Creacion de un protocolo de comunicación

1st Díaz Rodríguez Andrés Felipe
Ingeniería Electrónica
Universidad Sergio Arboleda
Bogotá, Colombia
andres.diazrodriguez@correo.usa.edu.co

2nd Novoa Neisa Lorena
Ingeniería Electrónica
Universidad Sergio Arboleda
Bogotá, colombia
lorena.novoa2@correo.usa.edu.co

3th Arcila Quevedo Carlos Andrés
Ingeniería Electrónica
Universidad Sergio Arboleda
Bogotá, colombia
carlos.arcila@correo.usa.edu.co

***Resumen*—En el siguiente informe se va a mostrar el diseño de un protocolo de comunicaión bidireccional entre dos aplicaciones de los cuales, ambos deben poder enviar y recibir archivos de hasta 10MB, adisional, se debe agregar un procedimiento de DEPLOY.**

***Abstract*—The following report will show the design of a bidirectional communication protocol between two aplications, both of which must be able to send and receive files of up to 10MB, additionally, a DEPLOY procedure must be added.**

***Keywords*—protocolo de comunicacíon, Deploy, ComboBox, puerto serial.**

I. Introducción

Para el proyecto se tomo como base de objetivos la guia de laboratorio presentada por el profesor [1]. Los objetivos definidos por esta eran diseñar una comunicación serial entre dos aplicaciones las cuanles deben enviar, recibir y guardar archivos de hasta 10MB. tambien debe mostrar una grafica y un texto donde muestre la implementación en tiempo real de la transferencia de bytes en ambas direcciones y la tasa de velocidad de transferencia bytes/s. Por último se debe debe hacer un procedimiento de Deploy.

Como base se de pruevas, se van a tener dos aplicaciones, ambas con la misma capacidad de enviar, recibir y guardar archivos. internamente, la comunicacion funcianará a traves de un puerto serial el cual permite transmitir bits que seran los encargados de cargar la informacion. Este puerto funciona a travez del hardwaew UART (Universal Asynchronous Reciver Transmitter). [2]

Para diseñar la comunicación entre Apps, se se tomo como instipración el protocolo de comunicación serial I2C, ya que este método ya que este método es similar al requerido por la guia.

El protocolo I2C puede enviar mensajes divididos en tramas de datos, la cual con lleva una dirección a la que debe llevar y son direccionados por un bite de start, unode stop y algunos de comprobación(en la imagen 3 se puede observar la composicion de un mensale en I2C) [3].

Para la realización de la interfaz, se hizo uso de las siguientes librerias:

- **QDebug**: esta biblioteca permite que se haga impresion de lo que se necesita ver en panantalla, para usar esta opción, solamente se debe agregar # include <QDebug> en main.cpp y adicional, si se quiere imprimir algo una grase, simplemente se escribe la linea de código ”qDebug()<< ”lo que quiera ejecutar”;” [4]
- **QMainWindow**: esta Herramienta de Qt es la encargada de proveer una ventana de aplicación y tiene muchos usos, uno de ellos, es para hacer menus, toolbars, entre otras propiedades; para llamar esta biblioteca, se hace de la siguiente manera: # include <QMainWindow> en el .cpp y segun lo que se vaya a usar se llama la propiedad de esta que se desee. [5]
- **QSerialPort**: esta librería provee todo lo necesario para acceder a los puertos seriales, se llama de la siguiente manera # include <QSerialPort> pero antes, en el archivo .pro del proyecto al lado de ”Qt += core gui”s debe escribir serial port, para realmente acceder a estos puertos, y desde el archivo.cpp se puede acceder a cualquiera de las funciones que tiene esta librería, que entre ellas esta la función RequestToSend() La cual retorna falso o verdadero cuando las banderas enstan en high o low. [6]
- **QSerialPortInfo**: esta librería sirve para encontrar los puertos seriales que se van a usar, su cabecera es la siguiente # include <QSerialPortInfo> y su constructor es así:
  QSrialPort(). [7]
- **QMessageBox**: Esta librería provee lo necesario para realizarle una pregunta al usuario, por ejemplo si desea guardar o abrir un archivo; Oara hacer uso de esta biblioteca se llama en ek header así: # include <QMessageBox>; y para hacer uso dentro del código, se usa de la siguiente manera: en el botón que se quiera usar se pone la siguiente línea de código:
  QMessageBox::information(”parent”,çonst Qstring

