Q: How will the send-window allow users to draw diagrams? How will it display diagrams as they are being drawn, and how will it retain these diagrams so that they don’t disappear when the window is repainted?

A: There will be a QImage at the send-window which will allow the user to draw on top of the QImage. I will use QPainter and get the coordinates of the user mouse to indicate where the user wants to draw, by using .drawline command the program will be able to paint onto the QImage/diagram. There will be a canvas variable which will have QImage type, and the user drawing will be ‘save’ to the variable and repainted using paintevent.

Q: How will you represent the drawing commands so that they can be sent to the other user whilst they are being drawn?

A: The position of the user input when drawing will be stored as QPoint, by using signal and slot, whenever the user draws, it will emit a signal with QPoint to the slot. That slot will then receive the QPoint and send it to the receiver window and the receiver window will take that QPoint and put it into another function that will take the QPoint and call draw function to draw on the receiver window.

Q: How will you serialize these commands into packets to be sent from the send window?

A: The command, can be QPoint, int, etc, will be converted into bytes and stored as QByteArray. With the help of QDataStream, different data types can be converted into QByteArray. It will be converted from QByteArray to QBitArray before sending out the packet from send window.

Q: How will you use threads to send and receive these packets, while the rest of the application keeps running? How will you use mutexes to make any relevant collections “thread-safe”?

A: There will be 2 threads, one transmitter thread and the other receiver thread. Both threads will be kept running. Each thread will wait until a certain pin state is met, for example, the receiver will wait until the transmitter sends a sent state high indicating it has finish sending and the receiver can read now. Mutex lock can be used when the transmitter is writing data bits, the would prevent receiver from reading it when it is not ready to be read, the transmitter will unlock the mutex once it has finish writing and the receiver can now read the data bits.

Q: How will you convert binary packets into a stream of 1’s and 0’s? How will you transmit this stream in a reliable way? For example, you may need to signal when a bit is ready to be read, and when the receive window has finished reading the current bit.

A: There will be a for loop to read each bits in QBitArray and write it to a global bool array. There will be a 3 way handshake before the transmitter transmit data to the global bool array, once the transmitter has successfully transmit the data to global bool array, it will set a sentState bool to becomes 1. The receiver upon getting sentState = 1, will start reading from global bool array. Once the receiver has finish reading it will send a receiveState = 1, and all state will be reset to 0 before the other round of transmit and receive starts.

Q: How will you receive and buffer packets at the other end? How will you deserialize them? How will you draw them on the receive window? How will you retain the currently received diagram so that when the window is repainted the diagram isn’t lost?

A: The receiver will read a global bool array and convert them into QBitArray, then to QByteArray. This QByteArray will be sent to the respective command at the receiver window drawing class, and inside each respective function, the data will be deserialise using QDataStream into different data types. These data types will then be passed on to draw function and be drawn on receive window. Since it is being drawn onto QImage and it is stored in as a variable, the diagram will not be lost once the window is repainted as the diagram is stored as a QImage varble, and every time the window is repainted, it will paint the QImage variable.

**Communication protocol**

Program Start

Open sender & receiver window

Sender window:

* + Serialise corresponding data from user input (eg: drawing, changing pen colour, etc) into array of bytes
  + Emit a signal (data & char)

Transmitter:

Function A

* + Stores data into thread safe queue
  + Call function B

Function B

* + Mutex lock

// Initiating 3-way handshake

* + Transmitter ready = 1
  + Mutex unlock
  + Wait for receiver ack = 1
  + Mutex lock
  + Transmitter ack = 1
  + Mutex unlock
  + Dequeue both char and data
  + Serialise char
  + Wait for receiver receive = 0
  + Mutex lock
  + Change both char and data to array of bits
  + Write each bit of char to global char array
  + Write each bit of data to global data array
  + Set sent = 1
  + Set transmitter ready and transmitter ack = 0
  + Mutex unlock
  + Wait for receiver receive = 1
  + Mutex lock
  + Set sent = 0
  + Mutex unlock
  + Wait for receiver receive = 0
  + If still have element in queue
    - Run function B again

Receiver:

Function A

* Wait for transmitter ready = 1
* Mutex lock
* Receiver ack = 1
* Mutex unlock
* Wait for transmitter ack = 1
* Wait for sent = 1
* Mutex lock
* Read each bit of char from global char array
* Change char from bits to bytes
* Deserialise char
* Given char alphabet, resize bit array for reading data
* Reach each bit of data from global data array
* Change data from bits to bytes
* Given char alphabet, emit different signal to receive window along with data
* Set receive = 1
* Set receiver ack = 0
* Mutex unlock
* Wait for sent = 0
* Mutex lock
* Receive = 0
* Mutex unlock
* Call function A again to receive another packet of data

This particular communication system require data to be written from transmitter class to global variables at main.cpp then pass on to receiver class. Global variables will act as a transmission medium and carry information in terms of Boolean, mimicking GPIO.

Global variables :

sendState -> Indicate the state of transmitter, 1 -> transmitter has sent data out, 0 -> transmitter has not sent data out.

receiveState -> Indicate the state of receiver, 1 -> receiver has receive the data, 0 -> receiver has not receive data

3-way handshake variable:

senderReady -> 1 -> transmitter is ready

receiverAck -> 1 -> receiver has received sender ready signal and has ack it

senderAck -> 1 -> transmitter received receiver ack signal

charStream (array) -> stores the char sent out by transmitter

dataStream (array) -> stores the data sent out by transmitter

Char is used to indicate different command that the user has selected, example: ‘p’ when user is drawing, ‘C’ when user clears the screen. It is being passed to receiver to allow the receiver to interpret what the following data is meant for.

When the user do something, the serialise data along with a char will be sent to the transmitter safe thread queue. Inside safe thread queue, the data and char will be stored inside their respective queue and the function that transmit data out will be called.

Before sending data to dataStream, the transmitter will initiate a 3-way handshake with the receiver using global 3-way handshake variables. Upon successful handshake, the transmitter will ensure that the receiver has finish reading the last data from dataStream, by making sure receiveState = 0. The transmitter will serialise char data from send window and change it from array of bytes into array of bits. The transmitter will first write the char bits onto global array charStream. After that, the transmitter will write each data bits onto global dataStream. After finish writing, the transmitter will make global variable sendState = 1.

The receiver will now receive that sendState = 1, and start reading each bits of charStream, storing it to a local char bit array. It will then change the bit array into bytes and deserialise it. The receiver will now know what the char alphabet is and resize the local data bit array accordingly. The receiver will now reach each bits of dataStream and store it to a local data bit array and convert it to bytes. Depending on char alphabet, the receiver will emit a signal and pass along the data array to receive window. The receive window will then call the corresponding function and make changes to the receive window. The receiver will now set receiveState = 1. It will wait until sentState = 0, before setting receiveState back to 0. The receiver will emit another signal that will call the receiver to read data again, starting from waiting for 3-way handshake.

The receiver will be independent to the transmitter, it will always run again when it is completed, to get ready and receive the next packet of data from transmitter.