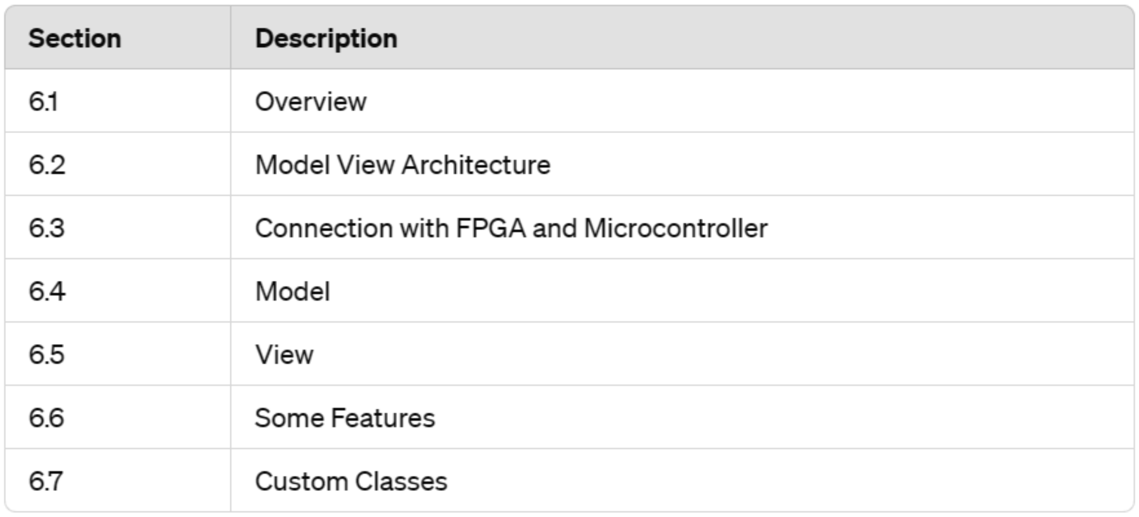
Chapter 6 Application Development



Section	Description
6.1	Overview
6.2	Model View Architecture
6.3	Connection with FPGA and Microcontroller
6.4	Model
6.5	View
6.6	Some Features
6.7	Custom Classes

6.1 Overview

We have developed a desktop application to display the results of the spectrum analysis process. It has been developed using Qt Framework. Specifically, using QtQuick. In QtQuick, a way of developing applications we make a UI layer in QML and backend in C++. This application follows the model-view architecture of software development. It has three main responsibilities:

1. A connection between FPGA and application for data transfer
2. Processing and display of data
3. A connection between ESP32 and application for control commands

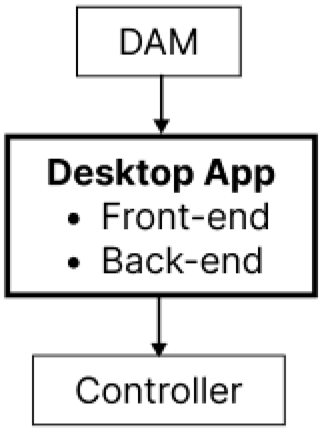


Fig 6.1 System block diagram

This application has a UI layer and a backend. The UI layer has been developed in Qt Modeling Language (QML), and the backend has been developed in C++.

6.2 Model View Architecture

To develop this application, we have used the model-view architecture of the software development process. This architecture has two distinct components:

1. Model
2. View

A model is the part of architecture that is responsible for taking data from a data source (FPGA in our case) and stores it in a data structure. This part of our application is written in C++.

View is the part of architecture that is responsible for taking data from model and displaying it on screen. This part of our application is written in Qt Modeling Language (QML).

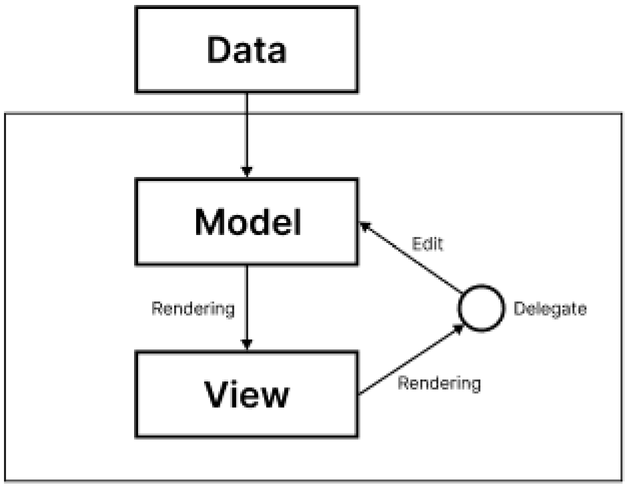


Fig 6.2 Model-view architecture block diagram

6.3 Connection with FPGA and Microcontroller

These are the two main links that transfer data into and out of this application. One of them is with FPGA and other with microcontroller.