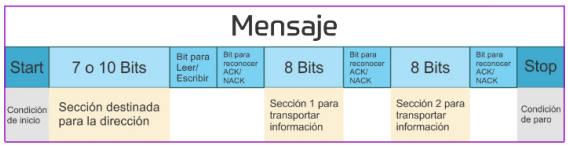

Figura 1. composicion de un mensaje enviado en I2C [3]

title",çonst Qstring text");
y de esta manera, al ejecutar el programa, saldrá un venta aparte un mensaje haciendole la peregunta al usuario. [8]

- **QFileDialog**: Esta librería permite al usuario seleccionar archivos o acceder a direcciones de archivos. Para acceder a esta lbrería se pone el el header # include <QFileDialog>; para su implementación, en un botón previamente creado, en la parte de codigo del botón se le agrega la siguiente línea de código: [9]
  QString filename =
  QfileDialog::getOpenFileName(QWidget*parent,
  QString
  caption, direción a la que se quiere acceder);
- **QPushButton**: Esta librería sirve para cerar botones con los cuales se pueda abrir o cerrar una acción propia de la aplicación que se desee crear. Para implementarlos, al igual que todas las librerías, primero se tiene que llamar: # include <QPushButtom>; luego se le da la información de la acción que se le de a realizar: [10]
  QpushButtom ∗ptr=ui−>quitar;
  QObject::connect(ptr,Signal(clicked()),parent,SLOT(close()));
- **QByteArray**: Esta librería provee las herramientas para realizar arreglos de bytes, en este caso, para ser enviados y recibidos por un puerto serial. se llama con # include <QByteArray> y se utiliza con la siguiente función [11]:
  QByteArray(const char *data, int size = -1)

Debido a que Qt no posee componentes para manejar los puertos seriales, en la investigación se encontro que hay ina librería independiente la cual tiene la capacidad de manejar los puertos seriales. esta biblioteca es la **QexSerialPort**. para hacer uso de esta biblioteca, esta debe ser descargada y se le debe hacer la siguiente configuración en el archivo .pro:

TARGET = qextserialportsencilltransmisio
CONFIG += console
CONFIG -= appbundle # MacOS no es el objetivo

Gracias a que **QexSerialPort** funciona igual que una clase normal de Qt, se utilizará de la misma manera que cualquier otra y se puede utilizar con el siguiente constructor, que es de los más utilizados: [12]

QextSerialPort::QextSerialPort ();

II. Objetivos

- Realizar una un protocolo de comunicación bidireccional donde ambas aplicaciones tengan la capacidad de enviar, recibir y guardar archivos.
- Las aplicaciones deben mostrar una gráfica que muestre el porcentaje de bytes de transferencia y velocidad.
- Crear un Deploy para las aplicaciones.

III. Metodología

para realizar el programa, se siguieron los siguientes pasos:

Para comenzar con la realización de la interfaz, primero se creo un archivo .widget y se agrego la cabecera:

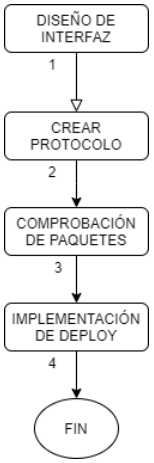

Figura 2. Diagrama para realizar las aplicaciones

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <QFileDialog>
#include <QFileInfo>
#include <QDebug>
#include <QSerialPort>
#include <QSerialPortInfo>

//constructor
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    foreach(const QSerialPortInfo &info, QSerialPortInfo::availablePorts()){
        qDebug() << "Name: " << info.portName();
        qDebug() << "Description: " << info.description();
        qDebug() << "Manufacturer: " << info.manufacturer();
        qDebug() << "Vendor: " << info.vendorIdentifier();
        qDebug() << "Product: " << info.productIdentifier();
        ui->port_box->addItem(info.portName());
```

Figura 3. Cabecera del programa

Luego se crearon los botones de la interfaz. El primero fue un botón denominado Files ya que contiene las opciones guardar y abrir.

Tambien se creo dos botones de enviar y recibir: Para la

