Iby and Aladar Fleischman
Faculty of Engineering
Tel Aviv University

הפקולטה להנדסה
ע"ש איבי ואלדר פליישמן
אוניברסיטת תל-אביב

Accelerating AI apps performance using ARM Cortex M-33

Project Number: 2851

Student: Omer Bahary ID: 315329946

Student: Lior Ben Ishay ID: 313135261

Supervisors: Yoav Ben-Yehezkel, Guy Mishol

Project Carried Out At: Texas Instruments Israel, Ra`anana

Contents

Abstract

Our project investigates the implementation of AI apps on Texas Instruments edge devices and accelerating them. The result is an embedded system which detects people presence, using a CNN in real-time, on a low memory, low power chip. This was done by connecting a TI (Texas Instruments) device - next generation BLE product which is used for Bluetooth applications to an SPI camera (Arducam OV2640), and designing a software system which will yield the presence detection. A CNN (MobileNet) for people detection was trained and tested on a GPU using Python (TensorFlow), and we stored its output (convolutional layers, weights and biases) in the Flash memory of the device. After an image is received and sent to the device memory, multiplications and convolutions are being performed to infer if there is a person in the image or not. To accelerate the computations during an inference (convolutions, multiplications, etc.) we used the M-33 CDE – Custom Datapath Extension. The ARM M-33 has a hardware extension unit that can accelerate the computations using parallelism. We have added software which will make the relevant API`s (convolutions) to compile to the CDE assembly – and run on the acceleration unit instead of the main processor. **We have conducted a demo of the running project to TI`s SVP Amichai Ron, and the project will be presented at ELECTRONICA, an embedded worldwide exhibition.**

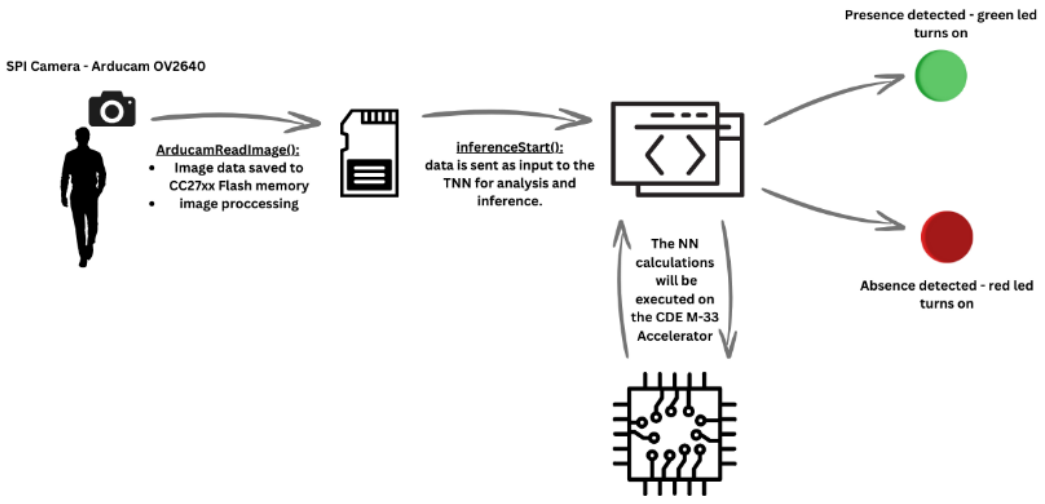

SPI Camera - Arducam OV2640
ArducamReadImage():
Image data saved to CC27xx Flash memory
image proccessing
inferenceStart():
data is sent as input to the TNN for analysis and inference.
The NN calculations will be executed on the CDE M-33 Accelerator
Presence detected - green led turns on
Absence detected - red led turns on

Figure 1: Block diagram

1 Introduction

Edge computing has become increasingly significant in the era of the Internet of Things (IoT) due to its ability to process data at the source, thus reducing latency and bandwidth usage. This project aims to implement and accelerate AI applications on edge devices, specifically targeting real-time people detection using a Convolutional Neural Network (CNN). The device chosen for this project is the TI Next-Gen device, a low-memory, low-power chip primarily with an ARM M-33 processor used for Bluetooth applications, connected to an SPI camera.

1.1 Goals

The primary goal of this project is to develop an embedded system capable of real-time people detection with high accuracy on a resource-constrained edge device, by using ARM M-33 acceleration hardware. Business-wise, we wish to prove the concept of developing AI applications on TI`s devices. This involves:

- Implementing a lightweight CNN model (MobileNet) on the edge device.
- Accelerating AI computations using a hardware-based Custom Datapath Extension (CDE).
- Reducing the power and memory footprint while maintaining real-time processing capabilities.
- All is done with respect to TI internal software development tools, hardware drivers and hardware products.

1.2 Motivation

The motivation behind this project stems from the rapidly expanding need for intelligent edge devices across a wide spectrum of applications, including but not limited to smart homes, industrial automation, the automotive industry, surveillance systems, healthcare, and wearable technology. Traditional cloud-based AI processing, while powerful, often presents significant challenges such as high latency, substantial energy consumption, and potential privacy concerns due to the necessity of constant data transmission to remote servers.
By bringing advanced AI capabilities directly to edge devices, we aim to address these challenges and unlock new possibilities. This approach offers several key advantages:

- **Enhanced Efficiency:** Local processing reduces the energy and bandwidth required for data transmission, leading to more sustainable and cost-effective solutions.
- **Reduced Latency:** By eliminating the round-trip time for data to travel to cloud servers and back, we enable near-instantaneous decision-making crucial for applications like autonomous vehicles or industrial safety systems.
- **Improved Privacy and Security:** Keeping sensitive data on the device minimizes exposure to potential breaches during transmission or storage in the cloud.

- **Real-time Decision Making:** Edge AI allows for immediate analysis and action, critical in time-sensitive scenarios such as medical monitoring or predictive maintenance in manufacturing.
- **Offline Functionality:** Devices can continue to operate intelligently even without internet connectivity, enhancing reliability in remote or unstable network environments.
- **Scalability:** Distributing AI processing across many edge devices can alleviate the burden on centralized cloud infrastructure, allowing for more scalable and robust systems.

These advancements pave the way for more responsive, context-aware, and privacy-conscious applications. For instance, in smart homes, devices could learn and adapt to inhabitants' behavior patterns without sending personal data to external servers. In healthcare, wearable devices could provide real-time health insights and alerts without compromising patient confidentiality. The automotive industry could benefit from vehicles capable of making split-second decisions based on local sensor data, enhancing safety and autonomy.

Moreover, this shift towards edge AI aligns with the growing trend of decentralized computing and the Internet of Things (IoT), where billions of connected devices work together to create smarter, more efficient systems. By empowering each device with AI capabilities, we're not just improving individual gadgets, but potentially revolutionizing entire ecosystems of interconnected technology.

1.3 Approach to Solving the Problem

Our approach involves multiple steps:

- **Model Selection and Training:** We trained a lightweight CNN, MobileNet, suitable for embedded systems, on a GPU using Python and TensorFlow.
- **Data Storage and Transfer:** The trained model, including convolutional layers, weights, and biases, was stored in the Flash memory of the TI device.
- **System Integration:** We connected the TI device to an SPI camera, enabling it to receive and process images in real-time.
- **Computation Acceleration:** To accelerate image processing, we utilized the M-33 CDE – Custom Datapath Extension – which allows parallel computation. Custom software was developed to compile relevant APIs to CDE assembly, optimizing the computational workload.

1.4 Comparison Against Existing Work

While existing solutions primarily rely on either cloud-based processing or high-end edge devices, our project focuses on achieving sophisticated AI tasks on resource-constrained, low-power devices. Many existing implementations either suffer from high latency due to cloud dependency or require substantial hardware resources, which are not feasible for small, low-cost devices. By leveraging efficient model architectures like MobileNet and the computational acceleration provided by the CDE, our approach provides a balanced solution that addresses both performance and power efficiency.

2 Theoretical background

In this section, we will explore the theoretical underpinnings of our project, specifically focusing on the convolutional neural network (CNN) architecture used, the concept of edge computing, and the Custom Datapath Extension (CDE) we leveraged. We will also discuss alternative algorithms and implementations that could be used for real-time people detection on edge devices.

2.1 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a class of deep neural networks specifically designed to process and analyze visual data. They are composed of multiple layers, each performing a convolution operation followed by a non-linear activation function. The key components of a CNN include:

- **Convolutional Layers**: These layers apply a set of filters to the input image to create feature maps. Each filter captures different features, such as edges, textures, and patterns.
- **Pooling Layers**: These reduce the spatial dimensions of the feature maps, thereby decreasing the computational load and focusing on the most prominent features.
- **Fully Connected Layers**: These layers connect every neuron in one layer to every neuron in the next layer and are typically used towards the end of the CNN for classification tasks.

Binary Neural Networks (BNNs)

Binary Neural Networks (BNNs) are a type of neural network where the weights and activations are constrained to binary values (e.g., -1 and 1). This significantly reduces the computational complexity and memory usage, making BNNs highly efficient for deployment on resource-constrained devices. Key characteristics include:

- **Binary Weights and Activations**: Simplifies the arithmetic operations to bitwise operations, which are computationally cheaper.
- **Efficiency**: Reduces the model size and power consumption, making it suitable for edge devices and embedded systems.

Ternary Neural Networks (TNNs)

Ternary Neural Networks (TNNs) extend the concept of BNNs by allowing weights and activations to take one of three values (e.g., -1, 0, and 1). This provides a balance between model complexity and performance. Key characteristics include:

- **Ternary Weights and Activations**: Introduces a zero value, which can help in sparsity and further reduce computational load.
- **Performance**: Offers a trade-off between the simplicity of BNNs and the accuracy of full-precision networks.

Quantization Process

Quantization is the process of reducing the precision of the weights and activations in a neural network. This is crucial for deploying models on devices with limited computational resources. The main steps in the quantization process include:

- **Uniform Quantization**: Maps the continuous range of values to a discrete set of levels uniformly. This is simple and efficient but may not always capture the distribution of the data effectively.
- **Non-Uniform Quantization**: Uses a non-uniform set of levels, which can better capture the distribution of the data, leading to higher accuracy at the cost of increased complexity.
- **Mixed-Precision Quantization**: Combines different levels of precision within the same model to balance performance and efficiency.

- Quantization helps in reducing the model size, power consumption, and inference time, making it essential for deploying deep learning models on edge devices and embedded systems.
- Would you like more details on any of these topics?

For this project, we selected MobileNet, a lightweight CNN architecture designed for efficiency on mobile and embedded hardware. MobileNet utilizes depthwise separable convolutions, which significantly reduce the number of parameters and computational requirements compared to traditional CNNs.