To make the connection of FPGA with desktop application, we have used Qt Network Library. Specifically, we have used QTcpSocket and QTcpServer classes.

To make the connection of the desktop application with ESP32, we have used Qt Serial Port Library. Specifically, we have used QSerialPort and QSerialPortInfo classes.

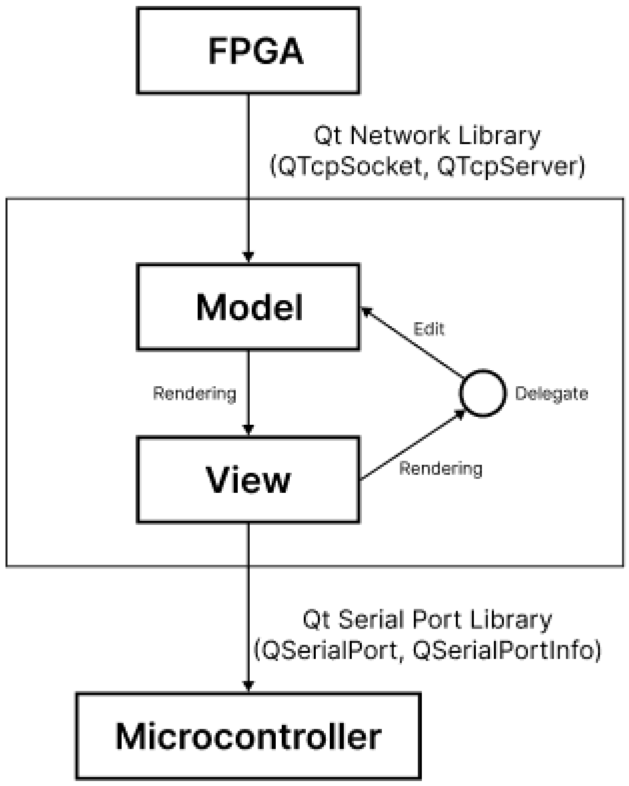


Fig 6.3 Complete system Block Diagram

6.4 Model

This part is written in C++. It is the part of model/view architecture that is responsible for holding data. To develop this part, we have used QAbstractTableModel. We used this class and subclassed it to make a class called DataModel. It acts as a 2D array of floating-point objects, QPointF. It has two columns and resizable number of rows based on data points available. This model is connected to ChartView on UI side. ChartView is a QML type. This is responsible for displaying charts of different kinds on screen.

6.5 View

It is the part is written in Qt Modeling Language (QML). It is the part of model/view architecture that acts as presentation layer. Note, UI and logic layers are completely separated. Some of the QML types we used along the way are:

1. QtQuick
2. QtQuick.Controls
3. QtCharts
4. QtQuick.Layouts

The wireframe (or something we call plotting) of UI layer looks like this:

