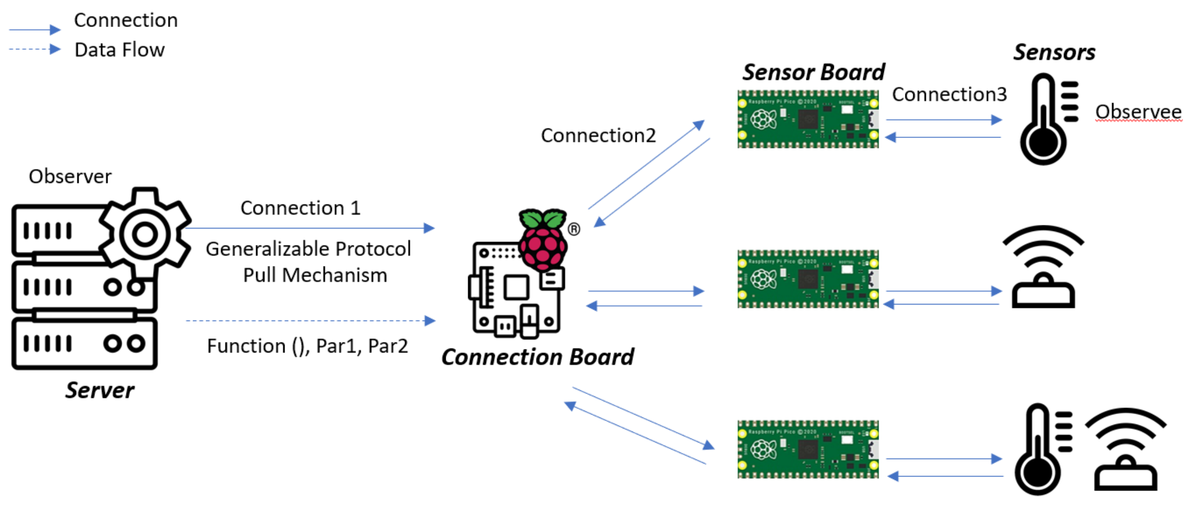
Project Title: *Sensor Data Aggregation and Visualization System*

Project Overview:

Motivation: In the context of environmental monitoring, the ability to gather, consolidate, and present sensor data is of paramount importance. This project aims to develop a comprehensive software system that collects data from a network of sensors, summarizes this information, and provides a user-friendly graphical user interface (GUI) on a server for end-users.

Project Scope: The project involves the development of a comprehensive software system with two primary data flow types: observation and configuration. The system architecture necessitates that the server be capable of configuring the attached sensors through the same I2C connection it uses for observation. This entails the server issuing "reading" requests in tandem with "edge processing" functions that are executed within the sensor board, denoted as the pico. To ensure efficient power management and battery conservation, an Observer Design Pattern is implemented, where the server pulls sensor data instead of pushing it.



Key System Components:

- **I2C Communication Protocol Development:** The project requires the creation of a customized I2C communication protocol to facilitate seamless data exchange between the server and the pico. This protocol encompasses serialized commands that the pico can decode to interact with specific sensors, enabling both read and write operations.

- **Edge Processing on Pico:** The pico device plays a pivotal role in executing various data processing functions locally, such as calculating averages, identifying maximum and minimum values, and generating histograms, among others. These operations are conducted on a set of collected samples.

- **Results Transmission:** Following data processing, the pico transmits the processed results and any potential errors back to the server for further analysis and visualization.

MAJOR QUALITY CONCERN:

Universal Communication Protocol for Interconnected Boards: In the realm of interconnected electronic boards, the need for a versatile and efficient communication protocol is paramount. This project aims to develop a universal communication protocol that transcends the limitations of specific communication mediums. The protocol will maintain a minimal packet size while affording flexibility to accommodate a wide range of features. It is designed to facilitate seamless communication among boards with minimal information sharing, enabling interoperability across communication channels such as SIP, UART, I2C, BLE, and Ethernet (ETH). https://www.oodesign.com/adapter-pattern

- **Medium Agnosticism:** The protocol is designed to be medium-agnostic, ensuring it can be seamlessly implemented across diverse communication channels, including SIP, UART, I2C, BLE, and Ethernet.

- **Minimal Packet Size:** The protocol is optimized for efficiency, minimizing packet size to reduce data transmission overhead while maintaining robust functionality.

- **Flexibility:** It offers the flexibility to accommodate various features and functionalities, allowing for extensibility as communication requirements evolve.

- **Minimal Information Sharing:** Boards utilizing this protocol will share only essential information required for communication, enhancing security and privacy.

