An Anime Figure Robot Head for Cyber-Physical Systems Research

Yufeng Chi*
chiyufeng@berkeley.edu

Jingyi Xu*
jingyixu@berkeley.edu

***Abstract*—This project presents the development of an anime-inspired robotic head designed for cyber-physical systems research, integrating advanced computational techniques, aesthetic considerations, and mechanical engineering. The system features a multi-platform architecture that combines STM32 microcontrollers, FPGA-based gesture recognition, and neural networks deployed on edge AI devices. By coordinating eye and head movements and dynamically responding to user interactions, the robot achieves lifelike and engaging interactions. The implementation of gesture recognition through machine learning, along with finite state machine-based behavior control, enhances the robot's capability for natural human-robot interactions. The mechanical structure, created using OnShape and Blender, supports versatile movement and expressive gestures. This work not only addresses challenges like the uncanny valley effect but also establishes a foundation for future advancements in humanoid robotic systems, with potential applications in entertainment, education, and human-computer interaction.**

I. Introduction

The design and functionality of humanoid robots have advanced significantly in recent years, yet achieving natural and engaging human-robot interactions remains a considerable challenge. A key obstacle is the "uncanny valley" effect [6, 7], where robots that closely resemble humans but display subtle imperfections in appearance or behavior evoke discomfort or unease in observers. Addressing this phenomenon requires a combination of stylized aesthetic design [5] and smooth, responsive motion control to enhance user perception and interaction quality.

Al-Sada et al. approaches this challenge by implementing a humanoid robot that incorporates anime-inspired behaviors and interactions, utilizing teleoperation and imitation-learned control policies. However, their work does not fully explore the computational requirements or input modalities needed for a seamless user experience.

Building on these concepts, this work introduces an anime-styled robotic head equipped with multiple computing components, shown in Figure 1, designed to express emotions through coordinated eye and head movements while dynamically responding to user interactions. This platform serves as a foundation for exploring advanced motion control strategies and optimizing workload scheduling for multitenant systems, ultimately contributing to more fluid and lifelike human-robot interactions. The codebase can be accessed from https://github.com/T-K-233/NAI-Head.

*These authors contributed equally to this work

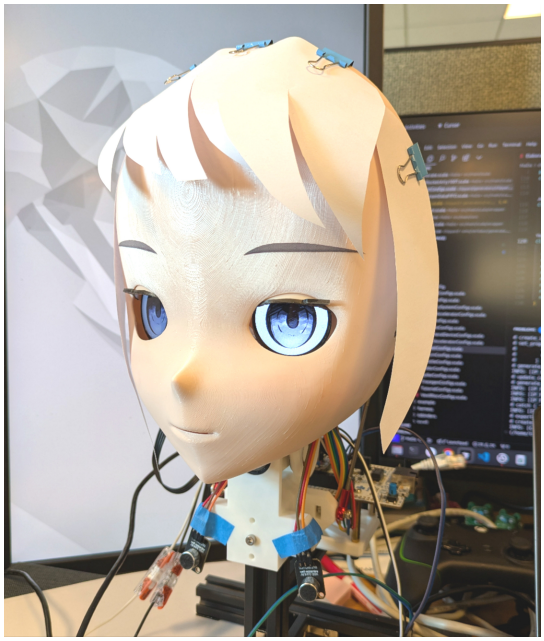


Fig. 1: The anime-styled robotic head with ability to express motions via eye and head movements.

II. The Robot Platform

The system is composed of three compute platforms. The host computer reads in camera frames from the camera and performs the front-end detection of the user gesture. The detected hand keypoints will then be sent to the RISC-V processor running on Arty FPGA for classification and detects a motion label from the landscape. Lastly, the user gesture is transmitted to the STM32 microcontroller, which controls both the microphones, the TFT LCD screens, and the various servo actuators. The system are connected together with Ethernet interface under a same local subnet, and the data packets are transmitted with UDP frames. In Section III we introduce the workload and static scheduling on the STM32 microcontroller; in Section IV we introduce the neural network running on baremetal environment; and in Section VII we introduce the software running on the host PC.