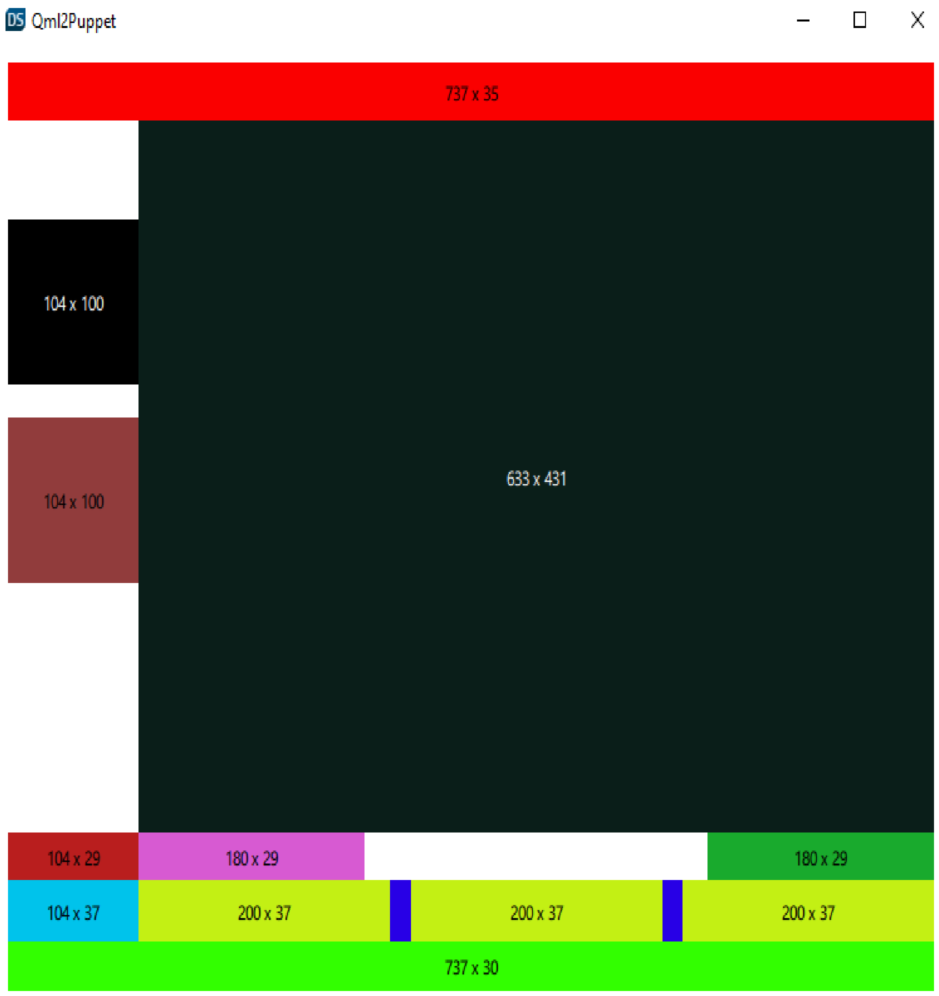
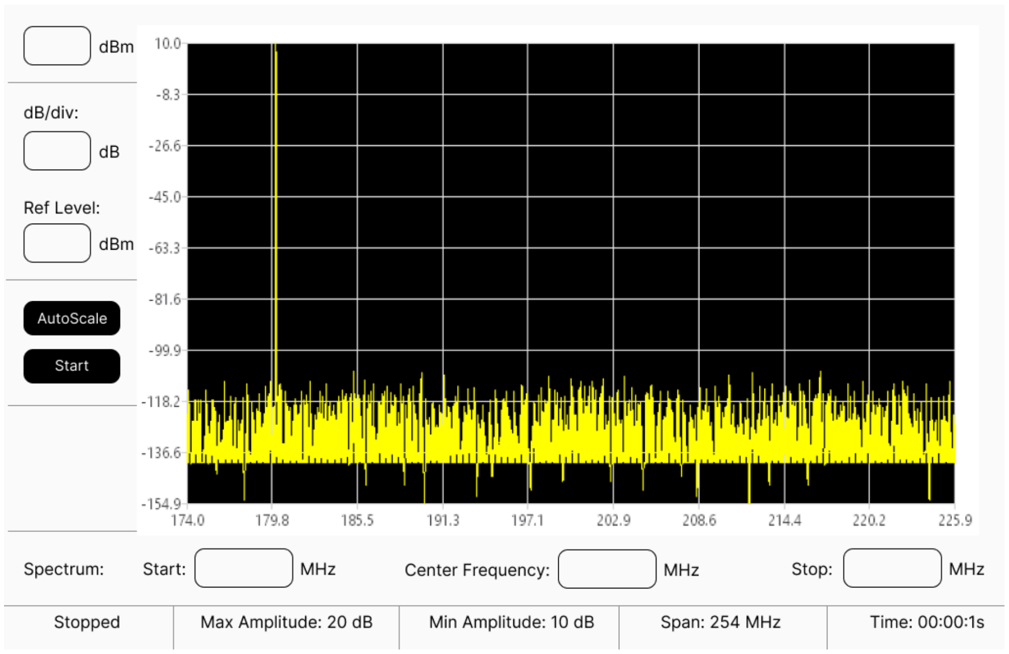


Fig 6.4 Wireframe

Figma design of UI layer looks like this,



Stopped | Max Amplitude: 20 dB | Min Amplitude: 10 dB | Span: 254 MHz | Time: 00:00:1s

Fig 6.5 Figma Design

6.6 Features

Some features of this application are:

1. Auto Scale Option
2. Start Frequency
3. Stop Frequency
4. Center Frequency
5. Span
6. Max & Min Amplitude
7. Running Time of Application
8. Reference Level
9. Adjustable dB/div

6.7 Custom Classes

This part Following classes have been developed for this application:

TCPPackets: this class is responsible for connecting hardware with application.