Key Communication Channels:

- **Serial Interface Protocol (SIP):** The protocol can be implemented on serial interfaces, ensuring compatibility with traditional communication methods.

- **Universal Asynchronous Receiver-Transmitter (UART):** UART-based communication is supported, enabling serial data exchange between boards.

- **Inter-Integrated Circuit (I2C):** I2C communication is facilitated, allowing for interconnection and data exchange between boards over short distances.

- **Bluetooth Low Energy (BLE):** BLE support enables communication between boards using low-power wireless technology.

- **Ethernet (ETH):** Ethernet-based communication is accommodated, suitable for scenarios requiring high-speed data transfer.

Quality Goals:

- Develop a universal communication protocol for electronic boards.

- Ensure medium agnosticism, enabling implementation on various communication channels.

- Minimize packet size while maintaining functionality.

- Support minimal information sharing for enhanced security and privacy.

Quality Concerns and Architectural Considerations:

- **Efficiency:** The system must ensure efficient data transmission between the server, Raspberry Pi, and the sensors to minimize latency and resource utilization. The protocol optimizes data transmission efficiency, reducing latency and resource utilization.

- **Interoperability:** An Adapter Design Pattern is implemented to enable interchangeable sensors with varying communication protocols, ensuring compatibility with a wide range of environmental sensors. The protocol ensures seamless interoperability across different communication mediums, fostering compatibility among diverse boards and devices.

- **Availability:** The system incorporates Watchdog or Ping architectural tactics to ensure the continuous availability of critical components, mitigating potential system failures.

- **Scalability:** Scalability is a key concern, with a focus on measuring the maximum frequency in relation to the length of the transmitted message. This ensures that the system can handle increasing data loads without performance degradation.

Protocol Transformation Steps and Examples:

- 1. **Understand the Differences:**
    - SIP is a serial communication protocol, while I2C is a synchronous, multi-master, multi-slave, packet-switched protocol.
    - SIP typically involves transmitting data bit by bit, whereas I2C uses a packet-based approach with a *start condition*, *address*, *data*, and *stop condition*.
- 2. **Identify Data Mapping:** Determine how the data in the SIP package maps to the I2C package. You'll need to know which parts of the SIP data correspond to the address, command, and payload in the I2C package.
- 3. **Create an I2C Frame:** Build an I2C frame that includes the following components:
    - ***Start Condition:*** A unique condition signaling the start of communication.
    - ***7-Bit or 10-Bit Slave Address:*** The address of the I2C device to which the data is being sent.
    - ***Data Payload:*** The actual data you want to transmit.
    - ***Stop Condition:*** A unique condition signaling the end of communication.
- 4. **Address Mapping:** Map the address from the SIP package to the 7-bit or 10-bit I2C slave address. Ensure it matches the addressing scheme used by the target I2C device.
- 5. **Data Mapping:** Map the data from the SIP package to the data payload in the I2C frame. Ensure it's formatted correctly according to the I2C device's requirements.
- 6. **Timing and Synchronization:** Ensure that the timing of data transmission aligns with the I2C clock speed and timing constraints. I2C is synchronized, so data bits should be sent/received in accordance with the clock signal.
- 7. **Implement I2C Protocol:** Use an appropriate programming library or I2C hardware module on your microcontroller or platform to send the I2C package with the mapped data.
- 8. **Error Handling:** Consider error-checking mechanisms, such as checksums or CRC, if necessary, to ensure data integrity during the transformation.
- 9. **Test and Debug:** Thoroughly test the transformed communication to ensure it works as expected. Debug any issues that may arise during the transformation process.

EXAMPLE_1: SIP→ I2C

SIP Packet Format:
- Sensor ID (1 byte)
- Command (1 byte)
- Data Payload (variable length)

I2C Packet Format:
- Start Condition
- 7-Bit I2C Slave Address
- Command Byte
- Data Payload (if applicable)
- Stop Condition

Example Transformation: Suppose we have the following SIP packet:
- Sensor ID: 0x12
- Command: 0x34
- Data Payload: 0x5678

We want to send this information to an I2C sensor with a 7-bit address of 0x50.

Transformation Steps:

- **Map Address:** In I2C, we need to shift the 7-bit address left by 1 bit and set the least significant bit (LSB) to 0 for write operation. So, the I2C slave address becomes 0xA0 (0x50 shifted left by 1 bit).

