Face recognition applications triggered by image comparison on MAX78000 FTHR RISC_V core

Hanqi XU, *Student ID number: 236804,*
Arthur de Oliveira Lima , *Student ID number: 236858*

***Abstract*—As a very common thing in life, face recognition is widely used on various devices, but these programs are often very energy intensive. At the same time, the popularity of the Internet of Things has higher and higher requirements on the size and energy consumption of equipment. Faced with the increasing demand for energy-efficient technology, this project aims to address the issue of high energy consumption in existing face recognition system. By using the RISC-V core instead of the mature ARM processor, we aim to create a solution that takes advantage of the latest advancements in technology. Starting from existing face recognition examples[1] that utilize both ARM and RISC-V processors, the goal is to optimize the program and create a more energy-efficient solution. By adding an image comparison algorithm to trigger the face recognition system and manage the memory, the result of the project is that the author successfully uses the CNN accelerator, the camera that come with the MAX78000 board, and the ESP32 board to build an autonomous face recognition device with lower energy consumption.**

I. Introduction

A. Problem statement

Facial recognition technology involves capturing an image of a person's face, processing the data, and then comparing it to a stored database of known faces. The goal is to recognize a person in an image and perform a specific action based on the recognition result. In the era of rapid development of the Internet of Things, face recognition is also widely used in various Internet of Things devices to enhance security, personalize user experience and simplify operations, such as security surveillance cameras. Inevitably, this means that the demand for small, low-power, high efficiency facial recognition systems has greatly increased.

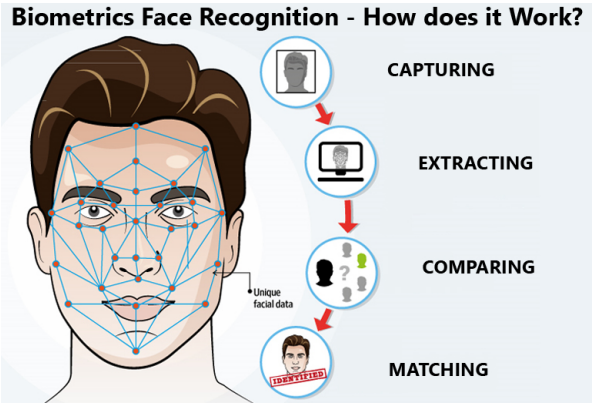


Fig. 1. Face Recognition: How does it works?[2]

B. Possible solutions

The RISC-V ISA, which appeared in 2010, has been widely used in recent years. It is designed to be highly scalable, customizable, and open-source, which gives it several advantages in terms of power consumption compared to the proprietary ARM ISA. The reduced instruction set and simpler architecture of the RISC-V core reduces power-hungry transistors and circuits, thereby reducing power consumption. In addition, the open-source nature of RISC-V provides more opportunities for optimization, such as fine-tuning cores for specific tasks, which can also help reduce power consumption. These advantages can help us achieve the goal of reducing energy consumption.

C. Contribution of the project

- Create an autonomous face recognition device and interact with the user on the results end.
- Optimizing the always-on mode face recognition process by adding trigger conditions.
- Work out a more energy-efficient algorithm
- Improve the accuracy of recognition.

D. Organization of the report

This report will introduce the following sections in detail:
Section II: Works done by the others related to the project.
Section III: Introduction of the hardware part, in which we will introduce the boards that we use: MAX78000_FTHR and ESP32 Devkit v1.
Section IV : Presentation of the software part, in which we will explain in detail how the image comparison module and face recognition module operate.

II. Related Work

We have read some articles on low-power face recognition and here are some of our findings.

Some works focus on improving the accuracy and efficiency of face recognition by improving the algorithm, such as the face recognition algorithm based on two-channel images and VGG-cut model proposed in 2020[3]. The authors use two channels of images (grayscale and edge) to improve recognition accuracy and use the VGG-cut model to extract features. The algorithm achieves high accuracy in experimental results on a RISC-V embedded FPGA platform.

A 2021 paper describes a battery-free, energy-harvesting system for face recognition in edge environments[4]. This

system can be used for wildlife monitoring, health monitoring, and security surveillance, among others, in line with the purpose of our project. The CNN model is based on the VGG-16 architecture and is trained on a large face dataset to achieve high accuracy. The system was evaluated on a dataset of 100 faces and achieved an accuracy of 93.4% with an energy consumption of 51.2 nJ per recognition

Meanwhile, there is a face recognition processor which utilizes a hybrid CNN architecture to achieve high recognition accuracy with ultra-low power consumption[5]. The proposed processor is designed to be integrated with CMOS image sensors (CIS) in mobile devices, allowing always-on face recognition functionality without consuming significant power. The proposed processor achieves 98.6% recognition accuracy on the LFW benchmark while consuming only 61.6 milliwatts of power. The proposed processor is also capable of recognizing faces in real time at a frame rate of 30 frames per second with a power consumption of only 14.2 milliwatts.

