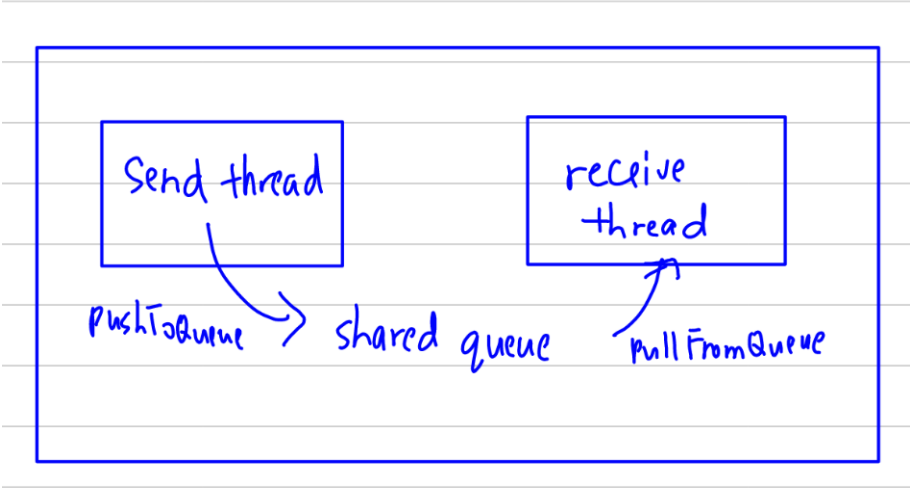
- In this section, multithreading technique is implemented. Multithreading allows the execution of multiple parts of a program at the same time by making good use of the available computer resource.

- Send and receive threads are implemented. The role of the send thread is to take serialised data and pass the serialised data into the queue/output pin, while the role of the receive thread is to read data bits from queue/input pin.

- There is a shared queue which can be accessed by both send and receive threads.

- The send thread will push the data into the shared queue via the pushToQueue() function.

- The receive thread will get the data from the shared queue via the pullFromQueue() function.



- Mutex is used to prevent race-around condition. If some thread reads data while another one is writing it might receive inconsistent data. So all reads must have finished before writing is allowed and new reading must wait until writing has finished. An example of the usage of mutex is shown below

```
mutex->lock();

serialisedQueue.enqueue(pushedData);
*dataReady=1; //data is pushed to queue and is ready to be read

mutex->unlock();
```