The mechanical structure of the robot is designed with OnShape and Blender, and manufactured with 3D printing fabrication method. More detail on this is covered in Section VIII

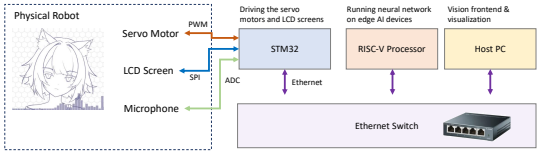


Fig. 2: The compute system is composed of three modules. The STM32 controls the physical robot and is responsible for realtime workloads; the RISC-V processor runs the neural network; and the host PC runs the first part of the vision processing pipeline and the visualization.

III. The STM32 Microcontroller

The STM32 microcontroller is responsible for real-time sensing and control of the robot's input/output (I/O) operations.

Given the platform's resource constraints and the strict deadlines associated with multiple real-time tasks, a feasible static schedule is crucial to ensure timely and successful program execution.

A. Processor Selection

The STM32 product family offers a wide range of options for embedded processing. The STM32F429ZI microcontroller unit (MCU) was selected based on the following criteria:

- Availability of the Nucleo development board for the selected MCU.
- Availability of required I/O ports, particularly the Ethernet port.
- Availability of resources, such as tutorials and example projects.
- Cost-effectiveness of the development board.
- Compact size and lightweight design.

B. Workload Description

The STM32 controller executes several tasks, prioritized based on their real-time requirements.

1) Audio Sampling

The audio sampling task continuously reads microphone input using the analog-to-digital converter (ADC) at a 25 kHz sampling rate for short durations. While the ADC acquires new samples, previously captured data is transferred to SRAM via direct memory access (DMA) engines.

The microphones are mounted on the same plane, roughly 100 mm apart. Two example signals from the microphone array are shown in Figure 3. A cross-correlation algorithm is then applied to the sampled data from the left and right microphones to determine the time delay corresponding to the peak correlation value. This time delay indicates the angle of the sound source.

2) LCD Control

The LCD driver is another time-sensitive workload. Two TFT LCDs, each driven by the GC9A01A IC, are connected to the STM32 via SPI buses. During each frame update, the driver transmits the coordinates of the first pixel along with the image dimensions to the GC9A01A control registers. It then streams pixel data to the display buffer.

(a) Sound coming from middle (b) Sound coming from left

Fig. 3: The analog signals from the microphone. The signals are centered around 1.65V, and the amplitude corresponds to the amplitude of the sound. The phase shift between the two signals roughly translates to the angle of the sound source from the plane of the two microphones.

(a) SPI transmit in polling mode (b) SPI transmit in DMA mode

Fig. 4: The CPU utilization when transmitting SPI data in polling mode and in DMA mode. (a) The yellow line shows the CPU's state when transmitting on SPI4, where a high level indicates that the CPU actively transmitting, and a low level indicates the CPU is available for other tasks, and the magenta line indiates the CPU state on SPI5. It can be observed that the two SPI transaction takes up entire CPU cycles, which prevents other tasks from being executed. (b) The yellow line shows the begin of a SPI transaction with DMA, and the magenta line shows the corresponding interrupt that configures the CS pin, indicating the end of a transaction.

Each pixel is encoded in a 565 format, combining 5-bit red, 6-bit green, and 5-bit blue values into two bytes. Given the 240x240 resolution of the LCD, each frame requires transferring 115,200 bytes of data. To reduce CPU load, DMA is employed to handle data transfer, allowing the CPU to perform other tasks concurrently. Figure 4 shows a comparision between using blocking polling transfer and DMA transfer.

3) Servo Control

To control the robot's five servos, five PWM signals are generated with a 50 Hz period and pulse durations ranging from 500 µs to 2500 µs. The hardware timers on the STM32 are used to generate these PWM signals, minimizing CPU involvement. The CPU only updates the Capture/Compare Register (CCR) values to adjust pulse durations.