A similar paper using an analog-digital computing architecture features a custom-designed face recognition processing unit (FRPU) that is integrated with a CIS (CMOS image sensor) to implement a complete system-on-chip (SoC) for always-on mobile devices[6]. The proposed FRPU includes a hybrid convolutional neural network (CNN) architecture that utilizes analog computing for convolutional operations and digital computing for pooling operations. The system is designed to run in real time and consume less than 1mW of power.

In a paper focused on CNN for limited memory, the authors implemented a real-time camera face recognition DNN in a RISC-V MCU with only 512KB of internal RAM [7]. The authors a lightweight face recognition algorithm designed specifically for microcontrollers with limited memory and processing power. The algorithm uses a compact neural network architecture with only 480K weights to meet the limitations of the microprocessor.

While most of these papers improve the energy consumption problem by optimizing CNNs, our attempt is more straightforward, saving energy by changing the triggering method of face recognition.

III. Hardware Architecture

The main hardware components of our project are the MAX78000_FTHR board produced by maxim integrated and the ESP32 board from Espressif. The MAX78000_FTHR board is responsible for capturing and comparing images, as well as face recognition, and ESP32 is responsible for wireless transmission. The two boards communicate using UART. Since MAX78000 has a 1.8V logic level and ESP32 uses 2.3V, to make them compatible, a logic level shifter is also required. We used a model available in the lab, using the 1.8V output from MAX78000 to power the low side and the 3.3V of ESP32 to power the high side.

A. MAX78000_FTHR

The MAX78000 is a new class of AI microcontrollers designed to provide ultra-low power consumption for neural network execution in IoT applications. It features two processors, Arm Cortex-M4 with FPU Processor and RISC-V RV32 Processor, and a hardware-based convolutional neural network (CNN). It combines a power-efficient AI processing system with a proven ultra-low-power microcontroller, and a convolutional neural network (CNN) accelerator enables battery-operated devices to run AI inference at microjoules of energy.

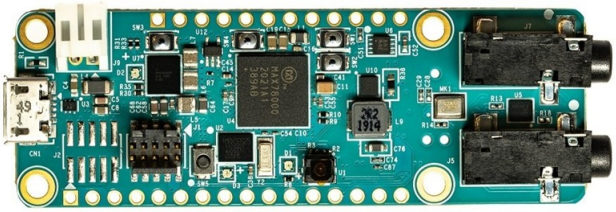


Fig. 2. MAX78000_FTHR Board

The 32-bit RISC-V coprocessor RV32 provides the system with ultra-low power consumption signal processing, and a deep neural network accelerator with 442KB of weight store memory and 512KB of data memory. It also features large on-chip system memory, including 512KB Flash and up to 128KB SRAM, and multiple high-speed and low-power communication interfaces(I2S, PCIF ..). The device is available in a compact 81-pin CTBGA package. The CNN engine architecture is very flexible, allowing the network to be trained using popular tools such as PyTorch and TensorFlow, and then converted to execute on the MAX78000 using tools provided by Maxim. This device has a 442k 8-Bit Weight Capacity and can support weights with precision of 1, 2, 4, or 8 bits. The input image size can be programmed up to 1024 x 1024 pixels, and the network depth can be programmed up to 64 layers. The channel widths for each layer can also be programmed up to 1024 channels[8].

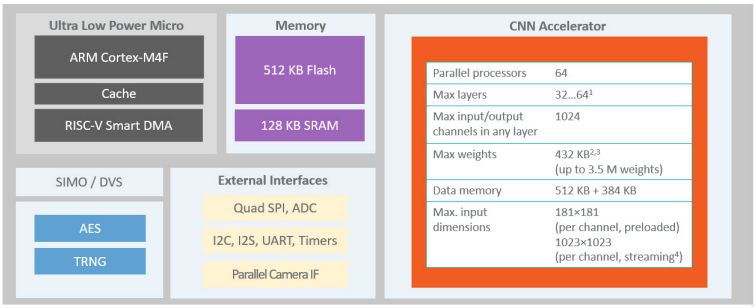


Fig. 3. MAX78000_FTHR features[9]