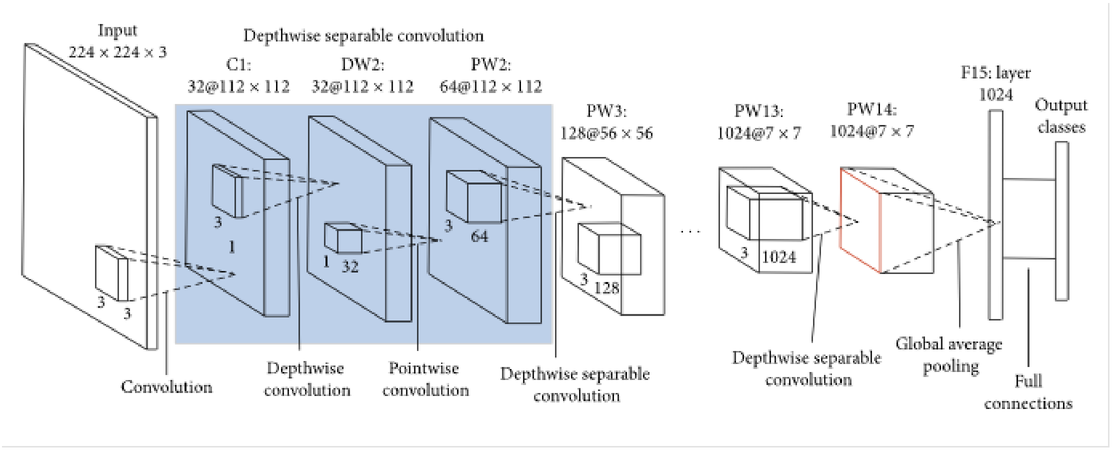


Figure 2: MobileNet CNN Architecture

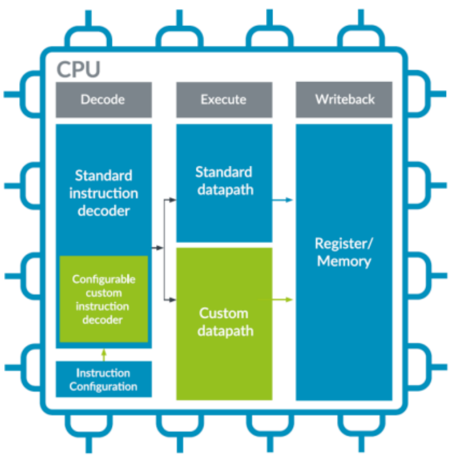
The version of mobileNet we have used was successfully optimized for our devices by using the quantization process. This optimization was done by a dedicated team at TI specialized in working on Edge-AI NN. They reduced the model's precision to 8-bit integers, making it more efficient for resource-constrained environments. Additionally, they translated the quantized model into C code, ensuring it was optimized for TI's architecture. This involved efficient memory management and optimized arithmetic operations, resulting in a high-performance, low-power machine learning model suitable for embedded systems.

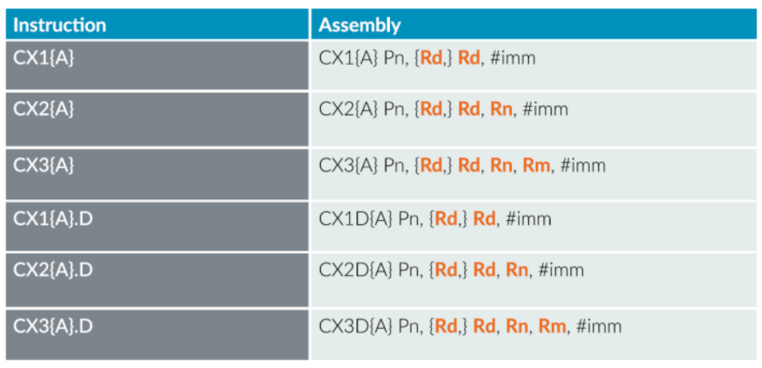
2.2 Edge Computing

Edge computing refers to the practice of processing data near the source of data generation, rather than relying on a centralized cloud infrastructure. This approach offers several advantages, such as reduced latency, lower bandwidth usage, and enhanced data privacy. In our project, edge computing allows us to perform real-time people detection directly on the TI Next-Gen device without needing to send data to the cloud.

2.3 ARM's Custom Datapath Extension (CDE)

ARM's Custom Datapath Extension (CDE) is an innovative feature designed to enhance the flexibility and performance of ARM Cortex-M processors, particularly in embedded systems and IoT devices. CDE allows semiconductor companies and OEMs to add custom instructions to the ARM instruction set, enabling hardware acceleration for specific applications without the need for a complete processor redesign.


CPU
Decode
Execute
Writeback
Standard instruction decoder
Standard datapath
Register/ Memory
Configurable custom instruction decoder
Custom datapath
Instruction Configuration



Instruction	Assembly
CX1{A}	CX1{A} Pn, {Rd,} Rd, #imm
CX2{A}	CX2{A} Pn, {Rd,} Rd, Rn, #imm
CX3{A}	CX3{A} Pn, {Rd,} Rd, Rn, Rm, #imm
CX1{A}.D	CX1D{A} Pn, {Rd,} Rd, #imm
CX2{A}.D	CX2D{A} Pn, {Rd,} Rd, Rn, #imm
CX3{A}.D	CX3D{A} Pn, {Rd,} Rd, Rn, Rm, #imm

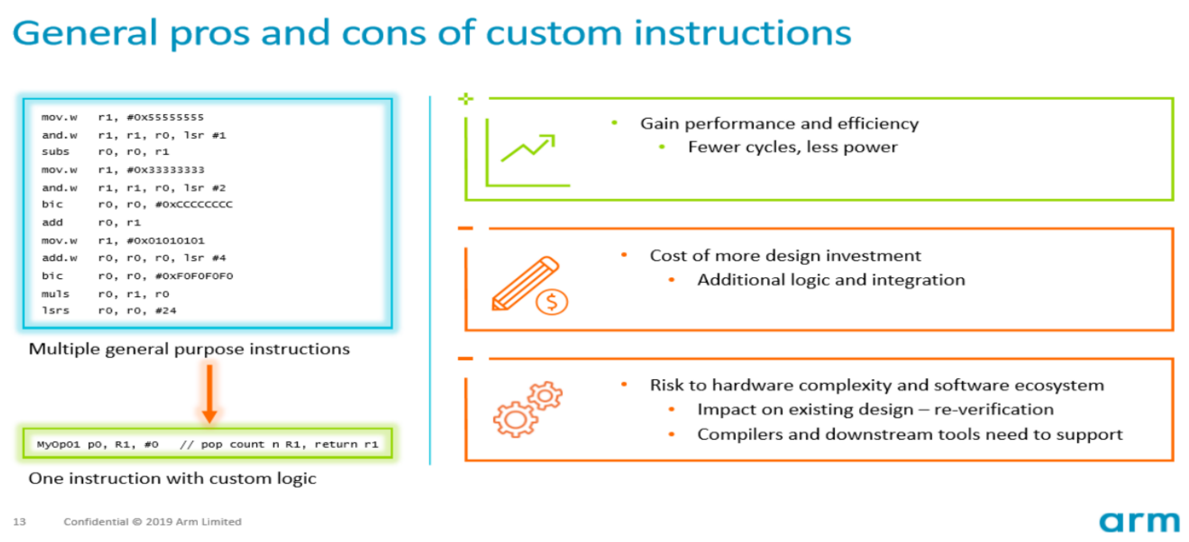
Figure 3: CDE Visualization

The primary purpose of CDE is to improve processing efficiency for specialized tasks. By implementing custom instructions tailored to specific algorithms or operations, CDE offers significant advantages for embedded systems:

- **Performance Optimization:** Custom instructions can accelerate specific operations critical to the application, compared to using standard instructions.
- **Flexibility:** Chip vendors can implement domain-specific instructions tailored to their hardware.
- **Efficiency:** By integrating custom operations directly into the processor, CDE can reduce power consumption and improve overall system efficiency.

These advantages are particularly beneficial for computationally intensive tasks such as signal processing, cryptography, or artificial intelligence algorithms.

On Next-Gen devices, Texas Instruments (TI) has decided to employ the CDE capability to enhance matrix multiplication performance. This is achieved through a dedicated hardware accelerator, designed by TI architects. When a custom instruction is encountered in the code, it's executed using this dedicated hardware, bypassing the need for multiple standard instructions to achieve the same result.



General pros and cons of custom instructions

```
mov.w   r1, #0x55555555
and.w   r1, r1, r0, lsr #1
subs    r0, r0, r1
mov.w   r1, #0x33333333
and.w   r1, r1, r0, lsr #2
bic     r0, r0, #0xCCCCCCCC
add     r0, r1
mov.w   r1, #0x01010101
add.w   r0, r0, r0, lsr #4
bic     r0, r0, #0xF0F0F0F0
muls    r0, r1, r0
lsrs    r0, r0, #24
```

Multiple general purpose instructions

```
MyOp01 p0, R1, #0   // pop count n R1, return r1
```

One instruction with custom logic

+
- Gain performance and efficiency
  - Fewer cycles, less power

–
- Cost of more design investment
  - Additional logic and integration

–
- Risk to hardware complexity and software ecosystem
  - Impact on existing design – re-verification
  - Compilers and downstream tools need to support

13 Confidential © 2019 Arm Limited

arm

Figure 4: CDE Instruction Example

From a software development perspective, working with CDE typically involves using intrinsics or inline assembly to access the custom instructions. Compilers and development tools need to be aware of these custom instructions to generate optimal code. This may require updates to the toolchain, or the use of specialized tools provided by the chip manufacturer.

In conclusion, ARM's Custom Datapath Extension represents a powerful tool for enhancing the capabilities of Cortex-M processors. By allowing the integration of application-specific hardware acceleration directly into the processor core, CDE enables developers to achieve significant performance improvements and energy savings in embedded systems, particularly for specialized and computationally intensive tasks.

2.4 Acceleration Hardware

To accelerate the computationally intensive tasks of the CNN, we employed the Custom Datapath Extension (CDE), a specialized hardware unit capable of parallel processing. The M-33 CDE in our TI device optimizes operations by allowing multiple computations to be carried out simultaneously, thereby speeding up the inference process. In the following figure we can see how a CDE assembly instruction is being executed on the M-33 acceleration hardware. In this example, X[1],...,X[3] are 8 bit integers, and W[0],...,W[32] are 2 bit weights (Ternary). The acceleration hardware runs the convolution on 4 parallel multiplication and addition units - accumulation and saturation.

2.5 Alternative Algorithms

While we utilized MobileNet for its efficiency and suitability for edge devices, other algorithms and models can be considered for similar applications:

2.4.1 YOLO (You Only Look Once): This is a highly efficient object detection algorithm that can achieve real-time performance. YOLO uses a single neural network to predict

$$\begin{aligned}
Y[3] &= \text{Saturate}\,(Y[3] + W33 * X[3] + W32 * X[2] + W31 * X[1] + W30 * X[0]) \\
Y[2] &= \text{Saturate}\,(Y[2] + W23 * X[3] + W22 * X[2] + W21 * X[1] + W20 * X[0]) \\
Y[1] &= \text{Saturate}\,(Y[1] + W13 * X[3] + W12 * X[2] + W11 * X[1] + W10 * X[0]) \\
Y[0] &= \text{Saturate}\,(Y[0] + W03 * X[3] + W02 * X[2] + W01 * X[1] + W00 * X[0])
\end{aligned}$$

Xs: 8 bit signed/unsigned inputs, Ws: 2-bit Ternary weights (+1/0/-1), Ys: 16 bit outputs

bounding boxes and class probabilities directly from full images in one evaluation.

2.4.2 SSD (Single Shot MultiBox Detector): SSD is another popular real-time object detection algorithm. It eliminates the need for a separate proposal generation step, enabling faster detection speeds.

2.4.3 Tiny YOLO: A variant of the YOLO algorithm, Tiny YOLO, is specifically designed for performance on resource-constrained devices. It trades off some accuracy for reduced computational requirements.

2.4.4 SqueezeNet: This is a small model that achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. It is designed to be efficient both in terms of memory and computational resources.

