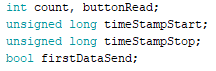
|  |
| --- |
| Diseño de Interfaces |
| Tarea 03 |
| Comunicación Serial |

|  |
| --- |
| Aldo Vargas  213495653 |

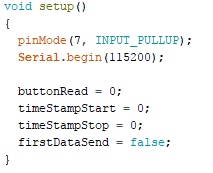
El objetivo de la siguiente práctica es el de comunicar de manera serial un dispositivo Arduino, a una interfaz elaborada con QT- Design, en donde se representa el valor recibido, y se muestra en la barra de progreso. La comunicación entre los dispositivos será a una velocidad de 115200 bauds/s.

Programación Arduino

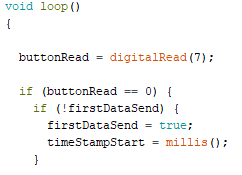
Para el programa Arduino, después de crear un nuevo proyecto se declaran variables enteras para un contador y una lectura analógica. Además, se declaran dos variables positivas para almacenar los tiempos necesarios para evitar el rebote eléctrico de la entrada del botón. También, se declara una bandera para el primer envío de datos.



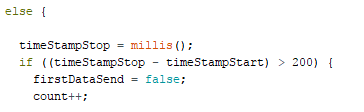
En la función setup () inicializamos el pin 7 como entrada con PULLUP, la comunicación serial a una velocidad de 115200 bauds por segundo, así como las variables globales.



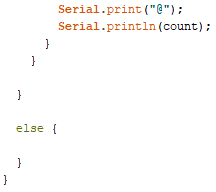
La función loop () comienza almacenando la lectura del botón, si es positiva, al entrar por primera vez guarda el valor del tiempo de ejecución.



Después de la primera ejecución, que es el evento de el botón siendo presionado, al soltarlo se guardará el segundo valor siendo el evento de liberar completamente el botón. Con estos valores, se calcula el tiempo real de presión, para evitar “botonazos”. Si el valor se mayor a los 200 ms el contador aumentará el valor para ser enviado.



Si el valor enviado acaba de ser el 100, el siguiente valor será 0. Finalmente, se empaqueta el envío de datos con un carácter y se envía de manera serial.



Programación en QT

Una vez cargado el programa en el Arduino, se crea un nuevo proyecto en QT-Design para aplicaciones con Widget y se configura para la versión más nueva. El diseño de la interfaz del programa que representará los valores recibidos desde el Arduino por medio de la comunicación serial consiste en una barra de progreso, y una etiqueta.

En el archivo de configuraciones .pro se agrega en la línea 7 la adición de serialport para comunicación.

QT += core gui serialport

Dentro del archivo MainWindow.h se incluyen las librerías necesarias para las funciones de comunicación serial.

#include <QDebug>

#include <QMainWindow>

#include <QSerialPort>

#include <QSerialPortInfo>

Dentro de los accesos privados de la clase, se declara un puntero del tipo QserialPort, al igual que una bandera para controlar la conexión del Arduino. Después, se declaran los prototipos de funciones para conectar y recibir datos del Arduino.

QSerialPort\* Arduino;

bool ArduinoIsConnected = false;

void connectArduino();

void serialReceiver();

Dentro del MainWindow se inicializa con cero la barra de progreso, así como la función de conexión.

La función de conexión empieza creando un objeto de la clase QserialPort, y configurando la bandera de conexión en falso. A continuación, se ejecuta la función connect() para el puerto, así como variables para la información de los dispositivos.

void MainWindow::connectArduino()

{

ArduinoIsConnected = false;

Arduino = new QSerialPort(this);

connect(Arduino, &QSerialPort::readyRead, this, &MainWindow::serialReceiver);

QString deviceName = "";

int productID = 0;

La función continúa imprimiendo en la consola los valores del objeto de comunicación serial.

qDebug() << "Puertos disponibles: " << QSerialPortInfo::availablePorts().length();