B. ESP32 Devkit v1

The ESP32 is a 2.4 GHz Wi-Fi and Bluetooth combo chip designed for power efficiency, robustness, versatility and reliability in mobile, wearable, and IoT applications. It features advanced low-power techniques such as fine-grained clock gating, multiple power modes, and dynamic power scaling. The chip is designed to minimize energy consumption, for example, by using a low duty cycle in low-power IoT sensor

Fig. 4. ESP32 DevKit v1

hub applications. The output of the power amplifier can also be adjusted to achieve the optimal balance between communication range, data rate, and power consumption. With only 20 external components, the ESP32 is a highly integrated solution for Wi-Fi and Bluetooth IoT applications.

he ESP32 Devkit v1 has 4 MB of flash memory, 520 KB of SRAM, and operates at a clock speed of 240 MHz. It has 3 UARTs, 2 SPIs, and 3 I2Cs, and supports IEEE 802.11 b/g/n/e/i Wi-Fi. It has an integrated TR switch, balun, LNA, power amplifier, and matching network for wireless connectivity, and supports WEP, WPA/WPA2 authentication, or open networks. The flash memory of the ESP32 is organized in a single flash area with 4096-byte pages and two different layouts based on the presence of BLE support.

IV. Software Architecture

The project directory is divided in modules, each one representing one feature. The modules involved are:

- The Finite State Machine, in `main_riscv.c`, that coordinates the flow.
- The camera module, `image_capture.c`, that communicates with the camera driver, takes and stores pictures and performs the comparison between images.
- The flash module, in `flash_memory.c`, that allows to store and retrieve data in flash memory
- The esp32 module, in `esp32.c`, that performs the UART communication with the esp32 board
- The faceID module, in `faceID.c`, that controls the CNN accelerator and deals with the face recognition feature.

Before start everything, the first thing to do is to put the ARM core into sleep and wake up only the RISC-V core when an interrupt occurs. We use two main functions to accomplish this. The first one is `MXC_NVIC_SetVector()`, which is used to set the interrupt service routine. It is a function that allows to set a call back function for a specific interrupt. So we built a function `WakeISR`, in which we set the `irq0` register value in the signal peripheral.

By using:

```
MXC_SEMA -> irq0 = MXC_F_SEMA_IRQ0_EN &
MXC_F_SEMA_IRQ0_CM4_IRQ
```

, we will enable interrupt generation for the signal and it will ensure that interrupts are not triggered on the CM4 core, which is the core of the ARM subsystem. This will ensure that interrupts are only triggered on the RISC-V core, which will wake it up instead of the ARM core. Finally we make ARM goes to sleep by using `__WFI()`.

However, during testing, we found that if we let the ARM sleep directly, we could not connect to the ARM core with the debugger, making it impossible to flash new code. Every time, an erase operation was needed. To solve this problem, we put a 10s delay before going to sleep to allow the debugger to connect and halt the device.

A. Finite State Machine

The flow of the application is organized using a Finite State Machine (FSM), structured in the main file, calling the other modules to perform the operations. The FSM is described in the image below:

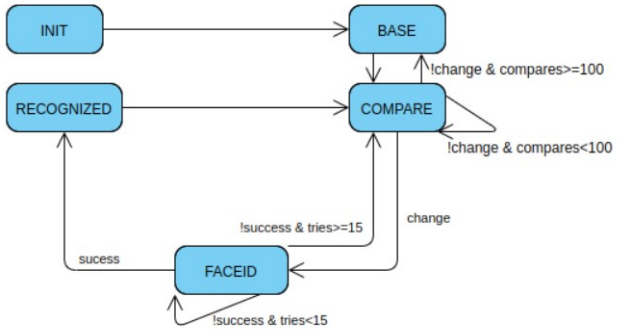


Fig. 5. Finite State Machine

The frequency of the FSM is also defined in the main file, using a timer. To save power, the timer was configured for 1Hz. RISC-V goes to sleep between states. Now, lets describe each state and which modules they call.

1) INIT: The FSM initializes the camera, flash and esp32 modules. Then, it goes to INIT BASE state. The camera module configures the picture params and the flip. The flash module allocates the memory in flash for the clusterized picture. The esp32 module configures the specified UART port for communication. BASE The FSM calls the camera module to take the base picture. Change to COMPARE state. The camera module takes a picture, clusterizes it and saves the result locally.