```
void MainWindow::on_send_button_clicked()
{
    mode = 'w';
}

void MainWindow::on_receive_button_clicked()
{
    mode = 'r';
}
```

Figura 4. botón enviar y recibir

funcion guardar primero se agrego al QFile en el archivo .cpp, donde se le creo el parent, el directorio y se le asigno el tipo de archivo a abrir. Para programar el funcionamiento de este botón, se hizo un if donde se le pregunta al usuario si el archivo esta vacio; si el archivo está vacío, su respuesta será un warning, de lo contrario, procederá a preguntar si desea guardar el archivo (ver diagramas en el anexo)

Figura 5. Método para guardar archivo

Figura 6. Método para abrir archivo

Luego se hizo el diseño para el puerto serial; donde se implemento un botón para mostrar el puerto a emplear y tambien se realizo un botón para abrir o cerrar el puerto serial y crear una comprovación mediante una función :

```
//metodo de clicked sobre boton del port
void MainWindow::on_port_button_clicked()
{
    ui->port_button->setCheckable(true);
    QString puerto = ui->port_box->currentText();
    if(ui->port_button->isChecked()){
        openSerialPort(puerto);
    }else{
        closeSerialPort();
    }
}
```

Figura 7. condicionamiento del serial port

```
//metodo para iniciar apertura del puerto
void MainWindow::openSerialPort(QString p)
{
    qDebug()<<"puerto abierto";
    disconnect(m_serial,SIGNAL(readyRead()),this, SLOT(readSerial()));
    m_serial->setPortName(p);
    m_serial->setBaudRate(QSerialPort::Baud115200);
    m_serial->setDataBits(QSerialPort::Data8);
    m_serial->setParity(QSerialPort::NoParity);
    m_serial->setStopBits(QSerialPort::OneStop);
    m_serial->setFlowControl(QSerialPort::NoFlowControl);
    connect(m_serial,SIGNAL(readyRead()),this, SLOT(readSerial()));

    if(m_serial->open(QIODevice::ReadWrite)){
        ui->port_button->setText("Close Port");
        ui->port_box->setDisabled(true);
    }else{
        QMessageBox::critical(this,tr("Error"), m_serial->errorString());
    }
```

Figura 8. Apertura del puerto serial

Como en la guia se requería que hubiese una gráfica que representara el porcentaje de bytes de transferencia y la velocidad de bytes/s, entonces se implemento la clase QPushButtom ya que esta permite crear un botón con una función que permiriera abrir una gráfica como la que se requiere:

Para armar los paquetes de datos se empleo la biblioteca QByteArray y tambien se implenento la función packet::append, cuando el programa esta listo comienza enviando dos 0x01, luego manda la extencion, los datos y el paquete; para la comprovacion de este, se hace mediante una operación XOR y cuando ya finaliza la operacion, envia dos 0x05.

```
void MainWindow::writeSerial(){
    QByteArray data;
    data = ui->plainTextEdit->toPlainText().toUtf8();
    int size = data.size();
    unsigned char Xor;
    QByteArray packet = 0;
    unsigned char ext = extetion.suffix().toInt();
    for (int i = 0; i < size - 1 ; i++){
        packet.append(0x01);
        packet.append(0x01);
        packet.append(ext);
        packet.append(data.at(i));
        Xor = checkSum(&packet);
        packet.append(Xor);
        packet.append(0x05);
        packet.append(0x05);
        qDebug() << "paquete"<<packet;
        if(m_serial->isWritable()){
            m_serial->write(packet);
            num_packet ++;
            qDebug() << "se envio packet" << num_packet;
        }
        packet.clear();
    }
}
```

Figura 9. Lectura de datos

Al hacer la comparación, el programa recive los datos y si los datos enciados son correctos muestra un 015, de lo contrario muestra un 10.