/\* Funcion ejecutada para cada puerto disponible \*/

foreach (const QSerialPortInfo& serialPortInfo, QSerialPortInfo::availablePorts()) {

qDebug() << "Identificador del fabricante (VENDOR ID): " << serialPortInfo.hasVendorIdentifier();

if (serialPortInfo.hasVendorIdentifier()) {

qDebug() << "ID Vendedor " << serialPortInfo.vendorIdentifier();

qDebug() << "ID Producto: " << serialPortInfo.productIdentifier();

/\* Condicion para nombrar el dispositivo \*/

if (serialPortInfo.productIdentifier() == 66 || serialPortInfo.productIdentifier() == 67 || serialPortInfo.productIdentifier() == 29987) {

/\* Bnadera de conexion exitosa \*/

ArduinoIsConnected = true;

/\* Extraccion de datos del dispositivo \*/

deviceName = serialPortInfo.portName();

productID = serialPortInfo.productIdentifier();

}

}

}

Si la conexión ha sido exitosa, se configura cada valor de la clase QserialPort, especificando la velocidad de transmisión a la misma que la velocidad de envío del Arduino. Una vez configurado, se condiciona la etiqueta para especificar el tipo de dispositivo conectado. En caso de que la conexión no sea exitosa, se imprime un error.

/\* Si la conexion es exitosa \*/

if (ArduinoIsConnected) {

/\* Configuración de objeto serial \*/

Arduino->setPortName(deviceName);

Arduino->*open*(QIODevice::ReadWrite);

Arduino->setDataBits(QSerialPort::Data8);

Arduino->setBaudRate(QSerialPort::Baud115200);

Arduino->setParity(QSerialPort::NoParity);

Arduino->setStopBits(QSerialPort::OneStop);

Arduino->setFlowControl(QSerialPort::NoFlowControl);

ui->label->clear();

/\* Impresion de informacion del dispositivo \*/

qDebug() << "Producto: " << productID;

if (productID == 67)

ui->label->setText("Arduino UNO R3 conectado");

else if (productID == 66)

ui->label->setText("Arduino Mega conectado");

else if (productID == 29987)

ui->label->setText("Arduino Nano Conectado");

else

ui->label->setText("Error 3");

}

/\* Conexión no exitosa \*/

else {

ui->label->clear();

ui->label->setText("No hay Arduino");

}

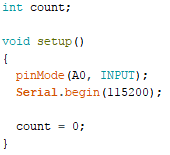
}

La ejecución del programa enviará el valor del contador de Arduino después de cada pulsación correcta de botón, imprimiendo el valor en la consola de Debugeo y representado en la barra de progreso de la interfaz.

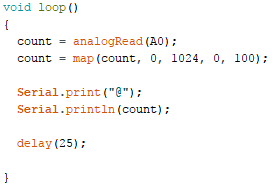
Lectura Analógica Tarea Extra

Cambiando el código en Arduino podemos para llevar a cabo una función alterna de lectura analógica. Esta función obtiene una lectura analógica a través de un pin digital.

Primero se declara un contador, y la comunicación serial a 115200 baud/s.



La función loop hace tres funciones: Leer el puerto analógico, mapearlo a un rango de 0 a 100 y enviarlo por comunicación serial para que sea cargado como un porcentaje en la barra de progreso.



Conclusión

Dentro la gran mayoría de aplicaciones que han sido implementadas en una interfaz gráfica de usuario, se representan valores por medio de elementos visuales, como en este caso la barra de progreso. Este elemento en función consigue imprimir los valores que han sido comunicados de manera serial correctamente, debido a que la configuración entre los valores de velocidad de transmisión, así como la estructura en la que vienen empaquetados los datos, ha sido especificada de ambos lados.