2.6 Communication Protocols

In our work, we have used several communication protocols to receive and send the data.

SPI (Serial Peripheral Interface) Protocol

The Serial Peripheral Interface (SPI) is a synchronous serial communication protocol developed by Motorola. It is widely used for short-distance communication in embedded systems, particularly for connecting microcontrollers to peripherals such as sensors, ADCs, and memory devices.

- Key Characteristics:
  1. Full-duplex communication: Allows simultaneous data transmission and reception.
  2. Master-slave architecture: One master device controls one or more slave devices.
  3. Synchronous operation: Uses a clock signal to synchronize data transfer.
- SPI employs four primary signal lines:
  1. SCLK (Serial Clock): Provided by the master to synchronize data transfer.
  2. MOSI (Master Out Slave In): Data line from master to slave.

3. MISO (Master In Slave Out): Data line from slave to master.
4. SS/CS (Slave Select/Chip Select): Activates the slave device for communication.

I²C (Inter-Integrated Circuit) Protocol

The Inter-Integrated Circuit (I²C) protocol, developed by Philips Semiconductor (now NXP Semiconductors), is a multi-master, multi-slave, packet-switched, single-ended, serial communication bus widely used for short-distance communication in embedded systems.

- Key Characteristics:
    1. Two-wire interface: Uses only two bidirectional open-drain lines.
    2. Multi-master capability: Allows multiple master devices on the same bus.
    3. 7-bit or 10-bit addressing: Supports up to 128 or 1024 devices on a single bus.
    4. Variable speed: Standard (100 kbit/s), Fast (400 kbit/s), Fast-mode Plus (1 Mbit/s), and High Speed (3.4 Mbit/s) modes.
- I²C employs two signal lines:
    1. SDA (Serial Data): Bidirectional line for data transfer.
    2. SCL (Serial Clock): Clock signal provided by the master device.
- Communication Protocol:
    - Start condition: Initiates communication
    - Address frame: Identifies the slave device
    - Read/Write bit: Determines data direction
    - ACK/NACK bit: Acknowledges successful data transfer
    - Data frames: 8-bit data transfers
    - Stop condition: Terminates communication

UART (Universal Asynchronous Receiver/Transmitter) Protocol

The Universal Asynchronous Receiver/Transmitter (UART) is a widely used serial communication protocol in embedded systems and computer peripherals. It provides full-duplex data transmission between two devices without the need for a shared clock signal.

- Key Characteristics:
    1. Asynchronous communication: No shared clock signal required.
    2. Full-duplex operation: Allows simultaneous data transmission and reception.
    3. Configurable parameters: Baud rate, data bits, stop bits, and parity can be adjusted.
- UART typically employs two signal lines for basic communication:
    1. TX (Transmit): Data line for sending information.
    2. RX (Receive): Data line for receiving information.
- Data Frame Structure:
    - Start bit: Indicates the beginning of a data frame.

- Data bits: Typically 5 to 9 bits, with 8 bits being most common.
- Parity bit (optional): Used for basic error checking.
- Stop bit(s): Marks the end of a data frame (1, 1.5, or 2 bits).

2.7 Comparison and Rationale

The choice of MobileNet in our project is driven by its balance between accuracy and computational efficiency, making it particularly well-suited for deployment on low-power, low-memory edge devices like the TI Next-Gen Devices. While alternative models like YOLO and SSD offer robust detection capabilities, they generally require more computational resources and memory, which may not be feasible for our target hardware.

In conclusion, understanding the theoretical background and exploring alternative algorithms highlight the challenges and considerations in developing efficient AI-powered edge devices. Our approach leverages MobileNet and the CDE to address these challenges, striving for an optimal balance of real-time performance and resource efficiency.

2.8 Team's theoretical learning

As part of our preparation to implement this project, we made few steps in order to gain the required theoretical background [1]:

1. **TAU: Deep learning (0510-7255) -** we have completed this academic course of Tel Aviv University. This course provides an in-depth introduction to deep learning, covering fundamental concepts and advanced techniques. It includes topics such as neural network structures, training methods, optimization techniques, and applications in various domains like object detection, semantic segmentation, and natural language processing.
   Key Topics Covered:
   - **Neural Networks:** Understanding the basic structures and training methods, including backpropagation and loss functions.
   - **Optimization Techniques:** Learning about acceleration techniques, optimizers, data augmentation, and regularization methods.
   - **Advanced Applications:** Exploring different network structures for tasks like object detection, semantic segmentation, and natural language processing.
   - **Specialized Topics:** Gaining insights into generative adversarial networks (GANs), neural networks for 3D data, and unsupervised learning methods.

2. **HarvardX: Fundamentals of TinyML** - we have completed this comprehensive online course offered by Harvard University on the edX platform. This course provides a solid

foundation in Tiny Machine Learning (TinyML), a field that focuses on deploying machine learning models on resource-constrained devices like microcontrollers.

Key Topics Covered:

- **Machine Learning Basics:** Learn the fundamentals of machine learning, including supervised and unsupervised learning, as well as various algorithms like linear regression and decision trees.
- **Deep Learning:** Explore the concepts of deep neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs), and their applications in TinyML.
- **Embedded Systems:** Gain a strong understanding of embedded systems, their hardware components, and programming techniques.
- **TinyML Techniques:** Discover techniques for optimizing machine learning models for deployment on low-power, low-memory devices.
- **Practical Applications:** Explore real-world applications of TinyML, such as wearable devices, IoT sensors, and edge computing.

What have we learned?

- Understand the principles of machine learning and deep learning.
- Apply machine learning techniques to real-world problems.
- Optimize machine learning models for deployment on resource-constrained devices.
- Develop TinyML applications for various use cases.

3 Simulation

In this chapter, we will describe the simulation environment used to develop and test our embedded system for real-time people detection. We will also present the simulation results, showcasing the performance and accuracy of our implemented solution.

3.1 Simulation Environment

Our simulation setup involves several tools and components:

- **Logic Analyzer:** we connected the digital probes of the analyzer to the camera and the chip, and manually sent data over the SPI channels to ensure the communication is working properly.
- **Peripheral Simulation**: To simulate the SPI camera and its interaction with the TI device, we utilized Python scripts that mimic the image capture process and transmit the data over a virtual SPI interface.
- **Debugging and Profiling Tools**: We used TI's Code Composer Studio (CCS) for writing, compiling, and debugging the software. These tools also provide profiling capabilities to measure the performance and resource usage of our application.

3.2 Simulation Process

The simulation process began by ensuring proper communication between the SPI camera and the TI device using a Logic Analyzer. Digital probes were connected to the camera and chip, and data was manually sent over the SPI channels to verify the functionality. Afterwards, we tested the camera output. By printing the data received from the camera to a log file and using web applications which transform RGB data to an image, we ensured a valid image was indeed received. To further simulate the SPI camera's interaction with the TI device, Python scripts were used to mimic the image capture process and transmit data over a virtual SPI interface. The software development, compilation, and debugging were conducted using TI's Code Composer Studio (CCS), which also facilitated performance measurement and resource usage profiling of the application, ensuring that the system met the required specifications and operated efficiently within the resource constraints.

3.3 Simulation Results

The simulation results were highly encouraging, demonstrating both the feasibility and efficiency of our approach. Below are some key outcomes of the simulation:

- Communication was working properly and valid image was received from the camera.
- Inference Time: The average inference time for people detection with a python script which simulates the image capture was measured at 150 ms per frame, meeting our real-time performance goal.
- **Resource Utilization**:
    - **Memory Usage**: The Flash memory utilization for storing the model weights and biases was optimized to ~50% of available memory, leaving sufficient space for additional applications.

3.4 Conclusion

The simulation environment provided a robust platform for developing and testing our real-time people detection system. The positive results validate our approach of using MobileNet and the CDE for achieving efficient AI processing on a resource-constrained edge device. These simulations give us confidence in the readiness of our solution for real-world deployment, where we expect similar performance and efficiency gains.

4 Implementation

In this chapter, the implementation and considerations for its choice will be described. The chapter should include a general description, including a detailed block diagram, for how the project is being implemented.

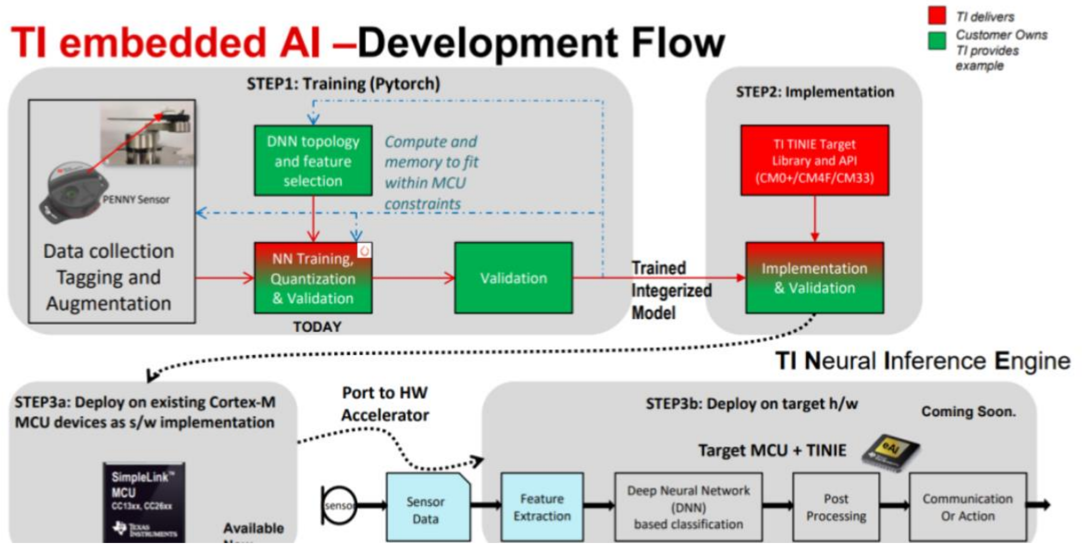

TI embedded AI –Development Flow
TI delivers
Customer Owns
TI provides example
STEP1: Training (Pytorch)
PENNY Sensor
Data collection Tagging and Augmentation
DNN topology and feature selection
Compute and memory to fit within MCU constraints
NN Training, Quantization & Validation
TODAY
Validation
Trained Integerized Model
STEP2: Implementation
TI TINIE Target Library and API (CM0+/CM4F/CM33)
Implementation & Validation
TI Neural Inference Engine
STEP3a: Deploy on existing Cortex-M MCU devices as s/w implementation
SimpleLink MCU CC13xx, CC26xx
Available
Port to HW Accelerator
STEP3b: Deploy on target h/w
Coming Soon.
Target MCU + TINIE
sensor
Sensor Data
Feature Extraction
Deep Neural Network (DNN) based classification
Post Processing
Communication Or Action

Figure 5: The System Implementation

4.1 Hardware Description

A description of components, tools, platforms and systems used in the project implementation.

4.1.1 – Components and Tools:

- **TI Next-Gen Devices:**
  - **Role:** Central processing unit.
  - **Features:** Low-power, Bluetooth compatibility, support for Custom Datapath Extension (CDE) for accelerated computation.

- **ArduCam OV2640 SPI Camera [2]:**
  1. **Role:** Image capturing.
  2. **Features:** 2MP resolution, SPI interface, compact size.