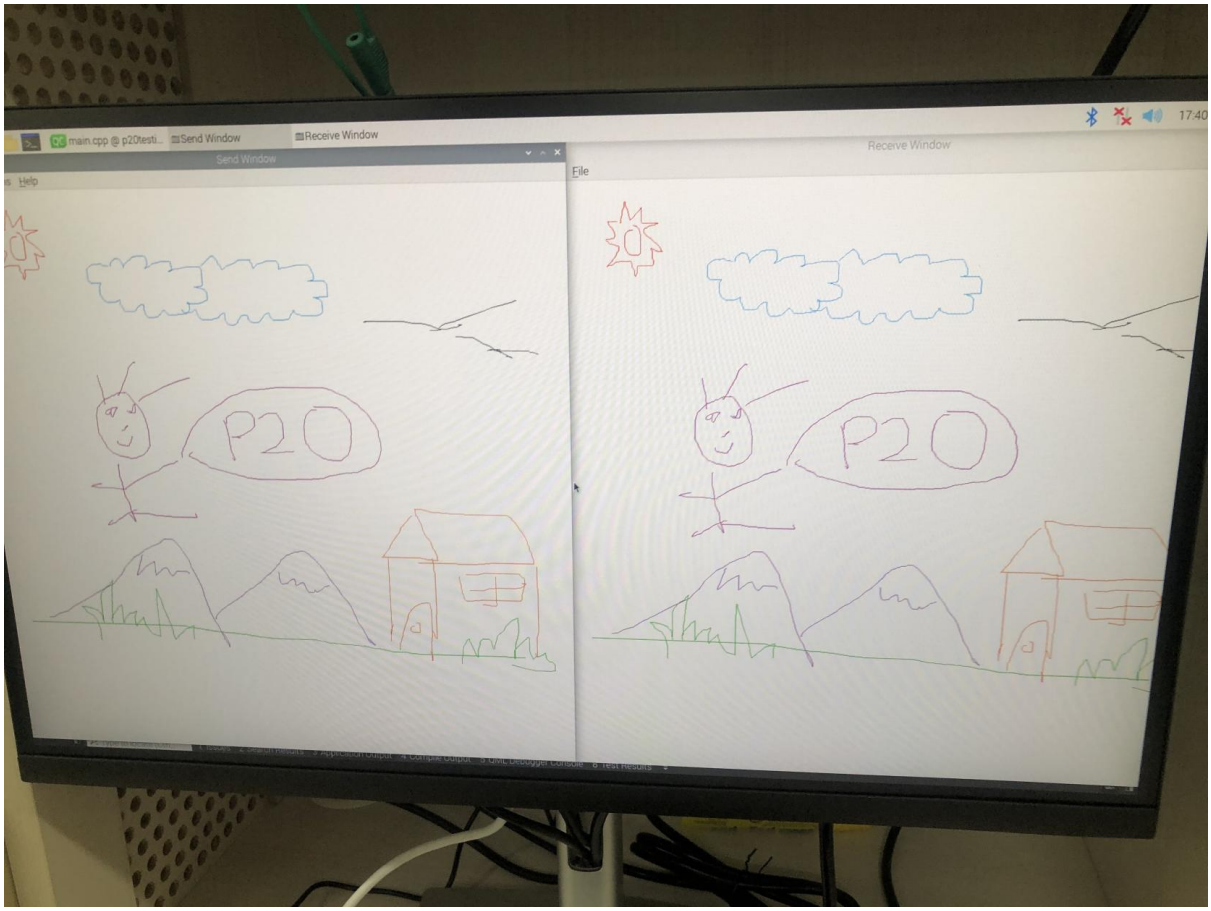
3.4 Implement your communication protocol using Booleans

- In this section, my bit-stream communication protocol is tested by toggling shared Boolean variables. The shared Boolean variables defined in this case are dataReady and readData. dataReady is set to one when a data bit is pushed into the shared queue and is ready to be read by the receiver. The readData variable will be set to one when the receiver is busy reading data and will be toogled back to zero after the receiver has finished reading data and is now ready to read a new data bit.

```
mutex2->lock();
*dataReady=1; //data is ready to be read
*readData=1; //receiver is busy reading data
mutex2->unlock();
```

3.5 Read and write to physical GPIO pins

- In this section, the communication is implemented using physical GPIO pins.
- Wiringpi library is included to use the GPIO pins available on the raspberry pi.
- Built-in function, digitalWrite(pinNum,value) is used to write to the pins, while digitalRead(pinNum) is used to read the pins value.
- To transmit and receive data using GPIO pins, we have decided to use synchronous communication with the help of clock. In this case, the data bits are sent by the sender during the first positive edge triggering while the receiver will read the data bits during the first negative edge triggering. By using synchronous communication, the problem of data out-of-sync can be prevented.
- The implementation of my code is tested using loopback mode (connect the send pins to the receive pins on the same Pi) and it is shown to be working properly.
- The result is shown in the figure below.



4 Optional Additional Work

To make our communication more robust, we used queue to store the transmit bit first before sending through the gpio. The advantage of doing this is that, even when the clock signal is disconnected while the drawing is still continue on the sender side, the receiver side will back in sync after the clock signal is being reconnected.

Besides that, even parity is being implemented for error detection at the receiver side. The sender will set the parity to zero if there is an even number of one bits in the dataframe. If the number of one bits adds up to an odd number, the parity bit is set to one. At the receiver side, the receiver will again calculate the number of one bits in the dataframe using the same method as the sender side. If the sender and receiver parity bits are the same, then there is no error in data transmission. However, this method may fail to detect errors introduced when an even number of bits in the dataframe is being altered.

Code reference :

https://www.newthinktank.com/2018/07/qt-tutorial-5-paint-app/