La comunicación serial tiene ventajas importantes a otros tipos de comunicación, remarcado la sencillez de la misma, así como la facilidad con la que se configura y gestiona. Este enlace entre dispositivos funciona siempre que ambos dispositivos estén “de acuerdo” a comunicarse, y que hayan sido correctamente enlazados. Posiblemente debido a mi inexperiencia en la rama puedo imaginar un enlace de comunicación serial que pueda antes que nada detectar la velocidad en la que se están transmitiendo datos. De esta manera y usando una analogía de dos personas comunicándose verbalmente, la comunicación serial podría descifrar el lenguaje que se intenta enviar. Con esto, será posible igualar la velocidad de transmisión, y entender el mensaje.

Bibliografía

<https://hetpro-store.com/TUTORIALES/qt-creator-y-arduino-qserialport/>

<https://hetpro-store.com/TUTORIALES/arduino-serialevent/>

<https://hetpro-store.com/TUTORIALES/puerto-serial/>

Códigos

Código Arduino

/\* Contador Serial con conexion a QT \*/

/\* Declaracion de variables \*/

int count, buttonRead;

unsigned long timeStampStart;

unsigned long timeStampStop;

bool firstDataSend;

/\* Funcion de configuracion \*/

void setup()

{

pinMode(7, INPUT\_PULLUP);

Serial.begin(115200);

/\* Inicializacion de variables \*/

buttonRead = 0;

timeStampStart = 0;

timeStampStop = 0;

firstDataSend = false;

}

void loop()

{

buttonRead = digitalRead(7);

/\* Boton presionado \*/

if (buttonRead == 0) {

if (!firstDataSend) {

/\* Bandera de inicio de envio \*/

firstDataSend = true;

/\* Guarda tiempo actual \*/

timeStampStart = millis();

}

else {

/\* Guarda tiempo al soltar botton

para calcular el Rebote \*/

timeStampStop = millis();

/\* Calcula tiempo minimo para lectura correcta \*/

if ((timeStampStop - timeStampStart) > 200) {

/\* Cambia bandera \*/

firstDataSend = false;

/\* Incrementa el dato a enviar \*/

count++;

/\* Evalua el limite de datos \*/

if (count > 100)

count = 0;

/\* Imprime al monitor serial \*/

Serial.print("@");

Serial.println(count);

}

}

}

else {

}

}

Archivo .pro

#-------------------------------------------------

#

# Project created by QtCreator 2018-01-31T20:26:40

#

#-------------------------------------------------

QT += core gui serialport

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

TARGET = QT-Arduino-3

TEMPLATE = app

# The following define makes your compiler emit warnings if you use

# any feature of Qt which has been marked as deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT\_DEPRECATED\_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

main.cpp \

mainwindow.cpp

HEADERS += \

mainwindow.h

FORMS += \

mainwindow.ui

MainWindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QDebug>

#include <QMainWindow>

#include <QSerialPort>

#include <QSerialPortInfo>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

explicit MainWindow(QWidget\* parent = 0);

~*MainWindow*();

private:

Ui::MainWindow\* ui;

/\* Creacion de objeto con comunicacion serial\*/

QSerialPort\* Arduino;

/\* Bandera de Conexion \*/

bool ArduinoIsConnected = false;

/\* Prototipos de Funciones \*/

void connectArduino();

void serialReceiver();

};

#endif // MAINWINDOW\_H

Main Window.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

/\* Constructor de Clase MainWindow \*/

MainWindow::MainWindow(QWidget\* parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

/\* Configuracion de Contexto para la interfaz de usuario \*/

ui->setupUi(this);

/\* Inicializar valor de la barra de progreso \*/

ui->progressBar->setValue(0);

/\* Conexion para el objeto de comunicacion serial \*/

connectArduino();

}

/\* Destructor de clase MainWindow \*/

MainWindow::~*MainWindow*()

{

Arduino->*close*();

delete ui;

}

/\* Funcion para leer los datos enviados del objeto de Comunicacion Serial \*/

void MainWindow::serialReceiver()

{

/\* Variables para desempaquetar el dato recibido \*/

int index0, index1;