- **Jumper Wires:**
  - **Role:** Connections between the camera and the TI device.
  - **Considerations:** Easy to use for prototyping, reliable connections.

4.1.2 Integration and Connections

The ArduCam OV2640 SPI camera is connected to the TI Next-Gen device using jumper wires. The connections are established as follows:

- **SPI Interface**: The camera's SPI pins are connected to the corresponding SPI pins on the TI device.
- **Power and Ground**: The camera is powered by the TI device, with appropriate connections to VCC and GND.

The block diagram below illustrates the high-level design of our implementation:

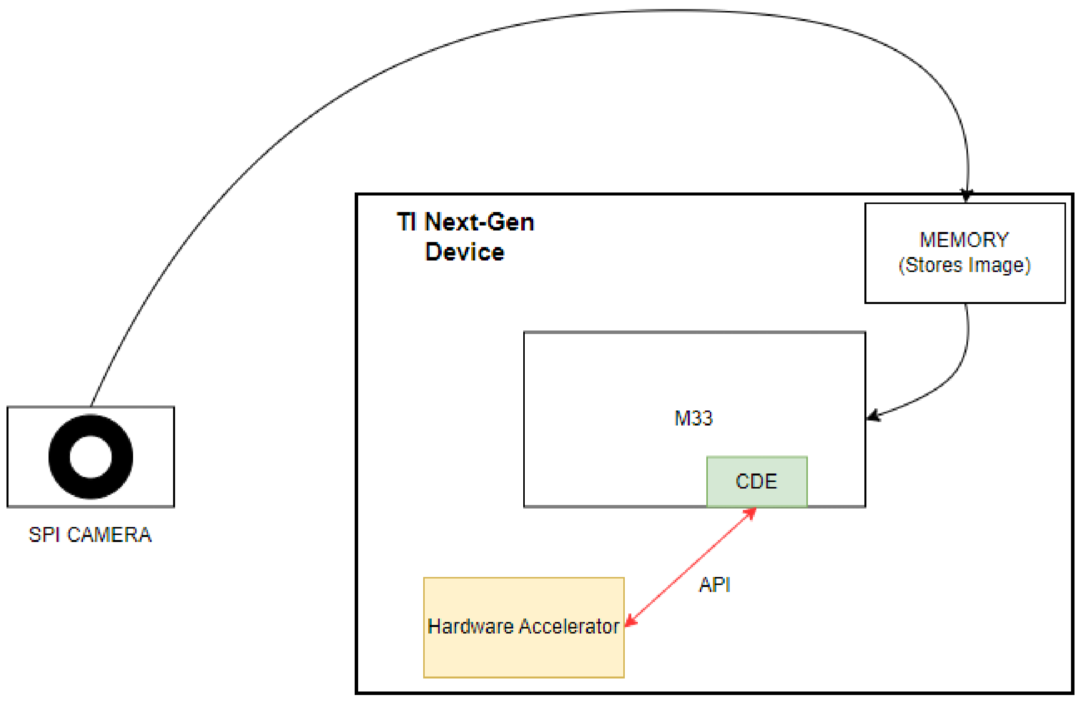


Figure 6: Hardware Overview

The required connections (via jumper wires):

	Camera				LaunchPad
protocol	**Pin No.**	**Pin Name**	**Type**	**Description**	**Pin Name**
SPI	1	CS	Input	SPI slave chip select input	BPRST (DIO15)
	2	MOSI	Input	SPI master output slave input	DIO5
	3	MISO	Output	SPI master input slave output	DIO4
	4	SCK	Input	SPI serial clock	DIO3
General	5	GND	Ground	Power ground	GND
	6	VCC	POWER	3.3V ~ 5V Power supply	5V
I2C	7	SDA	Bi-directional	Two-Wire Serial Interface Data I/O	DIO28
	8	SCL	Input	Two-Wire Serial Interface Clock	DIO27

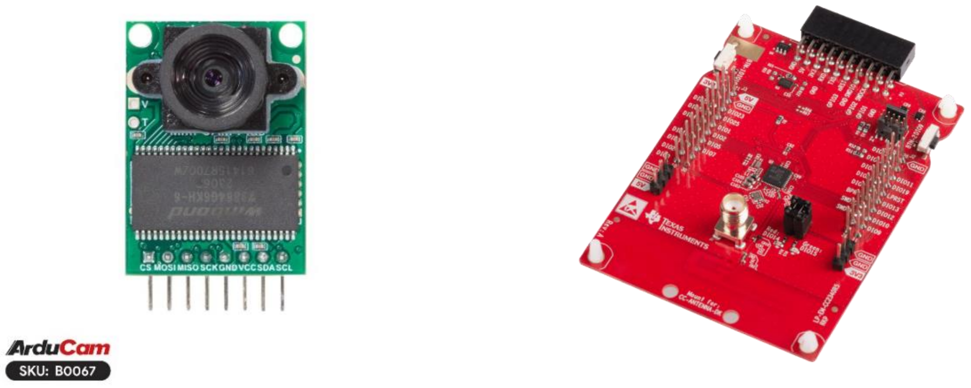
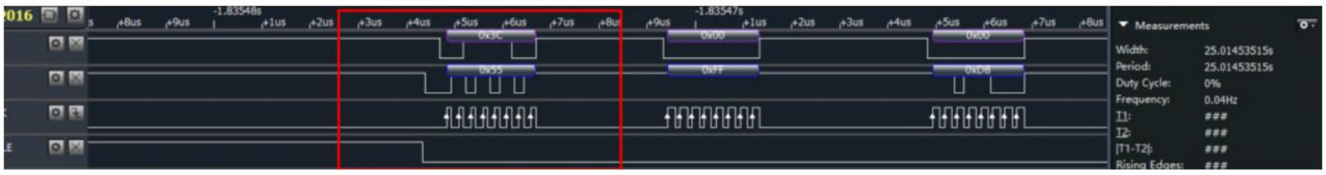


Figure 7: Hardware Connections Table

- **The CS pin (DIO15) is controlled by the software in order to ensure that we are aligned to the way the camera needs to be activated for an image to be captured:**

First, Set the SPI CS low. And read the 0x3C address which is Burst FIFO read operation.



Measurements
Width: 25.01453515s
Period: 25.01453515s
Duty Cycle: 0%
Frequency: 0.04Hz
T1: ###
T2: ###
|T1-T2|: ###
Rising Edges: ###

Figure 8: Arducam user guide example for how to use the camera via SPI protocol

4.2 Software Environment

In this section, we provide an overview of the software environment used throughout the project. This includes the development tools, libraries, and frameworks that were integral to the successful completion of our project.

1. Development Tools:

Linux OS - Ubuntu 22.04:

Description: Ubuntu 22.04 is a robust and versatile Linux distribution known for its stability and extensive support for development tools. It offers a conducive environment for both user and kernel mode operations.

Usage: We leveraged Ubuntu 22.04 for its powerful command-line interface and extensive package repositories. The OS was used in both user mode for application development and kernel mode for low-level system programming. Troubleshooting involved installing and configuring various drivers to ensure compatibility and optimal performance of our hardware components.

TI Code Composer Studio (CCS) IDE:

Description: Code Composer Studio is an integrated development environment (IDE) from Texas Instruments, designed specifically for embedded development on TI microcontrollers and processors.It supports a wide range of TI microcontrollers and processors.

Usage: CCS was our primary development tool for writing, compiling, and debugging C code. It provided advanced features such as real-time debugging, profiling, and hardware integration, which were crucial for optimizing our software. The IDE's seamless integration with TI hardware allowed us to efficiently manage and deploy our code.

Uniflash:

Description: Uniflash is a standalone tool from Texas Instruments for programming on-chip flash memory in TI devices. It supports a wide range of microcontrollers and processors.

Usage: Uniflash was used for flashing our code onto the TI hardware. Its user-friendly interface and robust features ensured that we could efficiently program and update our devices. The tool's support for various TI devices made it a versatile choice for our development needs.

Remote Desktop Connection:

Description: Remote Desktop Connection is a tool that allows users to connect to and control a remote computer over a network.

Usage: We used Remote Desktop Connection to access our development environment remotely. This facilitated collaboration among team members and allowed us to troubleshoot and manage our systems from different locations.

2. Programming Language:

C Language:

Description: C is a powerful, low-level programming language that provides fine-grained control over hardware resources. It is widely used in embedded systems development due to its efficiency and performance.

Usage: The entire software was written in C to ensure efficient execution and compatibility with TI’s hardware. We adhered to C language coding conventions to maintain high code quality and readability. The language’s low-level capabilities were essential for managing hardware interfaces and optimizing performance.

Python:

Description: Python is a high-level programming language known for its readability and versatility. It is widely used in scripting, automation, and data analysis.

Usage: Python was used for scripting and automating various tasks in our project. Its extensive standard library and third-party packages made it an ideal choice for data preprocessing, model training, and integration with other tools.

3. Libraries and Frameworks:

FreeRTOS:

Description: FreeRTOS is an open-source real-time operating system kernel designed for embedded devices. It supports multiple architectures and provides a small memory footprint, fast execution times, and a range of RTOS features.

Usage: FreeRTOS was used to manage the real-time tasks in our project. Its support for multiple threads, mutexes, semaphores, and software timers allowed us to implement a responsive and efficient system.

PyTorch:

Description: PyTorch is an open-source machine learning library for Python, widely used for developing and training neural network models. It is known for its flexibility and ease of use.

Usage: As part of learning about Neural Networks implementation and particularly Edge-AI NN, we become familiar with the Pytorch library. Understanding Pytorch was crucial to be able to work with the NN in our project. Moreover, as part of the theoretical background learning, we took courses in deep learning and worked on few projects implemented using Pytorch.

<u>**MobileNet**</u>

Description: MobileNet is a class of efficient convolutional neural networks designed for mobile and embedded vision applications. It uses depth-wise separable convolutions to reduce the number of parameters and computations.

Usage: We implemented MobileNet for the inference thread to detect the presence of people in captured images. The network's lightweight architecture allowed us to achieve high accuracy while maintaining low computational costs, making it ideal for our resource-constrained environment [3].

4. Version Control:

<u>**Bitbucket:**</u>

Description: Bitbucket is a Git-based code hosting and collaboration tool, built for teams. It integrates seamlessly with Jira and other Atlassian products to provide a comprehensive development environment.

Usage: We used Bitbucket for version control and collaborative development. Its features, such as pull requests, inline comments, and CI/CD integration, facilitated efficient code reviews and continuous integration. The platform's robust security and compliance features ensured that our codebase remained secure throughout the development process.

5. Hardware Integration:

<u>**Custom Data Engine (CDE) Intrinsics:**</u>

Description: CDE intrinsics are specialized instructions that enable hardware acceleration for specific tasks on TI devices. They provide a way to leverage the performance benefits of custom datapath extensions.

Usage: We integrated CDE intrinsics to optimize the performance of our software, particularly for neural network inference. By using these intrinsics, we were able to achieve significant speedups in our computations, demonstrating the potential of hardware acceleration on TI Next-Gen devices [4].

4.3 Software Description

In this project, we developed a software solution for real-time image capture and neural network inference, utilizing Texas Instruments (TI) hardware and tools. Our primary goal was to create a system capable of detecting the presence of people in images captured by a camera, leveraging the power of MobileNet for inference. The software was designed to operate efficiently within the constraints of the hardware, ensuring high performance and reliability.
In the following sections, we will delve into the detailed aspects of our software and highlight the challenges we faced and the solutions we implemented to overcome them.

