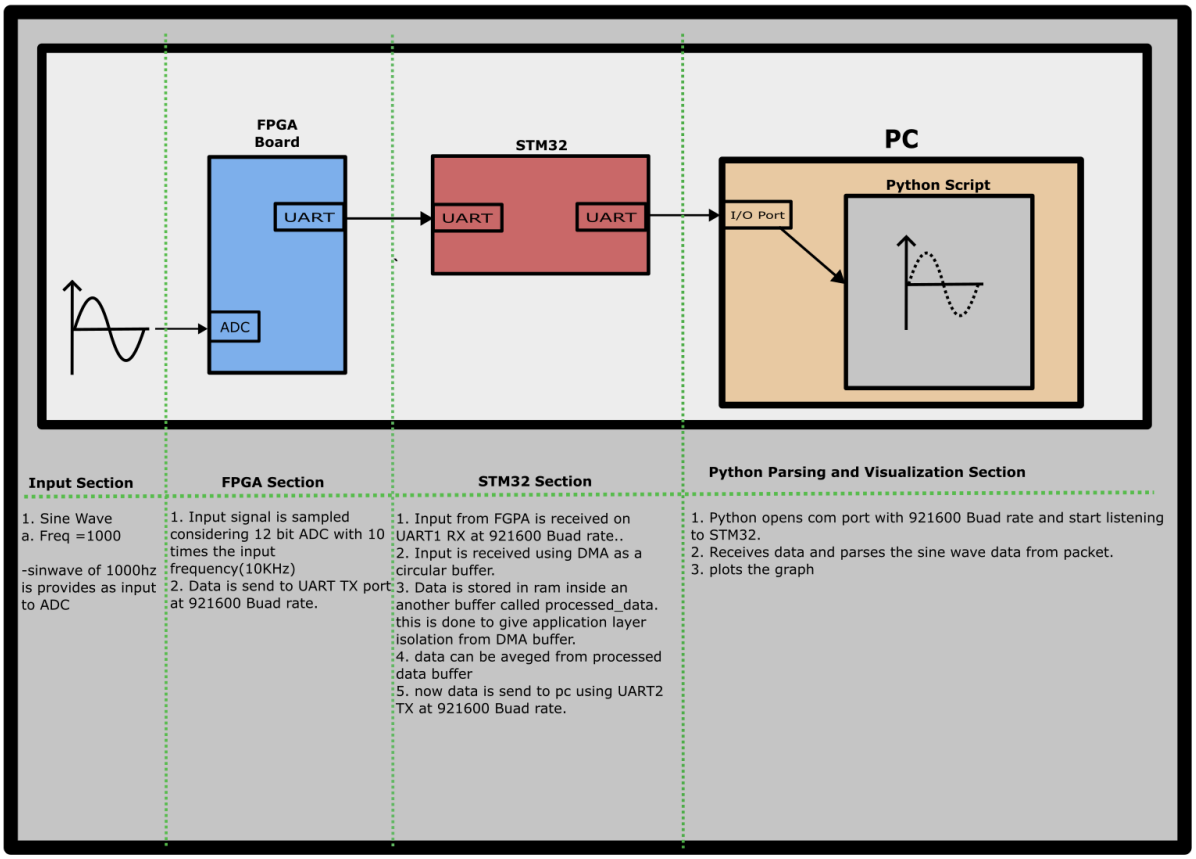
Deliverables

1. **Block diagram and schematic (PDF or image)**



2. **HDL code, STM32 firmware (C files), and Python script (with sample data).**
    a. **HDL Code**

    I have not yet had the opportunity to work hands-on with FPGA. To begin learning, I recently purchased a Verilog book about two weeks ago and have now taken this as a chance to start learning on real hardware. I've ordered a Digilent Basys 3 Artix-7 board for my own practice.As a first step, I tried building a simple data logger in Vivado, where a sine wave is generated and the output is logged into a simulation file. I then used Python to plot the results.I know this is just a small beginning, but I am genuinely eager to grow in this skill. I am committed to learning FPGA development steadily, and I look forward to building stronger capabilities with continued practice on actual hardware.