2) COMPARE: The FSM calls the camera module to compare the current environment to the base and increment the compares counter variable. noitemsep

- If the result is change, change to FACEID state
- If result is no-change, verify compare count:
  - If compare count = 100 : go back to BASE state to take a new base picture
  - If not, stay in COMPARE state

The camera module sends the command to the camera to take a picture, waits for its transfer, clusterizes the result and runs

the SAD algorithm. If the result is bigger than the predefined threshold, returns change.

3) FACEID: The FSM calls the camera module to store the clusterized base pic in flash memory. Then calls the faceID module to try the identification for a maximum of 15 tries. If one face is recognized, go to RECOGNIZED state. If not, calls the camera module to retrieve the clusterized base pic from flash memory and goes back to COMPARE state. The camera module calls the flash module to store the clusterized base pic in flash memory, passing its address and size. The flash module stores it in flash the memory block it was given.

The faceID module loads the faces database and initializes the CNN accelerator. After that, it takes a picture, draws the frame, loads it in the CNN hardware, waits for it to process, gets the result and compares it to every subject in the database. The result can be, based on a percentage, `no_face`, unknown, adjust and a recognition. It runs 5 times for every picture taken (normal and shifting some pixels in each direction) and 15 times the whole process. If at any given point it gets a recognition, it turns off the CNN accelerator, unloads the database and returns the subject detected. If it never gets a recognition, it just returns that no subject was found.

4) RECOGNIZED: The FSM calls the esp32 module to send the picture to the esp via UART. Then, it returns to COMPARE state. The esp32 module first sends a `img_cmd` to the esp, with the image's size, dimensions and pixel mode. Then, it sends the picture in the subsequent transactions, line by line (the packet size is the size of one row of pixels).

B. Camera Module

This module uses the camera driver provided by maxim to take pictures with the camera. It opens a DMA channel and to set the camera parameters, most importantly the image pixel resolution and pixel format. In this project, the resolution chosen was 170 x 150 pixels (height X width), because of the model used to train the face recognition CNN, later explained. The pixel format was RGB565, meaning that each pixel requires 2 bytes, resulting in 16 bits: first 5 for red level (from 0 to 31), the next 6 for green level (from 0 to 63) and the last 5 for blue level (from 0 to 31).

Fig. 6. RGB565

To take pictures, it calls the respective function from the driver and awaits for it to be available. It can then be retrieved from the driver (a memory space is allocated when it initiates). To decode the data (that comes in a `uint8_t` array of raw data) for a rgb structure data type, bit masks (bitwise AND operations) and bit shifting were necessary. All functions in this module have the same return codes, specified with an enum in the .h file. The function `img_capture()` can be called with the capture mode to either capture base and store it or to take a new one and compare it to the base.

There exist several algorithms for image comparison, such as MAD, SAD, SSD, MSD, MSE, etc. Considering the resolution and memory limitation of the images we use, the running consumption time and other factors, we chose MSE(mean-square error) and SAD(Sum of Absolute Differences) for testing, and finally our program chose SAD as the image comparison algorithm.

Sum of Absolute Differences (SAD) algorithm, which is the simplest matching algorithm commonly used for image matching, where the absolute value of the difference between the corresponding values of each pixel is summed to evaluate the similarity of two images. The logic of MSE is basically similar to that of SAD, however, instead of the simple summation of SAD, MSE requires taking the square of the difference between each pixel in picture A and the corresponding pixel in picture B, adding them together and dividing them by the number of pixels.

In the test process, we found that the calculation time of MSE is longer than that of SAD, and the accuracy of the result is not much different from that of SAD, so we finally chose SAD which has a simpler operation logic (V1 - version of the algorithm).

Usually SAD is mostly based on grayscale images, however, since the pixel type of the image we choose is RGB565, we need to perform the difference and summation operations on the corresponding values of R, G, B separately.

Naturally, every two pictures taken will have a significant difference as output, so some experimentation and testing is required to identify the threshold to define when they are actually different. Since this number may be considerably high and difficult to define based on the environment conditions, the first improvement developed was to discard the contribution of pixels with low differences (basically the same pixel) to the absolute sum (V2). A minimum difference is set as a parameter in the .h file and defined using, again, experimentation. This proved to drastically decrease the overall SAD output and also increase the gap from an unchanged environment to a changed one.

A critical problem faced in this step of development was memory availability. The MAX78000 was not capable of storing the base and later the new picture to compare. The solution developed was to clusterize the image in blocks and compare these blocks instead of individual pixels. Since the SAD algorithm relies on the absolute difference, representing a block of pixels as the sum of all the pixels it contains should still result in useful results. The process can also be understood as a lesser resolution image. After development, this proved to be even more effective (V3), again increasing the gap on the output of no change and change, especially when a minimum difference in block difference is still used (V4). This probably is due to the fact that only bigger area changes in the environment should result in a considerable change in the pixel block.