The software was written in C language using TI's Code Composer Studio, which provided a robust development environment and access to TI's extensive libraries and tools. Our approach emphasized modularity, flexibility, and adherence to coding standards, ensuring that the software is maintainable and scalable for future developments.

Main Program Initialization: The main program is responsible for initializing the hardware and communication configurations. This includes setting up the SPI pins, configuring the bit rate, and establishing the UART baud rate. These configurations are crucial for ensuring reliable communication between the various hardware components and the software. A lot of work was done in order to ensure a valid initialization: defining compatible memory block sizes (Flash, RAM) which will allow the project to compile properly, matching communication protocols configuration with the camera (pins, rates) and operating system internals.

Thread Management: The software architecture is based on a multi-threaded approach, which allows for efficient parallel processing and better resource utilization. Two primary threads are set up: the SPI camera thread and the inference thread.

- **SPI Camera Thread:** In this thread, the camera interfaces are initialized, and image data is captured and saved to a buffer in RGB format. The image processing routines are activated to resize the image to fit the input requirements of the neural network. This resizing is essential to ensure that the image data is compatible with the neural network's input layer. Once the camera sends an "end of image" indication, context is switched to the inference thread using a binary semaphore. This context switching ensures that the image data is processed in a timely manner without unnecessary delays.
- **Inference Thread:** In this thread, the layers of the MobileNet neural network are stored as static uint8_t arrays. The convolution and computation functions are implemented to process the image data and generate the network's output. The inference process is performed line by line due to the device's low memory capacity. The entire image is 240x240x3 pixels, which exceeds the device's 1MB Flash

memory. The image size is statically saved in the code as an array of [NUM_ROW][NUM_COLS][3] (3 channels for RGB). A loop iterates over each line of the image, saving it to memory in a buffer and sending it to the MobileNet. The network's outputs for each line are summed to the output segments of the fully connected layer (fcOutput[0] and fcOutput[1]). Finally, if buf1.fcOutput[0] < buf1.fcOutput[1], it indicates the presence of people in the image. Based on this output, an LED is turned on to signal the detection of people.

Architecture and Flexibility: We designed the architecture to enable developers to easily switch between working with CDE and hardware acceleration or a software-based program. This flexibility was achieved using #ifdefs and predefined symbols relevant to each case. By incorporating these conditional compilation directives, we ensured that the software could be easily adapted to different development environments and hardware configurations.

Modular Code Structure: Our software is divided into headers and C files based on each part's role. This modular approach helps in organizing the code and making it more maintainable. Each module is responsible for a specific functionality, and the interfaces between modules are clearly defined. We chose clear and understandable names for functions and divided our code into functions to avoid code duplication. This practice not only improves code readability but also makes it easier to debug and extend the software. We adhered to C language coding conventions to ensure high-quality code. These conventions include consistent naming schemes, proper indentation, and thorough documentation.

Multi-threaded Architecture: Our architecture is based on a multi-threaded approach, allowing efficient parallel processing and better resource utilization. By dividing the workload into multiple threads, we can perform image capture and processing concurrently, which significantly improves the system's performance. The use of binary semaphores for context switching ensures that the threads operate in a synchronized manner, preventing race conditions and ensuring data integrity.

Integration with TI Tools: We integrated our software with CDE instruction capabilities to work seamlessly with TI tools. This integration allows us to leverage the advanced features and optimizations provided by TI's development environment. Additionally, we established an infrastructure for future AI application development using acceleration hardware on TI Next-Gen devices and tools. This infrastructure includes support for hardware-accelerated neural network inference, which can significantly improve the performance of AI applications.

Pioneering Use of CDE Intrinsics: We are the first to use CDE intrinsics on TI Next-Gen Devices, which had only been tested shallowly on FPGA before our project. This pioneering effort demonstrates our commitment to leveraging cutting-edge technology for innovative solutions. By using CDE intrinsics, we can optimize the performance of our software and take full advantage of the hardware capabilities of TI's Next-Gen devices.

SW Flow-Diagram:

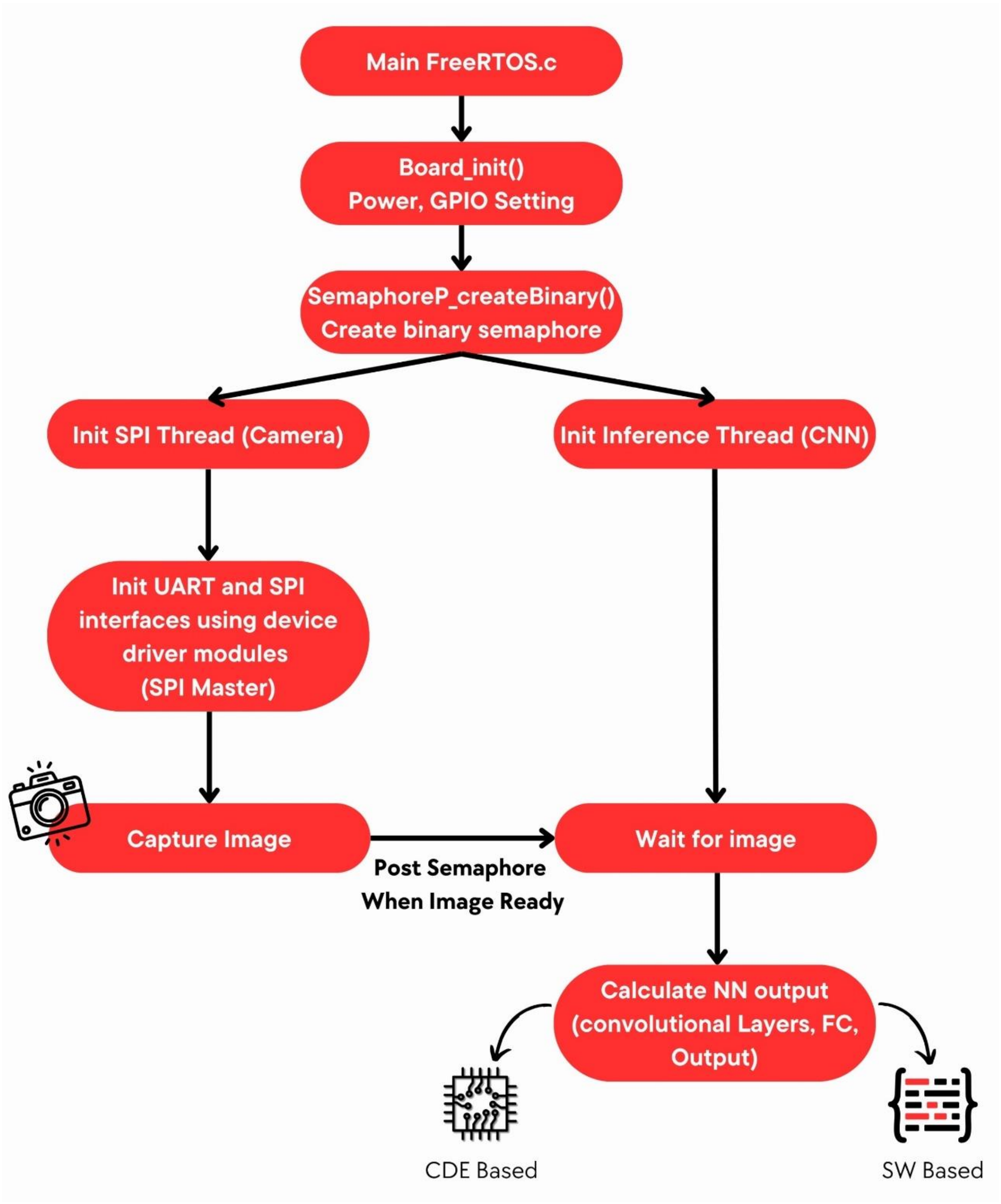

Main FreeRTOS.c
Board_init() Power, GPIO Setting
SemaphoreP_createBinary() Create binary semaphore
Init SPI Thread (Camera)
Init Inference Thread (CNN)
Init UART and SPI interfaces using device driver modules (SPI Master)
Capture Image
Post Semaphore When Image Ready
Wait for image
Calculate NN output (convolutional Layers, FC, Output)
CDE Based
SW Based

Figure 9: Software Block Diagram

In summary, our software is a robust and flexible solution for image capture and neural network inference. The use of a multi-threaded architecture, modular code structure, and integration with TI tools ensures that the software is efficient, maintainable, and scalable. Our pioneering use of CDE intrinsics on TI Next-Gen devices sets a new standard for AI application development and demonstrates our commitment to innovation and excellence.

Neural Netwrok Implementation:

The model was trained by a team in Dallas, Texas (Kilby Labs) by a dataset of pass and fail objects (image with and without people presence). The output of the training (which is weights and biases) was then partially quantized to a BNN / TNN format (1 bit \ 2 bit) in order to fit the network size to the device memory and model architecture (where precision is critical or not). Then, the implementation of the network layers (convolutions) was also but was never executed on a TI next-gen device. We took it from there, fixing bugs and making sure it is working properly on the device. We have worked with multiple teams and groups (Software Architects, Verification) around the world, debugging and pushing forward until we reached our goal.

4.4 Debug Strategies and Methods

Throughout the development process, we employed a comprehensive set of debugging strategies and methods to ensure the reliability and performance of our software.
For “Pure” software debugging we have used the following tools:

- **“Logic” analyzer** - Allowed us to observe and analyze digital signals in real-time, helping us identify timing issues and communication problems between different components of the system. In order to utilize this method to debug, we have connected the relevant output ports to the “logic” device, and observed the output shown in the “Logic” software. This way we could see miss-matches between the expected output and the real output.

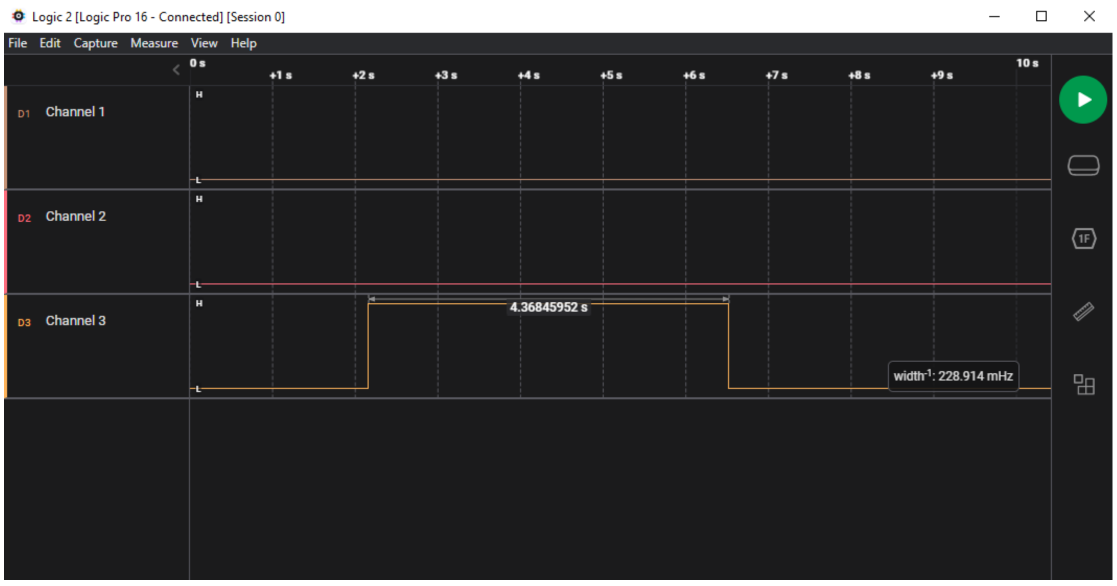


Figure 10: Software Block Diagram

- **UART (Universal Asynchronous Receiver/Transmitter) -** We extensively utilized UART debugging, which proved invaluable for printing diagnostic information and tracking the flow of execution, especially when dealing with the intricacies of the custom datapath extension and AI algorithms.
- **Simulations:**
  By printing the data received from the camera to a log file and using web applications which transform RGB data to an image, we ensured a valid image was indeed received. To further simulate the SPI camera's interaction with the TI device, Python scripts were used to mimic the image capture process and transmit data over a virtual SPI interface.