4) UDP Communication

The final task involves handling UDP communication between the STM32 and other components of the robot system. The implementation leverages the STM32CubeIDE libraries, which support multiple layers of the OSI model. Specifically:

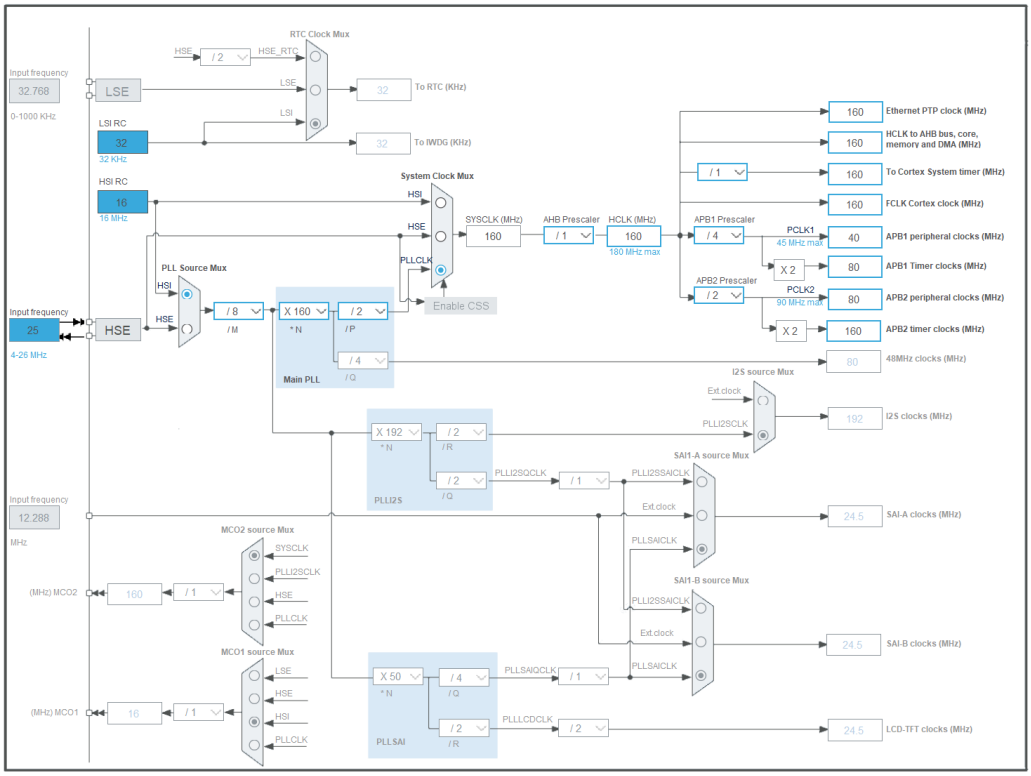


Fig. 5: The clock tree of the STM32F429ZI. System clock is set to 160 MHz, generated from the main PLL. APB1 and APB2 peripherals are set to their maximum possible clock frequencies, which is 40 MHz and 80 MHz respectively.

- Physical and Data Link Layers: Managed using the onboard Ethernet (ETH) peripheral.
- Network Layer: Handled by the Lightweight IP (LWIP) stack.
- Transport Layer: Implemented using the example code from Controllerstech.

This modular approach ensures efficient and reliable UDP communication while meeting the real-time constraints of the system.

C. MCU Resource Allocation

Since none of the tasks requires high precision clocks, the clock source of the MCU is configured to be the hight-speed internal RC oscillator (HSI). The system clock is then generated from the on-chip PLL. We select 160 Mhz to be the system clock as it is sufficiently fast, easily dividable into other slower clock domains, and provides enough headroom to the absolute maximum 180 MHz. The resulting clock tree is shown in Figure 5.

Timer 6 and Timer 7 are used to generate interrupt events for SPI and PWM updates. Timer 6 is configured to trigger an interrupt at 10 Hz interval, while Timer 7 is configured to trigger at 50 Hz.

Timer 1 and Timer 4 are used to generate the PWM signal for the five servo motors. They are configured at the start of the application program, and the values are updated in the interrupt handler of Timer 7.

Timer 2 is used as the trigger source for ADC sampling. When update event happens, it generates a signal on the trigger output (TRGO) pin, which will send to the ADC and starts the sampling and conversion. The ADC will then communicate with the DMA engine to perform a predefined number of conversions, and writes the data to a target SRAM location via DMA.