Below, the pictures used for testing and the results achieved for each algorithm.

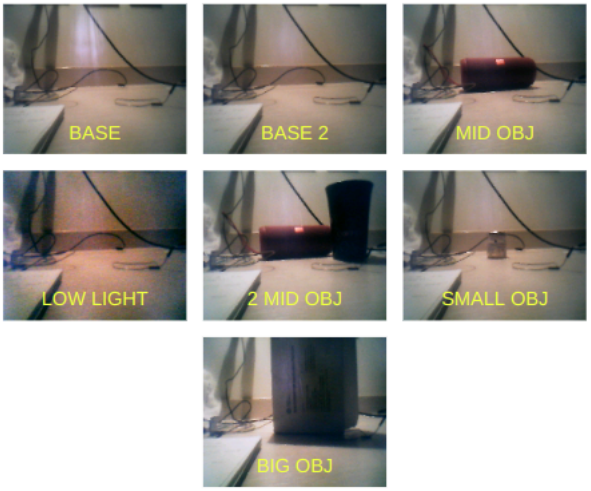


Fig. 7. Testing images for the different compare algorithms. The BASE 2 picture is picture taken just after the first to see the output of an environment with no actual change.

SAD output (x10^6)				
PIC	V1	V2	V3	V4
BASE 2	0.65	0.28	0.16	0.03
LOW LIGHT	1.07	0.52	0.43	0.32
SMALL OBJ	0.85	0.37	0.35	0.21
MID OBJ	1.09	0.54	0.53	0.43
2 MID OBJ	1.09	0.52	0.58	0.50
BIG OBJ	1.09	0.53	0.52	0.43

When a base picture is taken, it is decoded, clusterized and then stored locally in a dynamic allocated array of `rgb_t` structures. When a compare picture is taken, it is decoded, clusterized and compared to the base picture using the final version of the SAD algorithm (using clusters and minimum difference), returning `IMG_CAP_RET_CHANGE` in case the result is greater than the predefined threshold `SAD_THRESHOLD`.

Finally, to again save memory space when needed, the module is able to free the clusterized base picture allocated memory by storing it in flash memory (using the flash module, later explained), being able to retrieve it again later. These operations are performed using `img_capture_free_base()` and `img_capture_rec_base()` functions.

C. Flash Module

The code of the flash module is mainly used to help us solve the problem of insufficient memory. When we run the face recognition process, in order to free up enough memory for computation and CNN, we need to store the base image temporarily in flash.

We have used three main functions to accomplish this operation, `flash_memory_init()` to allocate the specified size(size of the base image) in flash memory to be used, `img_capture_free_base()` to store the base image into flash and remove it from SRAM, and `img_capture_rec_base()` to restore the base image to SRAM.

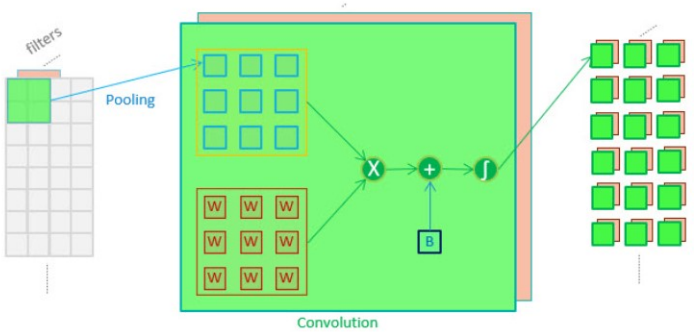
When we initialize everything about the image capture, `flash_memory_init()` is called. This function can allocate some memory in flash without affecting the main program operation and saving. The flash memory is divided into many pages. in the init function, it first determines how many pages of flash need to be used based on the required memory size. if the number of used flash pages plus the number of newly required pages is greater than the total number of pages, then an error is returned and it is informed that it cannot be stored. If there are enough pages in the flash to store, then the function will take all the data in those pages and use them to prepare for later writes.

Before starting the faceID process,the `img_capture_free_base()` function will be called to leave more space. When trying to store the base image into flash, the first step is re-confirm is whether there is still enough memory in flash, and when it is confirmed, we can call the `flash_memory_write()` function in flash driver provided by maxim to modify the data on the allocated memory in flash. When writing data to flash, the program writes one byte at a time and reads it when it is done to confirm that the data was written successfully. Once we get the feedback that the copy was successful, we can use the `free()` function to remove the base image from the SRAM.