```
unsigned char MainWindow::checkSum(QByteArray  *b)
{
    qint16 b_len = b->length();

    unsigned char Xor = 0;

    for ( int i = 0 ; i < b_len ; i ++ )
    {
      Xor = Xor ^ b->at(i);
    }
    return  Xor;
}
//metodo para cerrar el puerto
void MainWindow::closeSerialPort(){
    if(m_serial->isOpen()){
        m_serial->close();
        ui->port_button->setText("Open Port");
        ui->port_box->setDisabled(false);
    }
}
```

Figura 10. Comprovación

POROTOCOLO

0x01	0x01	Extension	Data	Check	0x05	0x05	0x07

IV. Resultados

Los resultados del código fueron los siguientes:

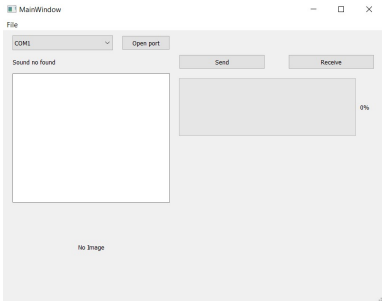

Figura 11. Interfaz de la aplicación con el puerto abierto

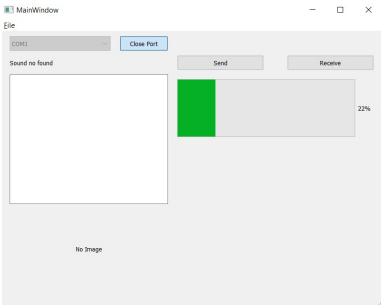

Figura 12. interfaz con el puerto cerrado y recibiendo un mensaje

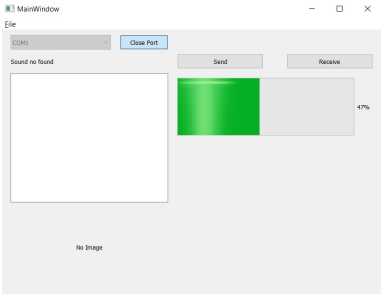

Figura 13. interfaz con la barra de carga