Conversion: this class is responsible for conversion of received data to decimal from Qn.m format. It also holds code for dB conversions.

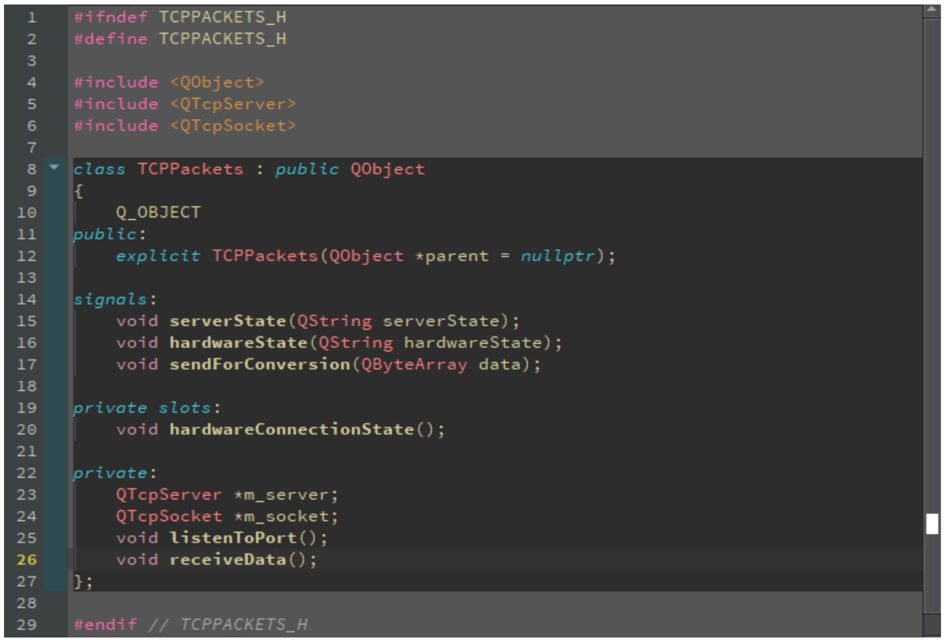
DataModel: this class is responsible for holding all data as a model and providing it to the UI layer.

Timer: this class is responsible for keeping account of running time of application.

cPanel: this class is responsible for dealing with start frequency, stop frequency, center, frequency, and all adjustable data from UI layer is here. So, the user who interacts with that data on UI side is interacting with this data on C++ side.

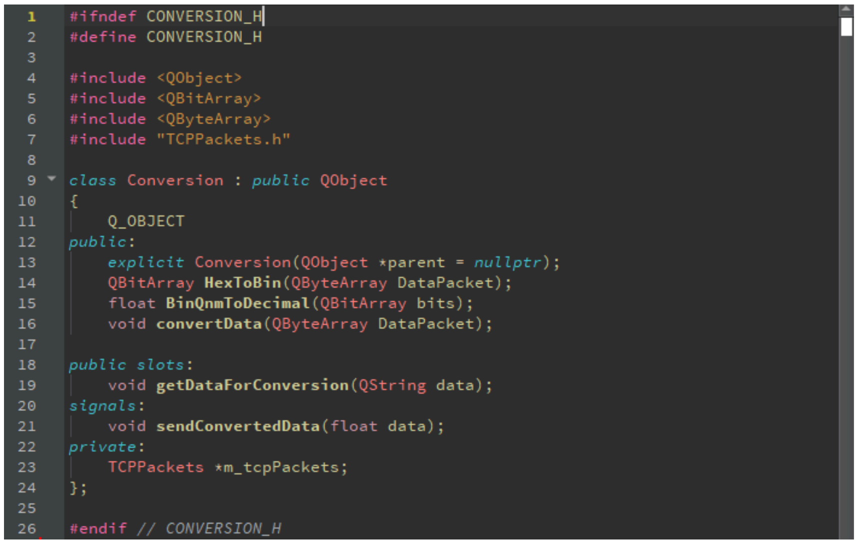
This is the class implementation of two most crucial parts of application, namely, TCPPackets and DataModel.