After the program gets the result of faceID, we start restoring the base image. The process of restoring the base image is similar. The first step is to first confirm with `malloc()` if there is enough memory to store the image, and if so, then use the `flash_memory_read()` function to read the data related to the base image and copy it to the memory location we just allocated with malloc.

D. face ID Module

The CNN model used in the program is the one that comes with the sample program faceid. It is an automatically generated, 512-length embedding by the ai8x-synthesis tool. It is trained with the VGGFace-2 dataset and uses the MTCNN and FaceNet models to generate embeddings that can be used for face image recognition. The size of images used for training was 200 x 150 pixels (height X width)The code runs a known answer test using pre-defined input samples[1].



[9]

Fig. 8. CNN: Convolution neural network

First of all, the initialization of the CNN involves loading the CNN's bias, kernels, activating the CNN-related interrupt,

clocks, and initializing the database. In order to initialize the database, memory needs to be allocated for various structures. It's worth noting that to prevent memory from being overconsumed, the allocated memory needs to be freed at the end of the analysis altogether.

After all the initialization is done, the program enters a loop that repeats 15 times. At the beginning of each loop, a photo is taken, and the `draw_frame()` function is used to draw a frame with a black line in the central part of the image to help CNN analyze the image afterwards.

After the frame is drawn, the face recognition process is started. The details of the image are obtained from the camera driver, and the CNN is run with the image's data inputted into it. When inputting the image, the selection range of the image can be adjusted by adjusting the offset of x and y.

While CNN analyses the data, the system is allowed to sleep to save energy, and then CNN automatically generates a hardware interrupt to wake up the system later. After the analysis is finished, the CNN is stopped, and the image data is unloaded to prepare for the next analysis to save energy.

The program then uses the `calculate_minDistance()` function that computes the minimum Euclidean distance between the data of an input image and a database of embeddings and returns the index of the closest match. The database is represented by a pointer to a structure that contains information about the embeddings and the topics to which they belong.

Depending on the percentage of the input image matching the embedding, the program returns a different conclusion.

- similarity>80%: return the corresponding subject name.
- 80% >similarity>40%: return adjust face and reminds the user to take a more accurate image.
- 40% >similarity>20%:return face unknown.
- 20% >similarity>10%: return no face in the image.

In each loop, the program runs the CNN five times by default, where each time has a different x and y offset. Once one of them has successful face recognition, the loop is jumped out and goes back to the main program for the next step. If there's always no face detected, the program goes back to image comparison.

E. ESP32 Module

In this subsection, the feature will be described both from the view of the MAX78000 (ESP32 module) and the ESP32 board (whole project). To develop a project for that board, the esp-idf tool for VS Code was used. It contains all the drivers and necessary tools to code, flash and integrate with that plataform. The description on how to use this tool and all other documentation and instructions can be found in the references table.

The ESP32 board runs a FreeRTOS and an application should be organized in threads. For this project, there are 3 threads: WiFi, uart and udp. The first deals with WiFi connectivity and events, allowing the system to connect to a WiFi network using the SSID and PASSWORD set with macros. The second deals with the communication with MAX78000. Finally, the last sends packets to the specified UDP server (IP and PORT in macros). The code for the project can be found inside `/tools/esp_tool` folder.

To communicate the boards using UART, the first step is configuring the port with the correct parameters (baud, parity, data bits, stop bits and flow control). This can be easily done using the UART driver provided by both SDKs. We decided to use UART2 port in MAX78000 (since UART0 is used for debugging, UART1 is not available in the Featherboard and UART3 is a low power option with limited baud) and UART2 in ESP, although in the first case it is only initialized, while in the second an event handler is assigned to deal with port events. This difference is due to the fact that, for this project, MAX78000 will only be transmitting and ESP32 only reading. This event handler has a specific event that is called when a given pattern (configurable) is matched, passing to the handler function the UART buffer before the pattern. This can be very useful to detect packets coming from the receiver that also knows that pattern. For this project `"\r\r\r"` is used.

To perform the communication between the boards in a versatile way and also to avoid overflow in the buffer (something that is expected because of the image size) the dialog is based on sending a command first with the parameters necessary to then send the actual data in a sequence of packets ended with the pattern. This allows the esp32 board to know what to expect and also opens the possibility to later use new commands for new features. The first byte of every message is the command code, which is an enum commom to both parts, and the rest of the packet is the a struct type containing information related to that command, also know by both parts. Than, in case data packets need to be sent in subsequent packets, the receiver knows how many to expect and the size of each. For now, the only command supported is the `MSG_TYPE_IMG` for which the information structure is `esp_msg_img_t` that cotains the image size, width, height and pixel format.