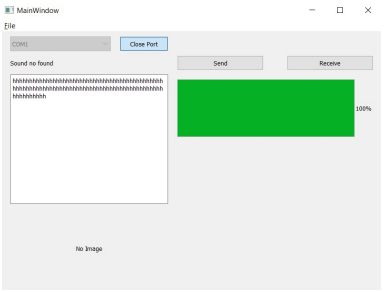

Figura 14. mensaje recivido con la barra cargada al 100 %

V. Análisis de resultados

- No se logro la implementacion de la grafica sujerida por el docente, ya que se tuvo mal manejo del tiempo empleado en el dasarrollo del software debido a factores como prubas del codigo, lectura de documentancion,etc.
- La implemetancion del protocolo funciono de manera correcta, mas sin embargo se debe enfatizar en el uso y creacion de estos mismos, para no estancarse durante la creacion de este mucho tiempo.
- Usamos Mainwindow detro del entorno de QT esta vez para el desarrollo del software, ya que este nos facilitaba la creacion de un tool bar para crear,guardar y guardar como archivos dentro del directorio especificado por el programa.

VI. Conclusiones

- Para una mayor eficiencia y fácilidad a la hora de realizar un protocolo con interfaz, es muy útil primero realizar la interfaz gráfica.
- Es importante guardar los datos recividos, para análizar si el protocolo esta funcionando de la manera en que se espera.
- Realizar un protocolo de comunicaciónes, es una tarea bastante larga y requiere de tiempo para que la labor sea exitosa.
- Crear un protocolo de comunicación bidireccional es muy útil ya que esta es una manera muy ordenada de enviar datos y si se quisiera, se podrian destinar a una ubicación que puede tener una función especifica.

Referencias

[1] I. C. Camacho, “Laboratorio creación de un protocolo de comunicación,” *Universidad Sergio Arboleda*, 2020.
[2] J. Gomar. (2018). [Online]. Available: ”https://www.profesionalreview.com/2018/12/19/puerto-serial-y-puerto-paralelo”
[3] V. Autores. (2017) Fundamentos de protocolo i2c. [Online]. Available: ”https://teslabem.com/nivel-intermedio/fundamentos/”