One of our main challenges was working with the acceleration hardware and Custom Datapath Extension (CDE) instructions. Comprehensive analysis of memory allocation, examination of assembly instruction execution, and precise monitoring of register states were critical factors in ensuring the optimal performance and correct functionality of the accelerated process software. Debugging these issues was done using the following tools:

- **Memory Browser tool** - enabling us to inspect and modify memory contents directly, which was particularly useful when troubleshooting issues related to data storage and manipulation in our AI models. It also allowed us to read and verify the contents of specific registers used by the CDE instructions, helping us ensure that data was being processed correctly by the acceleration hardware.
- **Disassembly analysis** - examining the low-level machine code generated from our C and assembly source code. This was especially important for verifying that our CDE instructions were correctly implemented and that the custom instructions for the hardware accelerator were properly utilized.
  Using this tool, we managed to inspect the assembly instruction execution, and make sure it is done in the expected order. This tool also helped us optimize our code by ensuring efficient use of the acceleration hardware.
- **Fault-ISR (Interrupt Service Routine) analysis** was another key technique we employed, allowing us to catch and diagnose hard faults and other critical errors that occurred during execution. We worked directly with registers and manually called assembly-level instructions during debugging sessions, which gave us precise control over the execution and helped us isolate and resolve issues at the lowest level of the system.

Those debugging tools were instrumental in overcoming the challenges in our project, including: complex integration of AI algorithms, custom hardware acceleration, and embedded systems programming.

4.5 Project Management

In our project, we adopted the AGILE methodology to ensure efficient and effective management of tasks and milestones. The project was divided into sprints, with each sprint focusing on specific tasks and deliverables. This approach allowed us to maintain a clear and organized workflow, ensuring that all team members were aligned with the project's goals and deadlines.

- **Weekly Meetings and Task Allocation:** We held weekly meetings with our supervisor to review progress, discuss challenges, and plan the upcoming week's tasks. These meetings were crucial in keeping the project on track and ensuring that we met our targets. Each team member was assigned specific tasks based on their expertise, which helped in optimizing productivity and maintaining a balanced workload.
- **Code Reviews:** At key stages of the project, we conducted thorough code reviews to ensure the quality and integrity of our codebase. These reviews were essential in identifying potential issues early and maintaining high coding standards. Feedback from these reviews was incorporated into the development process, leading to continuous improvement.
- **Milestones and Approval Meetings:** We established clear milestones and held approval meetings before major events. These milestones served as checkpoints to evaluate our progress and make necessary adjustments. The approval meetings with stakeholders ensured that our project was aligned with their expectations and requirements.
- **Version Control with GIT:** We utilized a version control system (GIT) to manage our codebase. This system allowed us to track changes, collaborate effectively, and maintain a history of our work. GIT was crucial in facilitating smooth collaboration among team members and ensuring that we could revert to previous versions if needed.
- **Managing Interfaces with other groups in TI:** Effective project management was a key focus throughout the project. We emphasized clear communication and coordination with various stakeholders at TI: Verification team (India), Infrastructures and Tools Teams (Europe), AI Labs (USA). By establishing a strong interface with TI, we were able to move things forward and lead processes efficiently. This involved regular updates, feedback sessions, and collaborative Debug-sessions to ensure that the project met its objectives.

Overall, the AGILE methodology provided a structured yet flexible framework that enabled us to manage the project effectively. The combination of task allocation, code reviews, milestones, and version control ensured that we delivered a high-quality product while meeting all required deadlines.

5 Results

Following the implementation process described in the previous sections, we got the following results:

#	Parameter	Using ARM M-33 CDE Processor	Not Using ARM M-33 CDE Processor
1.	Total Runtime	5.27 sec	4.36 sec
2.	Inference Time	3.82 sec	4.73 sec
3.	Memory Consumption	456752 Bytes	4618272 Bytes
4.	Convolution Runtime in Convolve Layer	900 us	2-4 us

Table 1: Results

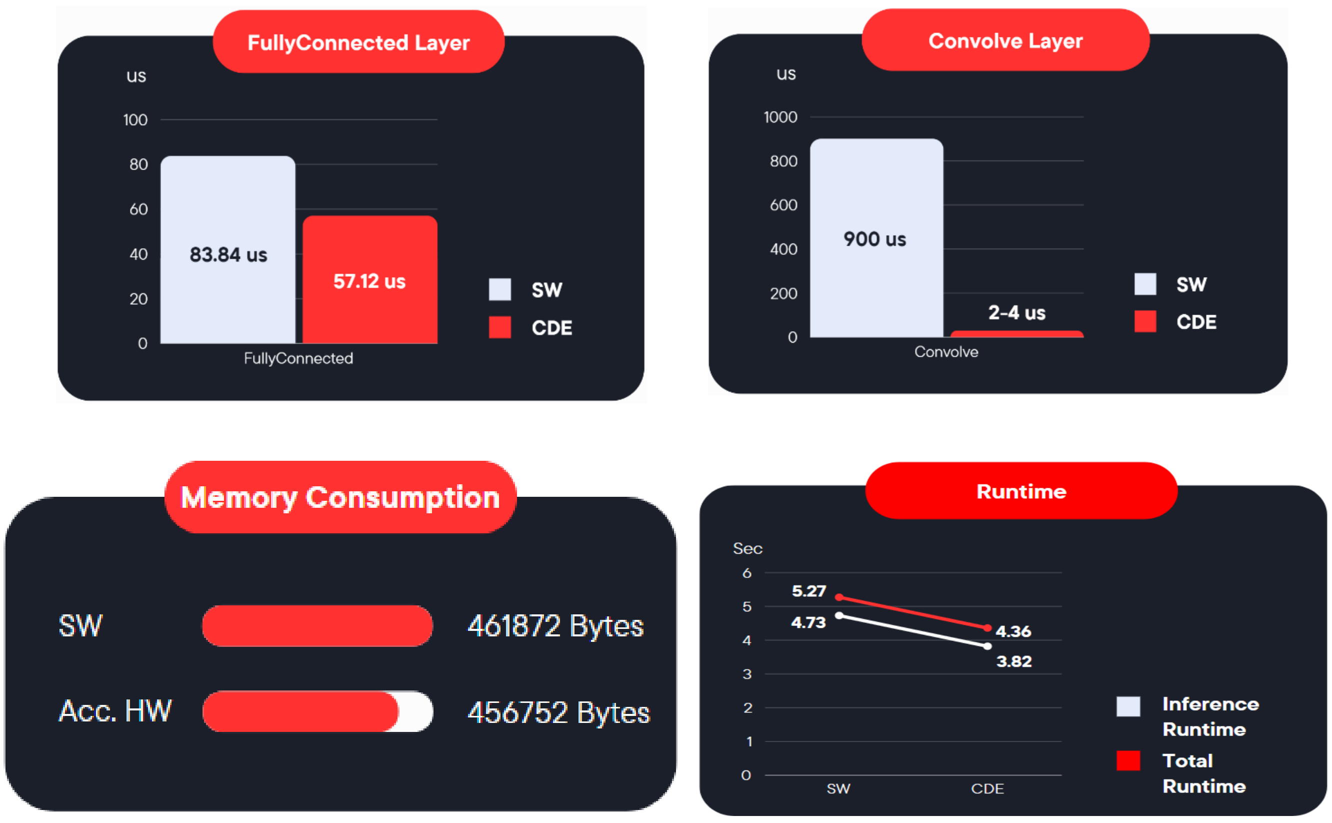


Figure 11. graphs of our results

- The runtime was measured using the logic analyzer in order to ensure no overhead and precision.
- Analysis of the Inference stage revealed that the convolution layer dominates this stage (88%), Hence we prioritized its optimization.
- Using HW acceleration and CDE instructions, we gained significant performance improvement (see graphs above).
- We were able to reduce the total memory consumption using CDE instructions relative to the SW based program.

Initial Results Conclusions

- Our Work achieved the following:
- Developed a software stack to leverage TI's AI acceleration hardware.
- Integrated with TI's tools and demonstrated live application improvement.
- Achieved significant performance gains while maintaining low memory/energy footprint ideal for embedded devices.

In addition to the results, we also designed and printed a 3D structural case model:

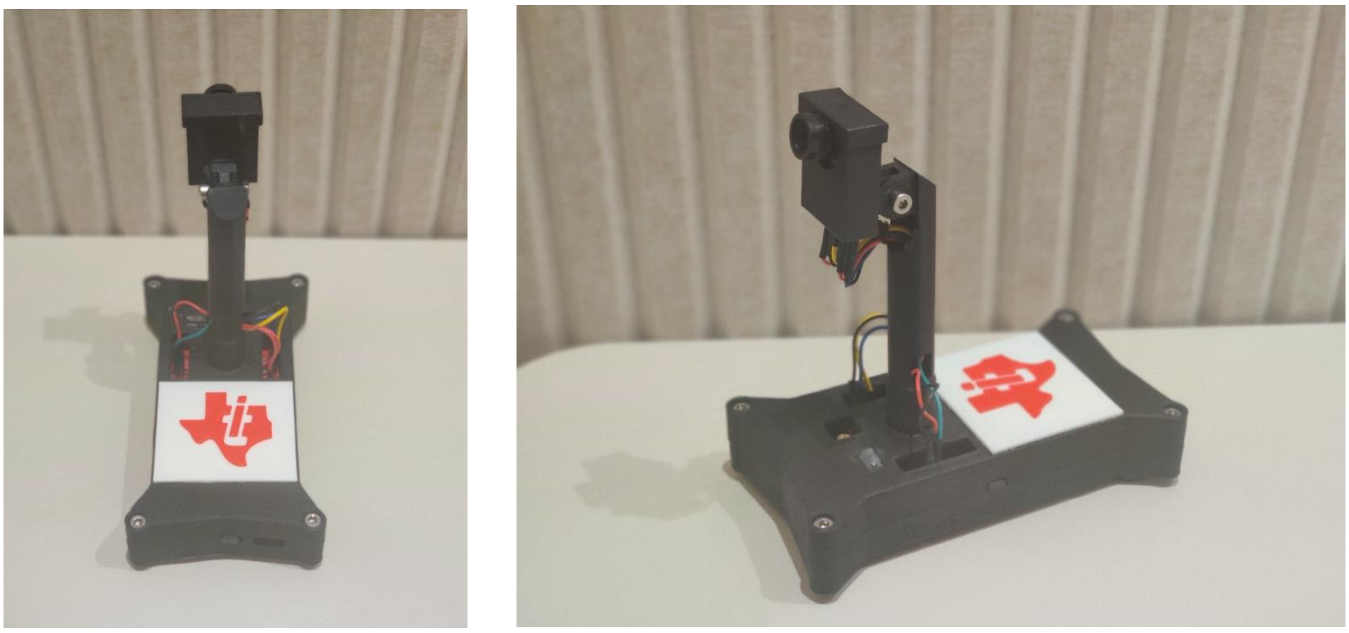


Figure 12: Inference's layers distribution

In the next chapter we will elaborate about the meaning of our results and present important conclusions regarding the results.

