Project: Implementation of Communication Protocol on FPGA

System Design and Architecture

I. Introduction

In this project, I have implemented two communication protocols, **UART (Universal Asynchronous Receiver and Transmitter)** and **SPI (Serial Peripheral Interface)**, using an FPGA board and Arduino. The project works in the following way: data is sent from a computer terminal to a USB-to-UART adaptor, which transfers this data to the FPGA board through the UART protocol. After receiving the data, the FPGA board sends it further to an Arduino using the SPI protocol.

In this design, the FPGA acts as the **SPI master**, and the Arduino acts as the **SPI slave**. The Arduino then processes the data by incrementing it and sends it back to the FPGA through SPI. Once the FPGA receives the incremented data, it sends the result back to the terminal through the USB-UART adaptor. In the terminal, both the decimal and hexadecimal values of the received data can be observed.

For implementation, I have used **Quartus software** to design and synthesize the FPGA modules, and **ModelSim** for simulation of UART, SPI, and even I2C protocol. Along with the hardware design, I have also started creating a **web dashboard**, where the plan is to display real-time waveforms of the communication. Currently, the dashboard detects ports but the live waveform feature is still pending.

The goal of this project is to demonstrate how multiple protocols can work together in embedded systems, and how FPGA can act as a fast and reliable controller between devices.

II. Modular Design

I have designed the project in a **modular way** so that each part is separate and easy to maintain, extend, or reuse. The main modules are:

1. **UART Communication Module**
    - Handles communication between the PC terminal and FPGA through USB-UART adaptor.
    - Responsible for receiving data from the terminal and sending processed data back.
    - Works asynchronously, so it does not depend on a shared clock.
2. **SPI Communication Module**
    - Handles communication between FPGA (master) and Arduino (slave).

- Transfers the received data to Arduino, and also receives incremented data back.
- Synchronous protocol, works with a shared clock signal.

3. **FPGA Controller Module**
   - Coordinates UART and SPI communication.
   - Acts as the central hub: receives data from UART, forwards to Arduino via SPI, collects response, and sends it back to UART.
   - Written and implemented in Verilog using Quartus.
4. **Arduino Processing Module**
   - Arduino receives data from FPGA through SPI.
   - Increments the received data (example: if input is 5, it makes it 6).
   - Sends the incremented data back to FPGA through SPI.
   - A second serial port on Arduino also displays the received and processed data directly on another terminal window in both decimal and hexadecimal format.
5. **Web Dashboard Module**
   - A software-based module that runs on PC.
   - Currently detects available ports and is planned to show real-time waveforms of communication.
   - Future plan is to connect FPGA/Arduino serial output to this dashboard and visualize UART/SPI signals.

Flow:

- Terminal → USB-UART → FPGA → SPI → Arduino → back to FPGA → back to UART → Terminal.

- By dividing the project into modules, it becomes easier to test each part individually. For example, UART can be simulated in ModelSim before combining with SPI. Similarly, Arduino SPI code can be tested separately before connecting to FPGA. This modularity also makes the system more maintainable and reusable for future projects.

III. Technology Stack

The project uses both hardware and software tools. The technology stack is:

Hardware

- **FPGA Board (Cyclone II EP2C5T144)**: Used as the main controller. It is powerful for handling parallel data and protocol logic.

- **Arduino Board**: Used as SPI slave and for simple processing of data (increment operation).
- **USB-to-UART Adaptor (CH340)**: Connects FPGA UART pins to PC terminal through USB.
- **PC Terminal (Putty or similar software)**: To send and receive data via UART.

Software

- **Quartus II**: For writing Verilog code, synthesis, and programming FPGA.
- **ModelSim**: For simulation of UART, SPI, and I2C protocols. It helps to debug the design before uploading to FPGA.
- **Arduino IDE**: To write and upload SPI communication code to Arduino.
- **Web Dashboard (HTML/JavaScript)**: For future development of real-time waveform monitoring. Currently detects ports.

Justification of Choices

- **FPGA** is chosen because it provides parallel execution and precise timing control, which is important for communication protocols.
- **Arduino** is chosen because it is simple, low-cost, and integrates easily with SPI and UART.
- **UART** is used for communication with PC because it is a simple and widely supported protocol for serial communication.
- **SPI** is used for FPGA-Arduino communication because it is faster than UART and works well with master-slave configuration.
- **Quartus and ModelSim** are industry-standard tools for FPGA design and simulation, which ensures reliability.
- **Web dashboard** makes the system more interactive and useful for monitoring. It adds a modern software element to the hardware project.

IV. Scalability Plan

Scalability is important because the system may need to handle more data, more devices, or higher speeds in the future. My plan for scalability includes:

1. **Expanding Devices**
    - More SPI slaves can be added to the bus, such as sensors or additional Arduino boards.
    - FPGA can manage multiple slaves by using separate chip-select lines.
2. **Higher Data Handling**

- Increase UART baud rate for faster communication with PC.
- Optimize SPI clock speed for faster data transfer between FPGA and Arduino.
- Use buffering in FPGA to handle bursts of data.

3. **Adding More Protocols**
    - I have already simulated **I2C** in ModelSim, so in future, I can add I2C sensors or devices into the system.
    - This will make the project more flexible and useful for real-world applications.
4. **Web Dashboard Improvements**
    - Add real-time waveform monitoring by capturing signals from FPGA/Arduino.
    - Store communication logs in a database or cloud system for analysis.
    - Support multiple users and remote monitoring.
5. **Cost and Reliability Consideration**
    - FPGA and Arduino are low-cost platforms, making the system affordable.
    - Reliability can be improved by proper clock synchronization and error-checking mechanisms in communication.

This plan shows that the project is not just limited to one FPGA-Arduino setup but can grow into a bigger system with more devices, faster communication, and modern monitoring tools.

V. Conclusion

The project demonstrates how FPGA can act as a central controller between PC, Arduino, and communication protocols like UART and SPI. By designing the system in a modular way, each part (UART, SPI, FPGA control, Arduino, and dashboard) is independent and reusable. The technology stack chosen ensures compatibility, performance, and ease of development.

The scalability plan shows that the system can handle more devices, higher speeds, and new features like real-time monitoring. The use of both hardware and software makes the project practical and also opens future possibilities like cloud integration or industrial applications.

In future work, I plan to complete the real-time waveform display on the web dashboard, integrate I2C fully into the design, and optimize data transfer speed. This project lays the foundation for building a complete embedded communication system that is modular, scalable, and technically strong.