[4] T. Q. C. Ltd. (2020) Qdebug class.
[5] T. Q. company. (2020) Qmainwindow class. [Online]. Available: https://doc.qt.io/qt-5/qmainwindow.html
[6] T. Q. company Ltd. (2020) Qserialport class. [Online]. Available: http://sindominio.net/ash
[7] T. Q. company Ltd. (2020) Qserialportinfo class. [Online]. Available: https://doc.qt.io/qt-5/qserialportinfo.html
[8] T. Q. company Ltd. (2020) Qmessagebox class. [Online]. Available: https://doc.qt.io/qt-5/qmessagebox.html
[9] T. Q. Ltd. (2020) Qfiledialog class. [Online]. Available: https://doc.qt.io/qt-5/qfiledialog.html
[10] T. Q. company Ltd. (2020) Qpushbuttom class. [Online]. Available: https://wiki.qt.io/How$_{t}o_{U}se_{Q}PushButton/es$
[11] ——. (2020) Qbytearray class. [Online]. Available: https://doc.qt.io/qt-5/qbytearray.html
[12] varios autores. (2012) El puerto serie y qt. [Online]. Available: http://www.disca.upv.es/aperles/qt/port$_{s}erie_{Q}t/port_{s}erie_{Q}t.html$

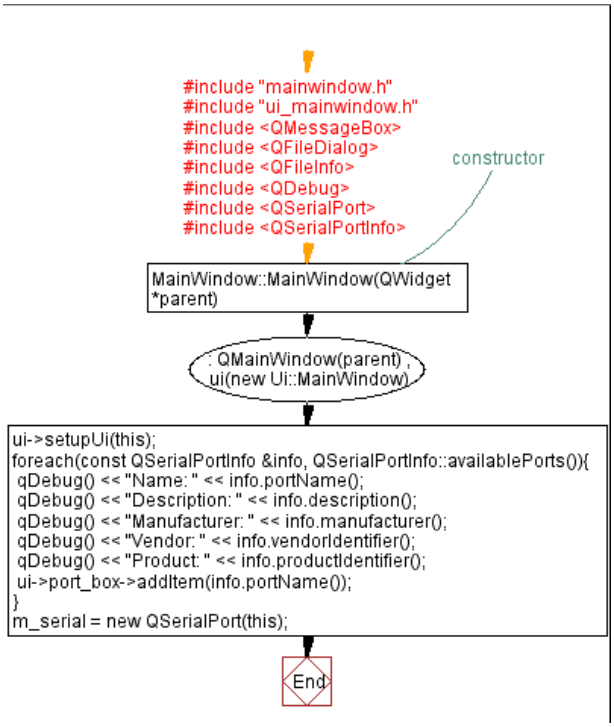

Figura 15. Cabecera y constructor

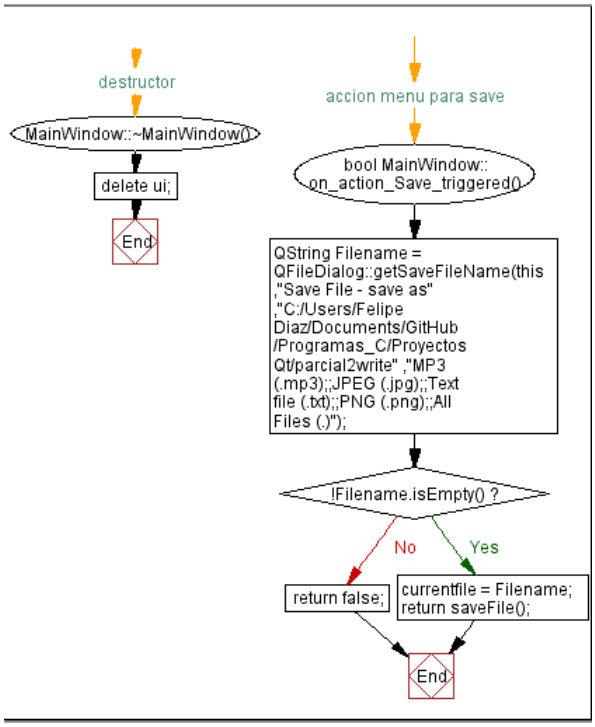

Figura 16. destructor

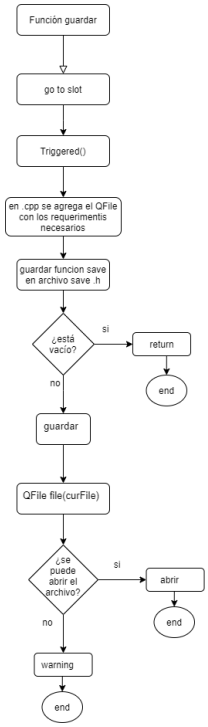

Figura 17. Flujo grama botón guardar

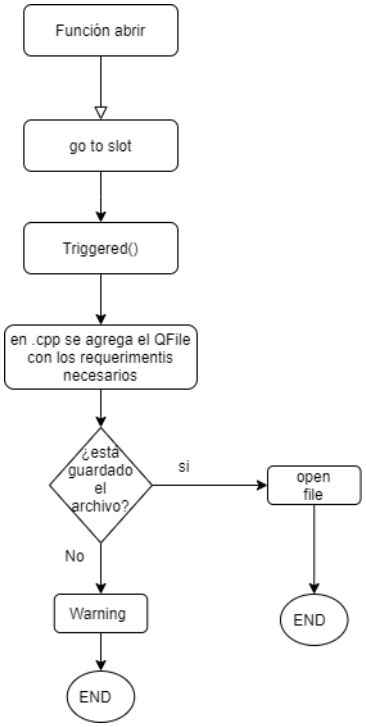

Figura 18. Diagrama del boton abrir

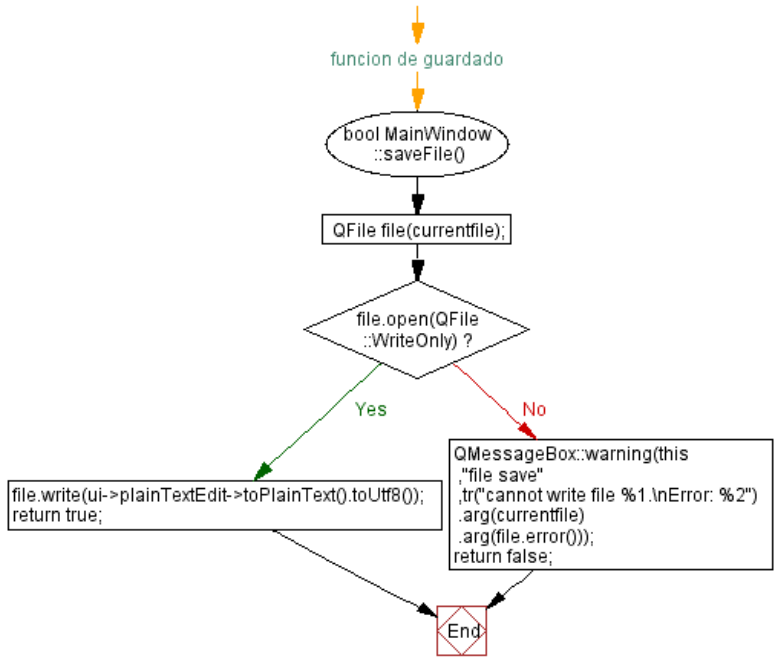

Figura 19. Botón guardado

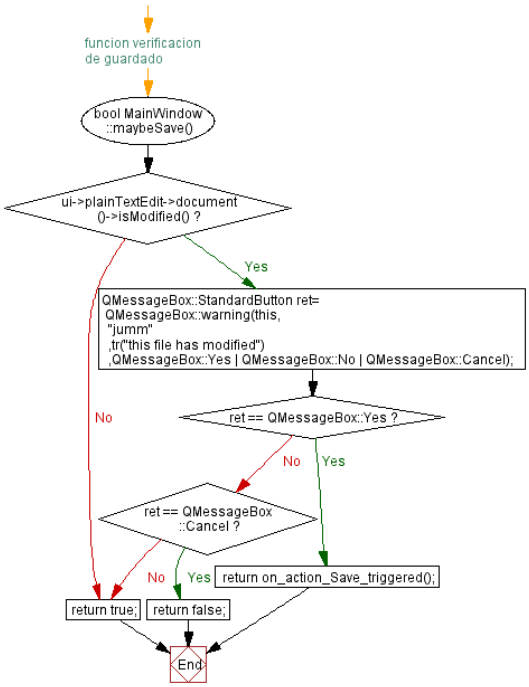

Figura 20. verificacion de guardado

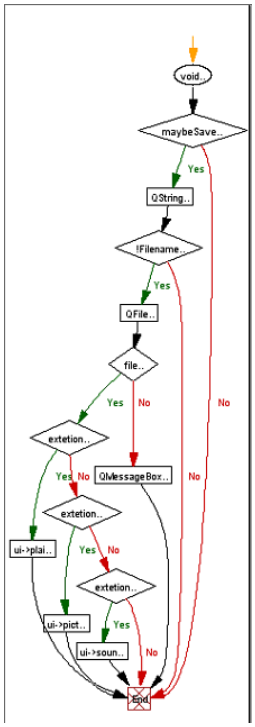

Figura 21. accion para subir archivo

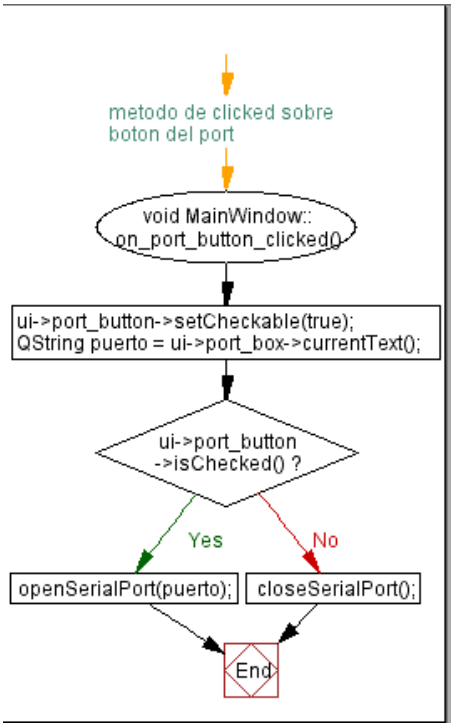

Figura 22. Caption

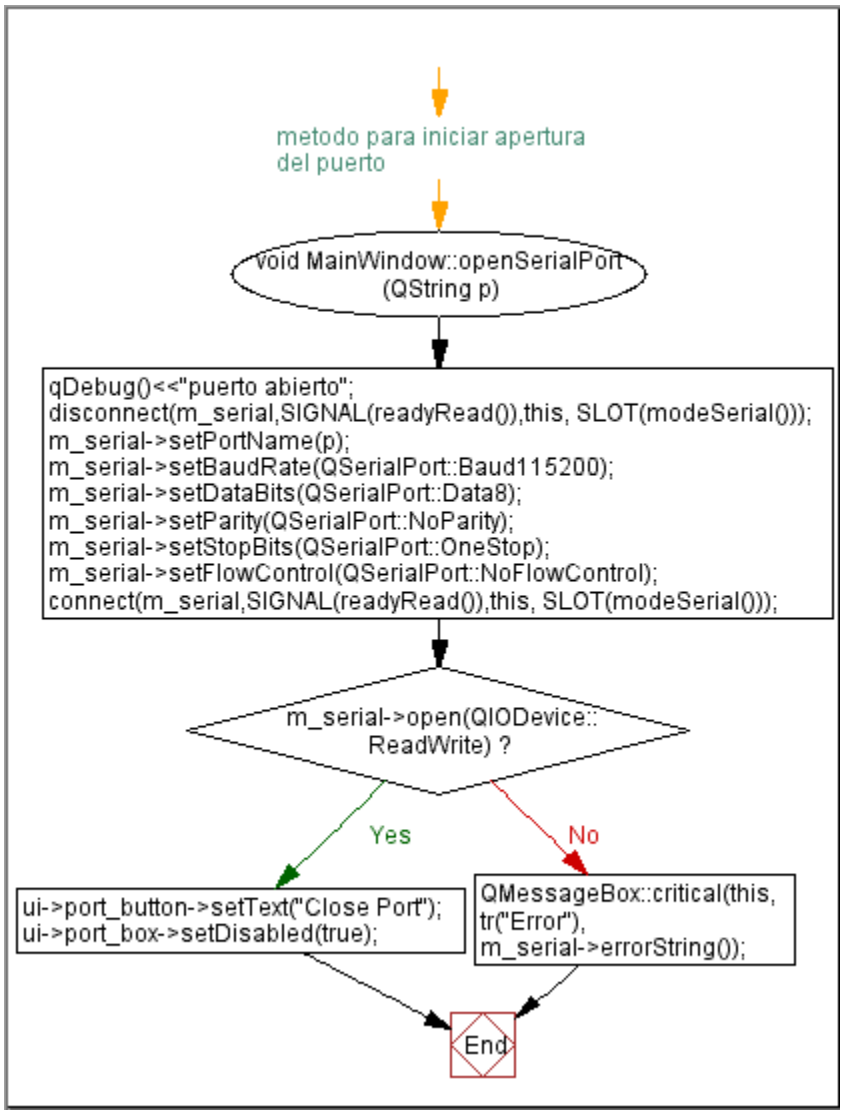

Figura 23. método para iniciar la aperturaa del puerto

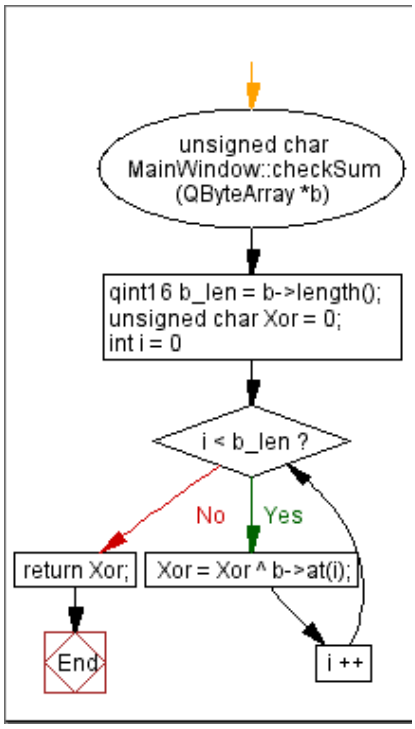

Figura 24. Caption

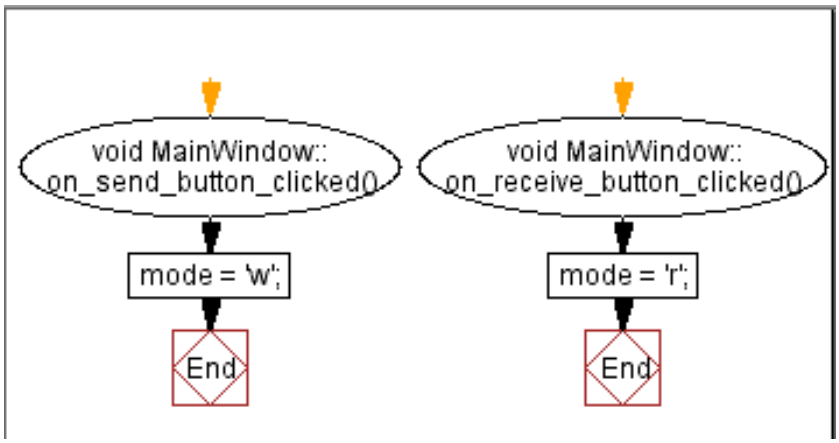

Figura 25. botones enviar y recibir