TCPPackets.h



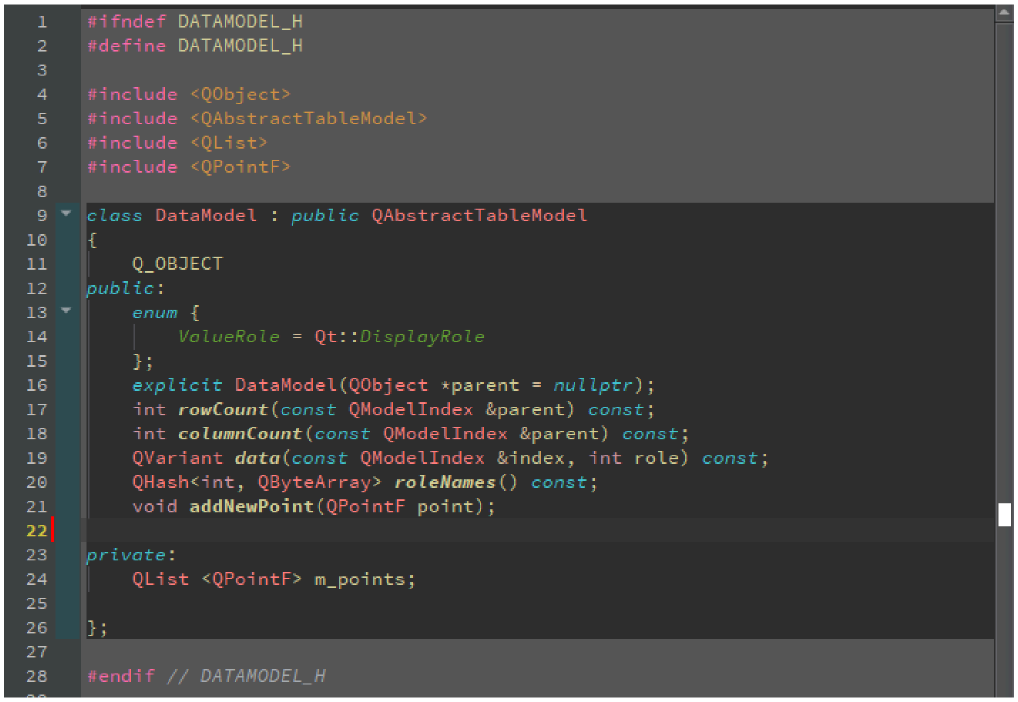
```
#ifndef TCPPACKETS_H
#define TCPPACKETS_H

#include <QObject>
#include <QTcpServer>
#include <QTcpSocket>

class TCPPackets : public QObject
{
    Q_OBJECT
public:
    explicit TCPPackets(QObject *parent = nullptr);

signals:
    void serverState(QString serverState);
    void hardwareState(QString hardwareState);
    void sendForConversion(QByteArray data);

private slots:
    void hardwareConnectionState();

private:
    QTcpServer *m_server;
    QTcpSocket *m_socket;
    void listenToPort();
    void receiveData();
};

#endif // TCPPACKETS_H
```

Conversion.h



```
#ifndef CONVERSION_H
#define CONVERSION_H

#include <QObject>
#include <QBitArray>
#include <QByteArray>
#include "TCPPackets.h"

class Conversion : public QObject
{
    Q_OBJECT
public:
    explicit Conversion(QObject *parent = nullptr);
    QBitArray HexToBin(QByteArray DataPacket);
    float BinQnmToDecimal(QBitArray bits);
    void convertData(QByteArray DataPacket);

public slots:
    void getDataForConversion(QString data);
signals:
    void sendConvertedData(float data);
private:
    TCPPackets *m_tcpPackets;
};

#endif // CONVERSION_H
```

DataModel.h



```
#ifndef DATAMODEL_H
#define DATAMODEL_H

#include <QObject>
#include <QAbstractTableModel>
#include <QList>
#include <QPointF>

class DataModel : public QAbstractTableModel
{
    Q_OBJECT
public:
    enum {
        ValueRole = Qt::DisplayRole
    };
    explicit DataModel(QObject *parent = nullptr);
    int rowCount(const QModelIndex &parent) const;
    int columnCount(const QModelIndex &parent) const;
    QVariant data(const QModelIndex &index, int role) const;
    QHash<int, QByteArray> roleNames() const;
    void addNewPoint(QPointF point);

private:
    QList <QPointF> m_points;

};

#endif // DATAMODEL_H
```