/\* Variable para casting del Caracter enviado por Objeto Serial \*/

int intData;

/\* Arreglo de Bites \*/

QByteArray serialRead;

/\* Variable para el Caracter enviado por Objeto Serial\*/

QString extractedData;

/\* Evalua el objeto serial conectado y disponible \*/

if (ArduinoIsConnected && Arduino->isReadable()) {

/\* Obtiene una lectura del objeto serial \*/

serialRead = Arduino->readLine();

/\* Guarda los indices fronterizos del dato recibido \*/

index0 = serialRead.indexOf("\n");

index1 = serialRead.indexOf("@");

/\* Extraccion de datos desempaquetado \*/

extractedData = serialRead.mid(index1 + 1, (index0 - index1));

/\* Guarda el valor del dato en entero \*/

intData = extractedData.toInt();

qDebug() << "Dato: " << intData << endl;

/\* Evalua que los indices fronterizos esten en posiciones correctas \*/

if (index0 >= 0 && index1 >= 0) {

qDebug() << "Datos extraidos: " << extractedData.toUtf8().constData();

qDebug() << "Datos leidos: " << serialRead << endl;

/\* Carga el valor numerico en la barra de progreso \*/

ui->progressBar->setValue(intData);

}

}

}

/\*Funcion para Conectar el objeto serial \*/

void MainWindow::connectArduino()

{

/\* Bandera de control \*/

ArduinoIsConnected = false;

/\* Inicializacion del objeto serial con un contexto \*/

Arduino = new QSerialPort(this);

/\* Funcion de conexion \*/

connect(Arduino, &QSerialPort::readyRead, this, &MainWindow::serialReceiver);

/\* Valores del dispositivo conectado serialmente \*/

QString deviceName = "";

int productID = 0;

/\* Busqueda de Puertos Libres \*/

qDebug() << "Puertos disponibles: " << QSerialPortInfo::availablePorts().length();

/\* Funcion ejecutada para cada puerto disponible \*/

foreach (const QSerialPortInfo& serialPortInfo, QSerialPortInfo::availablePorts()) {

qDebug() << "Identificador del fabricante (VENDOR ID): " << serialPortInfo.hasVendorIdentifier();

if (serialPortInfo.hasVendorIdentifier()) {

qDebug() << "ID Vendedor " << serialPortInfo.vendorIdentifier();

qDebug() << "ID Producto: " << serialPortInfo.productIdentifier();

/\* Condicion para nombrar el dispositivo \*/

if (serialPortInfo.productIdentifier() == 66 || serialPortInfo.productIdentifier() == 67 || serialPortInfo.productIdentifier() == 29987) {

/\* Bandera de conexion exitosa \*/

ArduinoIsConnected = true;

/\* Extraccion de datos del dispositivo \*/

deviceName = serialPortInfo.portName();

productID = serialPortInfo.productIdentifier();

}

}

}

/\* Si la conexion es exitosa \*/

if (ArduinoIsConnected) {

/\* Configuración de objeto serial \*/

Arduino->setPortName(deviceName);

Arduino->*open*(QIODevice::ReadWrite);

Arduino->setDataBits(QSerialPort::Data8);

Arduino->setBaudRate(QSerialPort::Baud115200);

Arduino->setParity(QSerialPort::NoParity);

Arduino->setStopBits(QSerialPort::OneStop);

Arduino->setFlowControl(QSerialPort::NoFlowControl);

ui->label->clear();

/\* Impresion de informacion del dispositivo \*/

qDebug() << "Producto: " << productID;

if (productID == 67)

ui->label->setText("Arduino UNO R3 conectado");

else if (productID == 66)

ui->label->setText("Arduino Mega conectado");

else if (productID == 29987)

ui->label->setText("Arduino Nano Conectado");

else

ui->label->setText("Error 3");

}

/\* Conexion no exitosa \*/

else {

ui->label->clear();

ui->label->setText("No hay Arduino");

}

}