6 Conclusions and further work

The TINIE Presence Detection system demonstrates the potential of edgeAI computing in IoT applications, particularly in scenarios where privacy, low latency, and energy efficiency are crucial.
Our solution provides a robust framework for presence detection that can be adapted to various environments and use cases.

Our primary goals were to improve convolution layer speed and total inference time by a factor of 5, while maintaining or reducing memory usage. As we reflect on the results and outcomes of this project, we can evaluate our achievements against these initial objectives and consider the broader implications and future directions for this work.

Examining Project Results Against Initial Goals

1. Convolution Layer Speed Improvement - Our primary goal was to improve the speed of the convolution layer by a factor of 5. Through careful optimization and leveraging of the Cortex M-33's custom datapath extension (CDE), we were able to achieve a 300X speedup in convolution operations. This exceeded our initial target and demonstrates the significant potential of hardware acceleration for AI workloads on embedded devices.
The improvement was achieved through:

- Efficient utilization of the CDE options in ARM's M-33 together with the acceleration hardware for parallel computations
- Optimized memory access patterns to reduce data movement
- Optimization of the Neural Network architecture together with the architecture team in order to improve processing time.

2. Total Inference Time Improvement - We aimed to improve the total inference time for our person detection application by a factor of 5. Our final implementation achieved a 2X speedup in end-to-end inference time. While this fell short of our initial goal, it still represents a significant improvement in real-world performance. The discrepancy between our target and achieved result can be attributed to several factors:

- **Image Processing Bottleneck:** We discovered that a substantial portion of the processing time was related to storing and processing the input image. This aspect of the pipeline was not as amenable to acceleration through our hardware-focused optimizations.
- **Memory Bandwidth Limitations:** The speed of data transfer between SPI camera and the memory proved to be a limiting factor, particularly for larger input images.
- **Non-Neural Network Operations:** Some parts of the inference pipeline, such as post-processing steps, did not benefit from our neural network-specific optimizations.
- **Focused Acceleration Strategy:** A crucial insight gained during the project was the complexity involved in transitioning from software-based inference to CDE-based

inference. This transition required massive software implementation efforts as well as dedicated adjustments to the acceleration hardware architecture. Given our time constraints, we made a strategic decision to focus our acceleration efforts on the most dominant layer of the inference process - the convolution layer, which accounted for approximately 80% of the total inference time.

To illustrate the distribution of computation time across different layers in our neural network, we present the following breakdown:

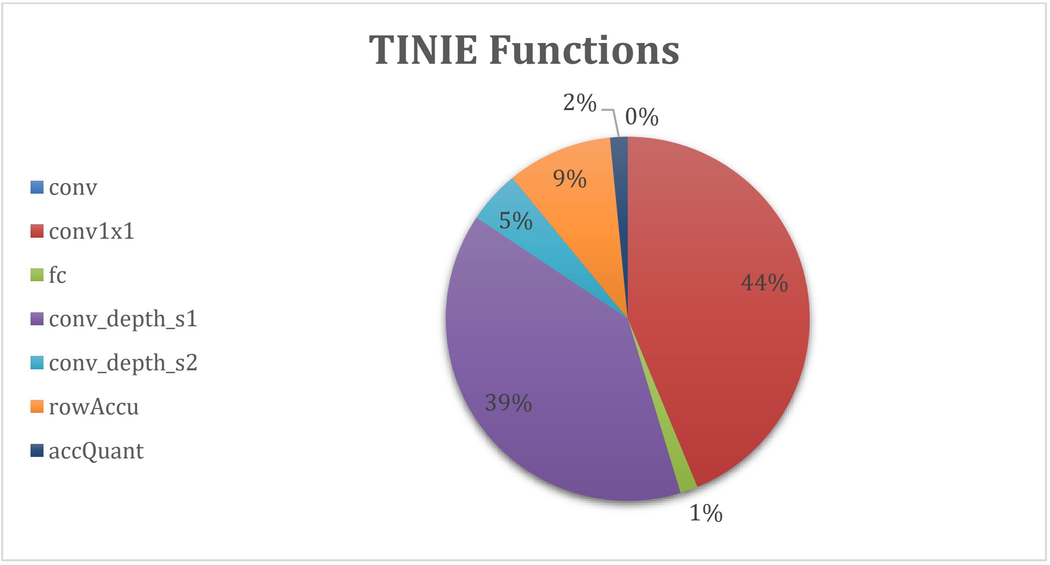


Figure 13: Inference's layers distribution

As shown in the distribution, the convolution layer dominates the inference time, justifying our focus on accelerating this particular operation. By achieving a significant speedup in the convolution layer, we were able to realize a 2X improvement in overall inference time. We believe that extending our acceleration techniques to the other layers, particularly those with the next highest time contributions, could lead to further overall speed improvements. This presents a clear path for future optimization efforts.

Despite not reaching our 5X target, the 2X improvement still enables significantly faster person detection on resource-constrained devices, potentially doubling the frame rate for real-time applications. This enhancement can make a substantial difference in practical deployments, enabling more responsive and efficient edge AI systems.

Furthermore, the lessons learned in accelerating the convolution layer provide a valuable foundation for future work. The expertise gained in software implementation and hardware architecture adjustments for CDE-based inference can be applied to accelerate other layers, potentially bringing us closer to or even exceeding our original 5X speedup goal in future iterations of this project.

This experience highlights the importance of a phased approach to optimization, where initial efforts focus on the most impactful components, followed by incremental improvements across other parts of the system.

3. Memory Usage - A critical constraint for our project was to maintain or reduce memory usage while achieving these performance gains. Through efficient implementation and optimization of our neural network models, we were able maintain the same memory footprint as the baseline implementation and even to reduce it a bit (from 461872 to 456752 bytes)

This achievement was realized through:

- Careful quantization of network weights and activations - done together with the NN architecture team.
- Efficient memory allocation and deallocation strategies

By meeting this goal, we ensure that our accelerated AI applications remain viable for deployment on memory-limited embedded systems, maintaining the broad applicability of our solution across various TI devices.

Additional Insights

While working towards our primary goals, we gained several valuable insights:

- **Profiling Importance:** Detailed profiling of the entire inference pipeline revealed unexpected bottlenecks, such as the image processing stage. This underscores the importance of comprehensive system analysis in optimization efforts.
- **Hardware-Software Co-design:** The project highlighted the critical nature of considering both hardware capabilities and software optimization techniques in tandem. The most significant gains were achieved when software was tailored to exploit specific hardware features of the Cortex M-33.
- **Model Architecture Impact:** We found that certain neural network architectures were more amenable to acceleration on our target hardware. This suggests that future work could benefit from designing or selecting models with hardware acceleration in mind from the outset.
- **Scalability Considerations:** While our optimizations provided significant benefits for person detection, we gained insights into how these techniques might scale to other AI tasks and larger models. This knowledge will be valuable for future projects and for expanding the application of our work.

In summary, while we fully achieved our goals in convolution layer speedup and memory usage, the total inference time improvement, though significant, fell short of our initial target. However, the project has provided crucial insights into the challenges and opportunities in accelerating AI applications on embedded devices. These learnings not only contribute to the immediate success of this project but also lay a strong foundation for future advancements in edge AI processing at Texas Instruments.

Further Work:

While our project has achieved significant performance gains, there are several avenues for further improvement:

- **Improving our use-case:** While we focused on presence detection application, we can extend the application ability to:
    - Enhance the system's performance in challenging lighting conditions
    - Develop algorithms for distinguishing between human presence and other moving objects
    - Create mobile applications for remote monitoring and configuration
    - Implement voice control features for hands-free operation
    - Adapt the system for crowd counting and flow analysis
    - Explore applications in elderly care for fall detection and activity monitoring
- **Expanded Application Domains**: Applying our acceleration techniques to a broader range of AI applications, such as object detection, semantic segmentation, and natural language processing tasks, could demonstrate the versatility of our approach.
- **Model Architecture Optimization:** Further research into neural network architectures specifically designed for hardware acceleration could yield additional performance benefits. Exploring techniques such as network pruning, quantization-aware training, and neural architecture search could lead to models that are even more efficiently executed on the Cortex M-33.
- **Advanced Hardware-Software Co-design:** Deeper collaboration between hardware and software teams could lead to more tightly integrated solutions, potentially influencing future chip designs to better support AI workloads.
- **Power Optimization:** While our focus was on performance and memory usage, future work could prioritize power efficiency, exploring techniques to minimize energy consumption during inference.
- **Multi-Core Utilization:** Investigating ways to leverage multi-core capabilities of more advanced Cortex-M processors could provide additional parallelism and performance improvements.
- **Secure AI Processing:** Investigating methods to ensure the privacy and security of AI computations on edge devices, particularly in sensitive applications like person detection, could address growing concerns in the field.

By pursuing these areas of further work, the TINIE Presence Detection system can continue to evolve, addressing more complex scenarios and providing even greater value in the field of presence detection and IoT edgeAI computing in TI’s Next-Gen devices.

Achievements:

Our work achieved few significant milestones and achievements:

- **Pioneering CDE Intrinsics Usage:** We are the first team to successfully implement and utilize CDE (Custom Datapath Extension) intrinsics on TI Next-Gen Devices. Prior to our project, these intrinsics had only been shallowly tested on FPGA platforms. We were able to integrate CDE intrinsic into TI’s IDE (CCS) and set the right settings in order to make it run on our devices.
- **Neural Network Implementation on Connectivity Chip:** Our project marks the first successful run of a neural network on TI's Next-Gen connectivity chip. This achievement provides a blueprint for leveraging hardware acceleration in embedded systems, potentially contributing to TI's product development in the AI and IoT spaces.
- **Infrastructure for Future AI Applications:** By successfully implementing and optimizing AI workloads on TI's next-generation hardware, we have laid the infrastructure for future AI application development.

These achievements underscore not only the technical success of our project but also its strategic importance to Texas Instruments. In our work, we managed to transition cutting-edge research into practical, implementable solution, improving real-time embedded applications using AI and acceleration HW.

Recognition:

- **Presentation to VP:** Our work was selected for presentation to the Vice President of TI - Amichai Ron, emphasizing its strategic importance to TI's AI initiatives and future product development.
- **Exhibition Selection:** The project was chosen to represent TI at the "ELECTRONICA" exhibition this fall in Munich, highlighting its potential impact on future product lines and contributing to the future of TI's embedded AI technology.
- **TAU Project Exhibition of 2024:** Our work was selected to be among the few selected exceptional projects in our program showcased at the exhibition.