In the MAX78000 esp32 module, a buffer is allocated when the module is initialized and it will be used by all other module functions by filling it with the data they need to send. When the actual send function is called, giving it the packet size, it appends the pattern to the end of the data stored in the buffer and sends it with the added size. In the case of image transfer, the outside module calls `esp32_send_img()` function with the image parameters: size, width, height, pixel format and raw image pointer. Then, the function first fills the buffer with the image command packet (and then calls send) and then proceeds with the image row by row, making it a total of height + 1 (number of rows + command) packets. The full transaction is described in image 9

On the other end, the ESP32 board is capable of interpreting a new command packet. It stays in idle until a packet is received, verify the packet type, saves the information of the data incoming and knows how to proceed with all the next packets received. Then, it goes back to idle, waiting for the next command packet. For the image command, in the current version, it basically works as a bridge to the UDP server. Every packet it receives it forwards, as it is, to the destination IP address and PORT, by unblocking the UDP threat. This threat, after startup (configuration) and after every

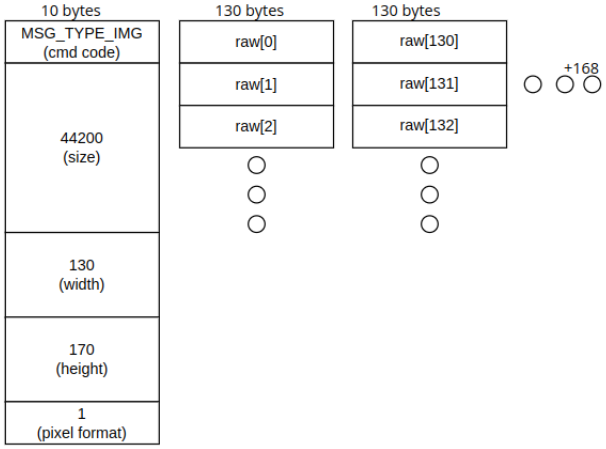


Fig. 9. Packets description for image transaction using ESP32

send packet, is blocked indefinitely. When unblocked, it sends to the destination the current content of the buffer and blocks itself again.

V. Testing and Results

While testing, the project proved to work well in a controlled environment. It showed remarkable stability in recognizing change in the picture, reacting in time to real disturbances when in a well lightened place, detecting both motion and new objects. It was successfully able to identify different faces registered in the database. It was also perfectly capable of getting the base picture back once the FaceID state passed and again being stable, not triggering it back if the change is not persistent. Even in lower frequencies for the state machine (longer sleeping periods), the system showed great results.

To receive the image sent by the ESP32, a simple python script run locally in a computer was successfully able to receive the packets. Using the same PC Tool provided by other examples for MAXIM78000, the raw data was converted to a JPEG file. The same python script saves it locally and also sends it to a cloud database storage. In this way, the system proved to be fully operational and also autonomous, provided the connection to WiFi.

However, some problems are still apparent and need to be improved:

- A few bytes are constantly being lost in the UART transfer, making it impossible to rebuild the image at the final node. This is most probably due to hardware faults related to the wires (simple jumpers) or even the level shifter. The strategy we followed was to diminish the baud rate and introduce delays between packets, which proved to make it work. Ideally, this issue could be tackled using flow control, introducing one more wire to the system. Also, by better connections, a more appropriate level shifter or even designing the CPU's to be in the same board. At software level, an acknowledgment protocol could also be developed, making MAX780000 re-transmit the packet in case of faulty reception. As a last resource, some image processing techniques could be used in the python script to identity this faults and fill missing bytes of the image.
- The change detection is still very reactive to light change, since it cannot distinguish between an actual change in the picture or just some shadow. This is due to the fact that changing the light will naturally incur in a RGB level difference, resulting in a superior SAD. A possible future solution could be associating a light sensor in the analysis or even using some more advanced image comparing algorithms to observe change in the light.