    b. **Stm32 codes are stored inside STM folder**
        i. Please refer the project folder.

    c. **Python script stored inside python folder**
        i. **Plotter_nodemcu2.py**
            1. This file helps to validate the input to STM32

2. File parses the packet(AA,length,data,checksum) and plots the data.

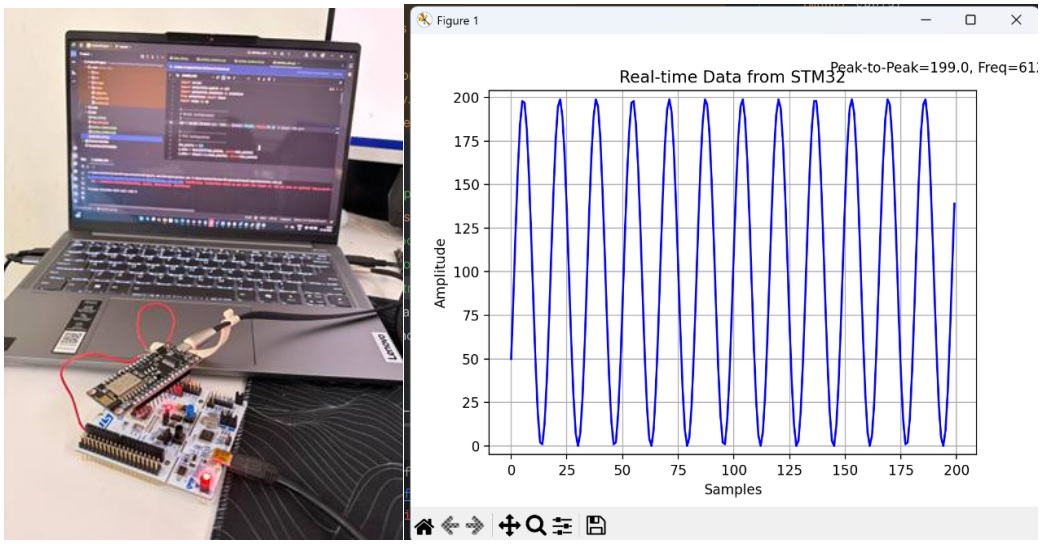
ii. **Plotter_stm.py**

1. This file helps to validate the output from STM32
2. File parses the packet (AA,length,data,checksum) and plots the data.

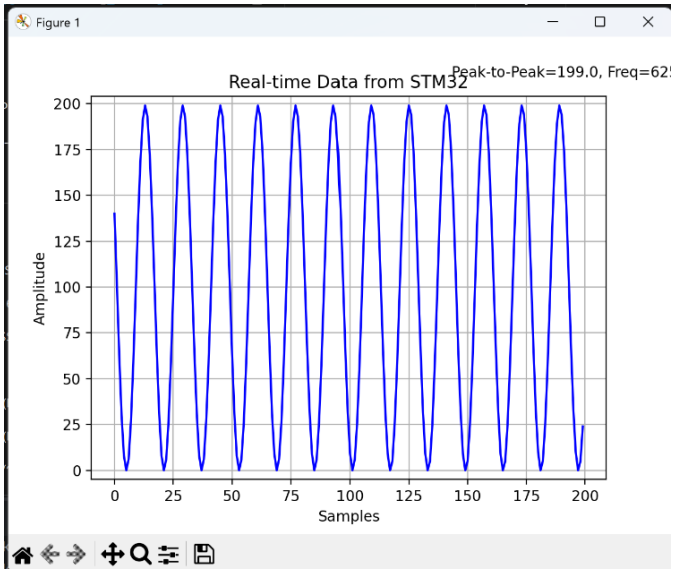
d. **Sample data is sin wave generated from nodemcu.**

3. **Simulation waveforms/screenshots and brief synthesis notes (e.g., resource usage)**

a. Validating input to stm32 using python script(sorry that title says from stm32 its actually to stm32). It's the same code we just change com port for nodemcu and stm32 to validate the data stream through uart



b. **Validating the simulation output at stm32 output**



4. **Short report**

a. Design Choices and Rationale

i. **FPGA Usage**
   1. Did not have access to an actual FPGA board during the project.
   2. As a beginner, I focused on learning simulation first.
   3. Created a signal in simulation and logged it to a file.
   4. Used Python to read the logged data and plot waveforms for validation.
   5. Though not the full FPGA demo, this helped me build a foundation.
   6. I have since committed to catching up by purchasing an FPGA for hands-on practice.