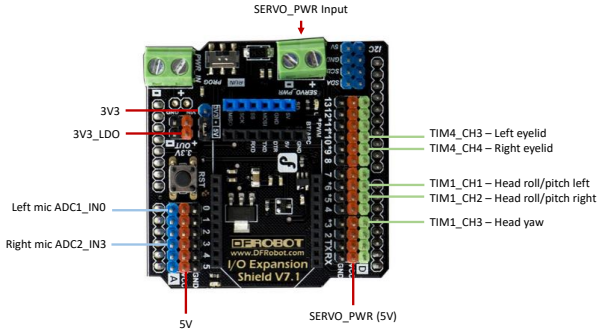
SPI4 and SPI5 are used to drive the TFT LCD screens. For the control commands, they are operating in the blocking polling and in 8-bit mode. After transmitting the command, it is updated to operate in non-blocking DMA mode and in 16-bit transfer mode. Since the chip select (CS) pins are controlled by software, an interrupt handler is set up to write the CS pins high after the data transfer.

Ethernet and the corresponding software library for UDP logic is running entirely in the main thread.

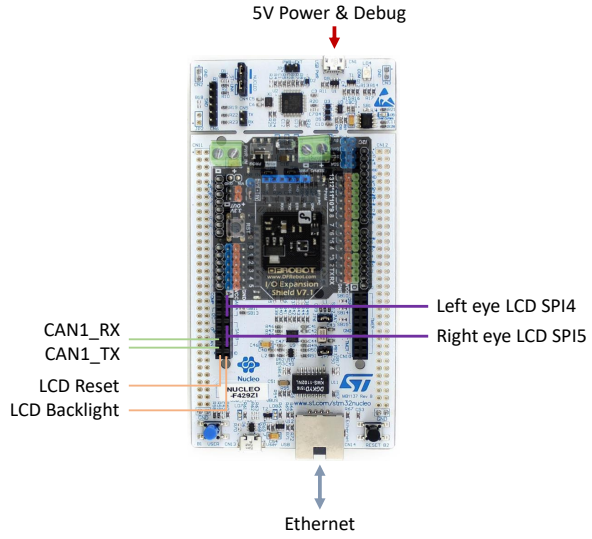
The IOs are planned with the wire harnessing in mind. The resulting pinmap is shown in Figure 6.

IV. The Vision Pipeline

The gesture recognition pipelines goes through three stages. First, when users pose, the camera video input is captured and pre-processed. Next, we use mediapipe library to extract the hand keypoints from the image. Lastly, the hand keypoint features are fed into a MLP network to classify the input hand gesture into one of hi, ok, no, point, heart. The output is sent over to STM32 to drive robot behaviors.



(a) Wiring connection on the DFRobot IO Expansion Shield V7.1



(b) Wiring connection of the STM32 Nucleo board

Fig. 6: Wiring connection from the STM32 and its expansion board to the peripheral devices. The mapping is designed to group pins with similar functions together.

The model's architecture is shown in Figure 7. It takes the 21 hand landmarks as input and runs through 3 hidden layers and outputs probabilities for each of the 5 labels, the highest of which selected as prediction label. This model is designed to be lightweight to be deployed on edge computing platforms including K230 and soft RISC-V core on FPGA. To train this model, we collected our custom dataset of 5000 samples. We trained the model from scratch and it achieves 97.2% accuracy.

To port the neural network to embedded runtime, we first implemented the C runtime and syscall handlers (libgloss layer) in Baremetal-IDE. In addition, we built the Baremetal-NN library to convert the pytorch model to embedded RISC-V environment. We successfully ported our custom pytorch model to K230 chip. We also generated a corresponding RISC-V RV32IMAFC TinyRocket SoC design on FPGA, leveraging the Ethernet controller IPs provided by Xilinx to perform UDP communication.

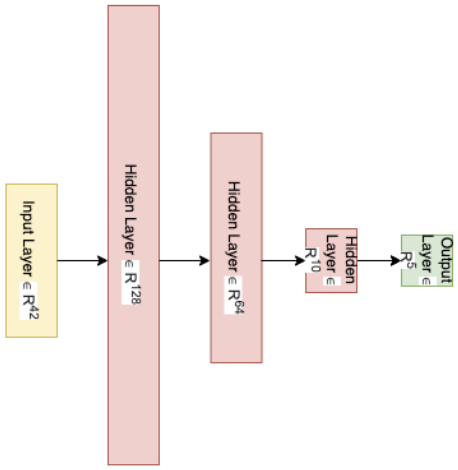


Fig. 7: Architecture of hand gesture recognition Neural Network

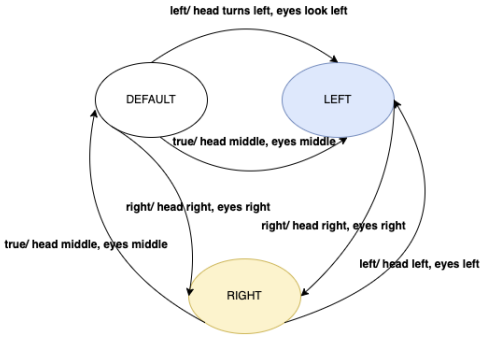


Fig. 8: FSM capturing the robot's reaction with respect to sound input(from left or right).