- **Create I2C Frame:**
  - *Start Condition:* Send the start condition to initiate communication.
  - *Slave Address:* Send the 7-bit I2C slave address (0xA0) indicating the device we want to communicate with.
  - *Command Byte:* Send the command byte (0x34).
  - *Data Payload:* If applicable, send the data payload (0x5678).
  - *Stop Condition:* Finally, send the stop condition to conclude communication.

I2C Frame Sent Over the Bus:

Start Condition --> [0xA0 (Slave Address)] --> [0x34 (Command)] --> [0x56 (Data)] --> [0x78 (Data)] --> Stop Condition

This transformed packet is now compatible with the I2C protocol and can be sent over the I2C bus to communicate with the target sensor. The sensor at address 0x50 will receive the command (0x34) and, if necessary, the data payload (0x5678) for further processing or response.

Example_2: UART→ I2C

UART Packet Format:

- Start Byte: A special character (e.g., 0x01) indicating the start of a packet.
- Address Byte: The address of the I2C device (1 byte).
- Command Byte: The command to be executed by the I2C device (1 byte).
- Data Payload: The actual data to be sent (variable length).
- Stop Byte: A special character (e.g., 0x04) indicating the end of a packet.

I2C Packet Format:

- Start Condition
- 7-Bit I2C Slave Address
- Command Byte
- Data Payload (if applicable)
- Stop Condition

Example Transformation: Suppose we have the following UART-like packet:

- Start Byte: 0x01
- Address Byte: 0x50
- Command Byte: 0x34
- Data Payload: 0x5678
- Stop Byte: 0x04

We want to send this information to an I2C sensor with a 7-bit address of 0x25.

Transformation Steps:

- **Map Address:** In I2C, the address should be 7 bits without the R/W bit (read/write bit). So, the I2C slave address becomes 0x25.
- **Create I2C Frame:**
  - *Start Condition:* Send the start condition to initiate communication.
  - *Slave Address:* Send the 7-bit I2C slave address (0x25) indicating the device we want to communicate with.
  - Command Byte: Send the command byte (0x34).
  - *Data Payload:* If applicable, send the data payload (0x5678).
  - *Stop Condition:* Finally, send the stop condition to conclude communication.

I2C Frame Sent Over the Bus:

Start Condition --> [0x25 (Slave Address)] --> [0x34 (Command)] --> [0x56 (Data)] --> [0x78 (Data)] --> Stop Condition

This transformed I2C packet is now compatible with the I2C protocol and can be sent over the I2C bus to communicate with the target sensor. The sensor at address 0x25 will receive the command (0x34) and, if necessary, the data payload (0x5678) for further processing or response.

Accelerometer sensor:
https://www.amazon.com/Pre-Soldered-Acceleration-Compatible-Tilt-Sensing-Accelerometer/dp/B0BXWHTXWT

Temperature Sensor:
https://www.amazon.com/Adafruit-MCP9808-Accuracy-Temperature-Breakout/dp/B00OKCQX96/ref=sr_1_1?crid=1ELZTE2T1RGI4&keywords=temperature+sensor+adafruit&qid=1694534368&s=industrial&sprefix=temperature+sensor+adafruite%2Cindustrial%2C104&sr=1-1

Raspberry Pi pico as sensor board:
https://www.amazon.com/seeed-studio-Raspberry-Microcontroller-Dual-core/dp/B08TQSDP28/ref=sr_1_2_sspa?keywords=raspberry%2Bpi%2Bpico&qid=1694534404&sr=8-2-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&th=1

Raspberry Pi for communication board:
https://www.amazon.com/Raspberry-Pi-Model-Desktop-Linux/dp/B00T2U7R7I/ref=sxts_b2b_sx_fused_v3_desktop_ref-tab-0?content-id=amzn1.sym.97762c05-7545-47e0-ae5c-1110ba2791f0%3Aamzn1.sym.97762c05-7545-47e0-ae5c-1110ba2791f0&crid=307G9HUQSGOY5&cv_ct_cx=raspberry%2Bpi&keywords=raspberry%2Bpi&pd_rd_i=B00T2U7R7I&pd_rd_r=6453395a-944a-4cfe-bafa-1a386939125d&pd_rd_w=7d0UF&pd_rd_wg=BwTE8&pf_rd_p=97762c05-7545-47e0-ae5c-1110ba2791f0&pf_rd_r=WA7TDT3XA7T0B40HXYRZ&qid=1694534570&sbo=RZvfv%2F%2FHxDF%2BO5021pAnSA%3D%3D&sprefix=raspberry%2Bpi%2Caps%2C172&sr=1-7-965fba24-1eed-4536-936e-b447f98a83bc&th=1