ii. **Alternative Hardware Setup**
   1. **NodeMCU** was used as a replacement input source:
      a. It was already available and very low cost.
      b. Provides a simple UART stream suitable for testing.
      c. Well supported with resources, making it easy to implement quickly.
   2. **STM32 Microcontroller** was chosen for data reception:
      a. Already had an STM32 board at hand.
      b. Familiar with STM32 from past experience.
      c. Powerful and widely used 32-bit MCU, ideal for embedded data handling.

iii. **Data Transfer Choice: DMA vs. Interrupts**
   1. **DMA (Direct Memory Access):**
      a. Chosen method for transferring UART data.
      b. Advantage: Frees the CPU to perform other tasks while DMA handles the data movement.
      c. Enables real-time post-processing or validation without CPU bottlenecks.
   2. **Interrupts (Not chosen):**
      a. Would consume significant CPU resources, especially at higher data rates.
      b. In large continuous data streams, interrupts lead to overhead and latency.
   3. **Conclusion:** DMA is more efficient and scalable for continuous data logging applications.

b. Challenges encountered and how you addressed them.

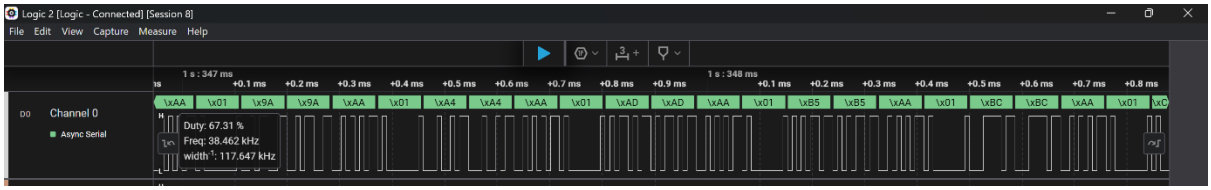
i. I has issues with uart stream in the beginning used logic analyzer to probe the data and find the issue with was jumper cable issue which had a brken connection. So I soldered it permenetly

ii. I was new to FPGA so it took time to read book and watch youtube video to get an idea. Finally decided I will learn in long run by committing to purchase a board. I have ordered my board and will look in to it.

iii. It was faced some challenge working from home under limited setup. But it was fun. I managed to scrap components from my hobby projects and gets things going.

c. **Results**

i. I successfully probed the data at both the **STM32 input** and the **UART output stream**, validating the packets using a **logic analyzer** and confirming correctness with **Python plots**.



d.

i. Baud Rate Limitation
   1. At 115200 baud, throughput is roughly 14,400 bytes/sec.
   2. Increasing the baud rate directly increases data transfer capacity.
   3. However, the upper limit is constrained by the maximum supported baud rate of the STM32 UART hardware.

ii. Trade-offs
   1. Higher Baud Rates:
      a. Pros: Higher throughput, faster transfers.
      b. Cons: Increased susceptibility to noise and data corruption, especially over longer cables.
   2. Lower Baud Rates:
      a. Pros: More reliable transmission, lower error rate.
      b. Cons: Limited throughput, slower data transfer.

iii. Suggestions for Improvement
   1. Optimize Baud Rate: Select the highest stable baud rate supported by both devices and environment.
   2. Error Handling: Implement CRC or checksum validation to ensure data integrity at higher speeds.
   3. Don't forget to set DMA circular buffer for uart.

iv. For the demonstration, I captured 16 samples per cycle. At a baud rate of 921,600, the channel supports roughly 92,160 bites per second = 115200 bytes per second. Each packet is 4 bytes × 16 samples = 64 bytes, which consumes only about 0.055% of the available UART bandwidth. This demonstrates that the system is highly under-utilized, leaving ample margin to increase the sampling rate or packet size as required.