In terms of power consumption, the average current was measured during the different stages of the application using the Power Monitor available for the MAX78000_EVKIT. This monitor outputs the current drawn for total MAX78000 load - called DDB, powered by 3.3V - and just for CNN, MCU, digital & memory loads - called CA, powered by 1.2V. Using those currents, the calculated power consumption in each state is detailed in this table:

Power Consumption (mW)		
STAGE	CA	DDB
INIT	12.6	75.2
COMPARE	7.3	69.0
FACEID	17.2	22.4
RECOGNIZED	8.5	13.2

First of all, it is necessary to put into context the consumption in each state. During INIT, ARM processor is still awake (the busy wait to allow debugger to interrupt), the camera is taking the base picture and CNN accelerator is off. During COMPARE, RISC-V is running, the camera is constantly taking new compare pictures and the other two cores are both off. During FACEID, both CNN and RISC-V are on, although the camera picture taking is less frequent. Finally, for RECOGNIZED, only RISC-V and the UART port are running while no camera use or CNN are required.

From the results, we can conclude that RISC-V can run its algorithm on its own using just around 8mW, which is a considerable improvement over the 17mW needed for CNN, that also requires coordination by at least one of the cores. Furthermore, the biggest conclusion is that the camera is the most power hungry element of the project, consuming significantly more power than any other feature.

A. Conclusion

Memory consumption is a key factor in most embedded system projects. Running a CNN can be a costly operation, even with a low power dedicated hardware like the accelerator in MAX78000 board. For this reason, finding ways to optimize the time on of the CNN, using triggers, can be a useful resource to save energy. This project aimed to run a simplified algorithm using only RISC-V low power CPU, capable of identifying the correct moment to activate (and deactivate) the CNN.

Furthermore, we aimed to develop a platform that is autonomous, but can be integrated to a higher level layer, which is a requirement for most IoT applications. Such thing was achieved using a ESP32 board that has wireless connectivity.

The biggest challenge, like in most embedded platforms, was managing the memory, that is very limited. Various

solutions were developed to save memory space: clustering information so that it consumes less space but still serves its purpose; freeing up space in the appropriate moment, while also storing it in a way to get it back later; loading and unloading the database; sending the necessary images to another platform.

In conclusion, the results of the project were very promising, showing the capability of saving almost half the power needed to run a CNN application by just detecting a change in the environment using RISC-V and then turning on the image recognition feature. Also, it proved to be a very scalable application, since it is autonomous and integrated with different application levels, even connecting to the cloud.

Furthermore, most of the modules develop are self-sufficient, meaning that they can be used by other applications in the future, just requiring some minor adaptations. We hope the current state of the project can be a blueprint for further advancements, even though it is already a real life application on its own. All the modules were also developed in a way to have room for expansion.

1) Future work and directions:

- Adapt parameters in the program or add new modules for specific applications.
- Improve image comparison functions to avoid false positive changes(ex:light).
- Diminish the camera power consumption using software and hardware strategies.
- Better physical connections for the integration with ESP32 board.
- Optimize CNN, do more learning or try new models to achieve more accurate face capture and recognition.

You can find all the code and other relevant information about the project here.

REFERENCES

[1] MAX78000_sdk/examples/MAX78000/CNN/faceid_evkit-riscv at master · analog-devices-MSDK/MAX78000_sdk. [Online]. Available: https://github.com/Analog-Devices-MSDK/MAX78000_SDK

[2] Biometric facial recognition system. [Online]. Available: https://www.starlinkindia.com/blog/biometrics-face-recognition/

[3] D. Su, Y. Li, Y. Zhao, R. Xu, B. Yuan, and W. Wu, "A face recognition algorithm based on dual-channel images and VGG-cut model," vol. 1693, p. 012151.

[4] P. Jokic, S. Emery, and L. Benini, "Battery-less face recognition at the extreme edge," in *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)*. IEEE, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/9462787/

[5] J.-H. Kim, C. Kim, K. Kim, and H.-J. Yoo, "An ultra-low-power analog-digital hybrid CNN face recognition processor integrated with a CIS for always-on mobile devices," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8702698/

[6] J. Choi, S. Lee, Y. Son, and S. Y. Kim, "Design of an always-on image sensor using an analog lightweight convolutional neural network," vol. 20, no. 11, p. 3101. [Online]. Available: https://www.mdpi.com/1424-8220/20/11/3101

[7] M. Zemlyanikin, A. Smorkalov, T. Khanova, A. Petrovicheva, and G. Serebryakov, "512kib RAM is enough! live camera face recognition DNN on MCU," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, pp. 2493–2500. [Online]. Available: https://ieeexplore.ieee.org/document/9022333/

[8] "MAX78000 - artificial intelligence microcontroller with ultra-low-power convolutional neural network accelerator."

[9] Keywords spotting using the MAX78000 | analog devices. [Online]. Available: https://www.analog.com/en/design-notes/keywords-spotting-using-the-max78000.html