V. Robot Finite State Machine

We define the Robot's behavior using two finite state machines. The first reacts to input sounds. Figure 8 shows that the robot turns its head and eyes to the direction of sound input.

The robot's reaction to user's hand gesture is represented as a one-state Mealy Machines, where the output solely depends on the input. The input/output relation is summarized in the lookup table in Table I.

gesture	eyes	head
Hi	enlarged	tilt
Ok	open	nod
No	closed	shake
Point	wink	tilt
Heart	heart eyes	stay still

TABLE I: The robot's behavior w.r.t hand gesture inputs.

VI. Embedded Runtime

Since we are in a baremetal RISC-V environment, we need to construct the C runtime from the ground up. We build the glossy library, our own implementation of the libgloss layer, to provide the basic startup and C runtime assembly program, as well as the basic implementation of system call routines.

On bootup, the program counter (PC) of the Rocket core is pointed at BootROM, located at **0x00010000**. Then, the BootROM code will guide the PC to perform basic initializations, and jumps to the base of the on-chip SRAM region located at **0x80000000**. There, it enters our start up script "crt0.S", which initializes the system state by setting general-purpose registers, floating-point unit (if available), global pointer, thread-local storage, stack pointer, and memory segments (data, BSS). It then invokes global constructors, calls the main function for the primary core, and loops indefinitely after program termination, while secondary cores follow a similar initialization path to run __main.

A custom linker script is also created to instruct linker to put sections to corresponding locations. The mapping is shown in Figure 9.

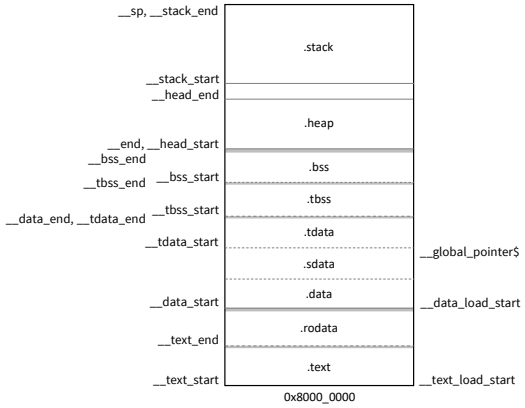


Fig. 9: Memory map specified by the linker script. The map starts at SRAM base address at **0x80000000**, and follows the convention of text, data, heap, and stack layout. The left side shows the symbols pointing to logical address, and the right side shows the symbols with physical address. The gray bar shows the different alignment of sections.

VII. Host PC Software

The host PC runs the vision frontend software, serving as an additional interface to interact with the system. Live video inputs are processed and displayed on the screen, with hand keypoints highlighted. This provides users with visual feedback on the captured scene and the recognized gestures.

Additionally, a visualization software is integrated to the system to sync actions from the STM32. VTubeStudio [4] is used as the software platform, which provides high range of motions and can build one-to-one motion mapping with the robot movements. A Python library is implemented to serve as the streaming bridge between the UDP communication used in the robot system and the WebSocket used to communicate with VTubeStudio APIs.

Fig. 10: On the host PC, VTubeStudio is used to provide a visualization of the robot state. The state is received from the STM32 via UDP, is passed through the custom UDP to Websocket bridge in Python, and sent to the software through its API.

VIII. Mechanical Design

We design the mechanical structure of the robot to support roll, pitch, and yaw movement on the neck, and blinking on the eyelid. The internal structure is designed with trial and error. Several design tricks are employed to improve the efficiency of the design flow and the reliability of the resulting structure.

A. Organic Shape Modeling

We generate the 3D model of the anime-styled face from VRoid Studio, which supports exporting to Blender via the vrm format.

B. Harnessing

Due to the limited spacing in the eye socket, it is hard to connect normal signal cables to the TFT LCD screen. Instead, we sourced the LCD screen modules that support connection via ribbon cables. We then adapt the ribbon cables back to the 2.54 mm jumper cables with a breakout board. This improvement greatly increases the reliability of the SPI signal connection, allowing us to run SPI up to 20 MHz

C. IGES File Format

We used Onshape and Blender together to design the organic shape of the robot face and the engineering structure that supports the components. After experimenting with the several 3D file format supported by these softwares, the Initial Graphics Exchange Specification (IGES) format [8] is selected due to its capability of expressing complex planes in Onshape in an editable fashion.

IX. Conclusion

In conclusion, this project demonstrates a comprehensive approach to designing an anime-inspired robotic head for

cyber-physical systems research, integrating aesthetics, mechanics, and advanced computational techniques. By leveraging a multi-platform system architecture and employing innovative workload management strategies, the robot achieves lifelike interactions and real-time responsiveness. The integration of gesture recognition and finite state machine-based behavior control highlights the potential for expanding human-robot interaction capabilities. Future work could explore the scalability of the system, the incorporation of additional sensory modalities, and further optimization of the neural network pipeline to enhance performance and adaptability. This endeavor not only bridges the gap between technology and creativity but also sets a foundation for advancing user-friendly humanoid robotic systems.

ACKNOWLEDGMENT

We would like to acknowledge BlueArrow for providing the servo motors used in the robot platform. We would also like to acknowledge "SiHaiMoYu" on Bilibili for kindly providing the Live2D model.

REFERENCES

[1] Mohammed Al-Sada, Pin-Chu Yang, Chang Chieh Chiu, Tito Pradhono Tomo, Mhd Yamen Saraiji, Tetsuya Ogata, and Tatsuo Nakajima. From anime to reality: Embodying an anime character as a humanoid robot. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–5, 2021.

[2] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40(4):10–21, 2020.

[3] Controllerstech. Stm32 ethernet - udp server, 2024. URL https://controllerstech.com/stm32-ethenret-2-udp-server/. Accessed: 2024-06-16.

[4] Denchisoft. Denchisoft official website. https://denchisoft.com/. Accessed: 2024-12-18.

[5] Tom Geller. Overcoming the uncanny valley. *IEEE computer graphics and applications*, 28(4):11–17, 2008.

[6] Roger K Moore. A bayesian explanation of the 'uncanny valley'effect and related psychological phenomena. *Scientific reports*, 2(1):864, 2012.

[7] Jun'ichiro Seyama and Ruth S Nagayama. The uncanny valley: Effect of realism on the impression of artificial human faces. *Presence*, 16(4):337–351, 2007.

[8] Brad Smith and Joan Wellington. Initial graphics exchange. *Specification (IGES), Version*, 3, 1986.

APPENDIX A
FPGA COMPUTE

To generate a RISC-V SoC design on FPGA, we leveraged Chipyard [2], an agile RISC-V SoC design framework. We used the TinyRocketConfig and modified the on-chip scratchpad SRAM size to 2048 kB. We also attached an external AXI bus to integrate with the external verilog IPs from Xilinx and other open-source projects.

One barrier we have encountered is that due to the cake config pattern design used by Chipyard, it is hard to customize the IO connection and pin map of the SoC design on the target FPGA. To address this, we implemented a wrapper framework in Chisel with the Mill build system. The framework wraps around the DigitalTop module generated by Chipyard, and connects the signals between DigitalTop, other IP blocks, and the FPGA IOs manually.

We use Vivado 2024.2 to synthesize the hardware design on Digilent Arty 35T, which uses 15274 LUTs and 11 blocks of BRAM.

We then use JTAG to upload the baremetal program that interfaces with the Ethernet controller and performs the neural network forward inference.

APPENDIX B
STM32 PROGRAMMING

A detailed writeup of the procedure of configuring various STM32 peripheral components can be accessed at https://notes.tk233.xyz/stm32/getting-started-stm32-edition

APPENDIX C
RISC-V BAREMETAL RUNTIME

A more detailed, step-by-step explanation of how we set up the components of the RISC-V baremetal runtime can be accessed at https://notes.tk233.xyz/risc-v-soc/risc-v-baremetal-from-the-ground-up-chipyard-edition