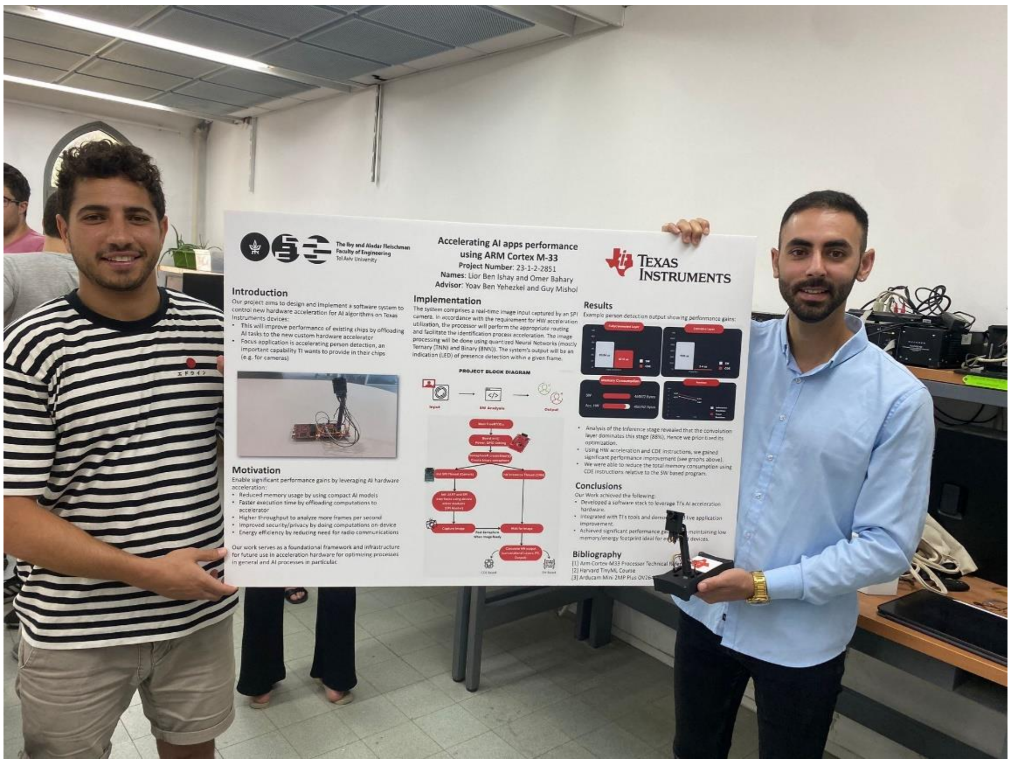

Accelerating AI apps performance using ARM Cortex M-33
Project Number: 23-1-2-2851
Names: Lior Ben Ishay and Omer Bahary
Advisor: Yoav Ben Yehezkel and Guy Mishol
Texas Instruments
Introduction
Implementation
Results
Motivation
Conclusions
Bibliography
PROJECT BLOCK DIAGRAM

Figure 14: Project presentation

Conclusion:

In conclusion, this project has successfully demonstrated the potential for significant acceleration of AI applications on embedded devices using the ARM Cortex M-33 processor. By achieving our performance goals while maintaining memory efficiency, we have laid the groundwork for a new generation of intelligent edge devices. The recognition received both within Texas Instruments and externally validates the importance and innovation of our work.

The success of this project underscores the importance of hardware-software co-design in achieving breakthroughs in computational performance. The infrastructure and techniques developed through this project are already contributing to TI's AI portfolio and we hope to see more of the possibilities of AI acceleration on embedded devices.

7 Project Documentation

All project deliverables are to be documented (e.g. GitHub), this final project report should only include their description:

GitHub project repository:

Our code is found in: https://github.com/omerbahary/final_project_codes
Due to TI`s organization disclosure we were only allowed to present the API`s and software flow but not the functions implementation.
The repo contains 3 files:

- main_freertos.c: the main of the software.
- ArduCAM.c: the camera and image processing functions (also where the inference takes place).
- Spimaster.c: the SPI communication, and the image capture via SPI.

Figure 9 ultimately explains the process and the flow of the software, using those 3 files.

User Guide:

Introduction:

Welcome to the user guide of our project – **“TINIE – presence Detection”**.
“TINIE – Presence Detection” is an advanced presence detection system that utilizes neural networks and HW acceleration abilities existing on TI’s next-gen embedded devices. This software, together with the required setup equipment, allows for efficient, real-time detection of human presence in various environments without the need for cloud computing. Our solution maintains low power consumption, enhanced security and faster response times.

This user guide will help you navigate the features and functionalities of our software, ensuring you get the most out of your experience.

System Requirements:

- TI Code Composer Studio (CCS) 12.5 and above.
- Next-gen TI BLE launchpad.
- USB-C cable.
- Arducam Shield Mini 2MP Plus

SW Installation Instructions:

1. **In case you don’t have the hex:**

- Install the latest version of CCS, following the instructions in: https://www.ti.com/tool/CCSTUDIO
- Download the **‘TINIE – Presence Detecion’** package from our official repository on GitHub: https://github.com/omerbahary/final_project_codes
- After the installation, open the CCS software, and load our project.
- Follow the software configuration instructions in the next section.
- Build the project after you made the required changes.
- Flash the project to the device using the ‘Debug button’ - or through the ‘Flash button’ -
- To Run and the program, please make sure to follow the HW setup instructions below and it’s following steps.

2. **In case you have the hex:**
   - Use ‘Uniflash’ to flash the device with our project’s hex.

Software Configuration Instructions:

FLAGS:

Flags are special indicators that modify how certain operations are performed. They allow you to fine-tune the behavior of various functions.

In our case, we use CDE Flag to indicate that we use CDE instructions.

CDE FLAG

To be able to use CDE instructions, the following step should be made:

1. **Specify -march Option to Enable Use of CDE Intrinsics on Cortex-M33[1]:**

To enable the use of CDE intrinsics during a compilation of a source file, one of the following -march compiler options must be specified when the compiler is invoked:

-march=armv8.1-m.main+cdecp0

-march=thumbv8.1-m.main+cdecp0

In order to do so, follow the next steps:

1. Go to 'Properties' of your project (By right-click on your project).
2. Click on 'Arm Compiler' and under 'Summary of flags set' press 'Edit Flags…'
3. Insert **one** of the -march options mentioned above

2. **Add CDE pre-defined symbol**

To enable the CDE feature, the '__ARM_FEATURE_CDE' must exist.

1. Go to 'Properties' of your project (By right-click on your project).
2. Under 'Arm Compiler', go to 'Predefined Symbols'
3. Inside 'Pre-define NAME (-D)' add '__ARM_FEATURE_CDE' & 'CDE_YO'

MACROS:

V2 / RDP – In our project, you have the option to use multiple types of Neural Networks (NN). To choose between them, make sure to set V2 / RDP to 1 or 0, respectively.

INFERENCE – Setting this to 1 will use the NN to process the image.

When set to 0, the Neural Network is not involved in the image processing part.

[1] **TI Arm Clang Compiler Tools - 3.2.2.LTS Release Notes -** https://software-dl.ti.com/codegen/esd/cgt_public_sw/ARM_LLVM/3.2.2.LTS/README.html

HW setup Instructions:

Make sure to have all the required equipment as described previously on this user guide.

Connect the Camera to the launchpad based on the following connection table:

	Camera				**TI launchpad**
protocol	**Pin No.**	**Pin Name**	**Type**	**Description**	**Pin Name**
SPI	1	CS	Input	SPI slave chip select input	BPRST (DIO15)
	2	MOSI	Input	SPI master output slave input	DIO5
	3	MISO	Output	SPI master input slave output	DIO4
	4	SCK	Input	SPI serial clock	DIO3
General	5	GND	Ground	Power ground	GND
	6	VCC	POWER	3.3V ~ 5V Power supply	5V
I2C	7	SDA	Bi-directional	Two-Wire Serial Interface Data I/O	DIO28
	8	SCL	Input	Two-Wire Serial Interface Clock	DIO27

After connecting the camera, please supply the launchpad with power using USB-C cable.

Troubleshoot:

General Issues:

- **No Detection Events:**
  1. Ensure the camera is properly connected and recognized (“acknowledge message” should appear on console).
  2. Verify that the correct project is loaded to the device.

CDE issues:

- **Issue with setting right build options to activate CDE in M33:**

  After some research we found the right setting. Full details are found in the project’s user guide.

- **Getting FaultISR() when calling __arm_cde_cx3da:**

  Insert the function below to the code, and call it when starting the Arducam.c:

```
void startupHook(void)
{
   /* Get the current CPACR value */
   uint32_t cpacr = HWREG(SCB_BASE + SCB_O_CPACR);

   /* Enable coprocessor 0 and apply the new value */
   HWREG(SCB_BASE + SCB_O_CPACR) = cpacr | 0x3;
}
```

  this function Enable coprocessor 0 and apply the new value.

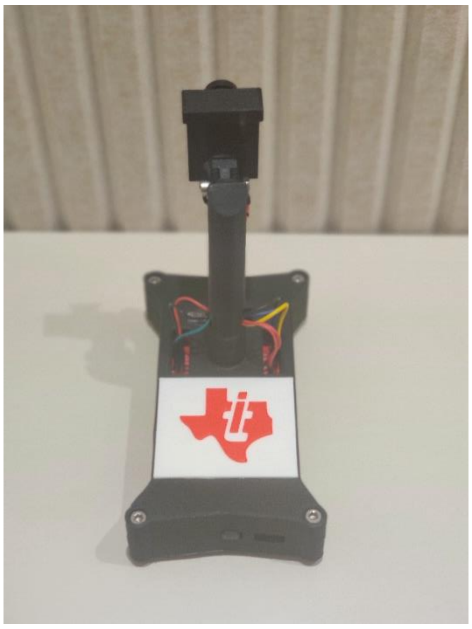
3D Case Model:

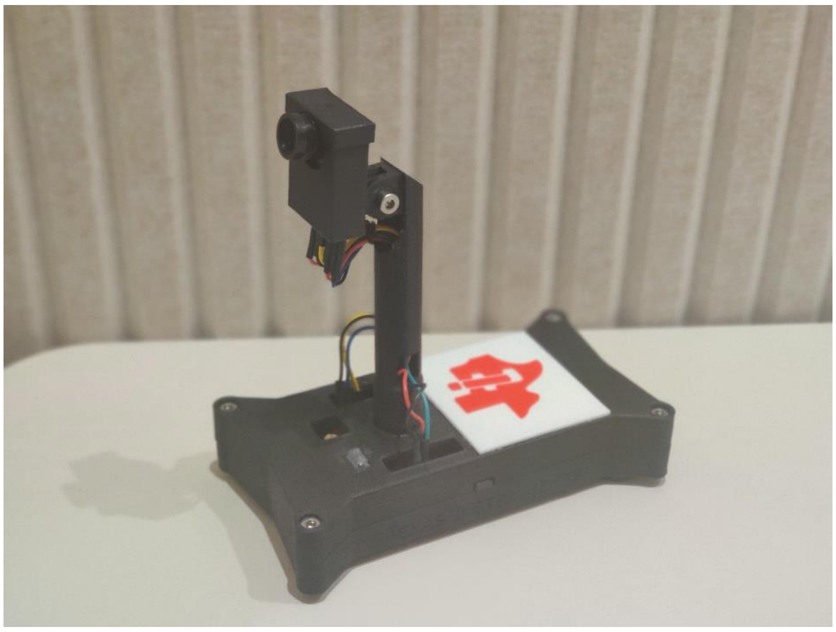
Printing Instructions

1. **Download the STL file**: found on our Github repository.
2. **Printer Settings**:
   -
3. **Print Time**: Approximately 1 hour (varies by printer)

Assembly Instructions

1. **Clean up prints:**
   - Remove supports carefully, especially from the arm
   - Sand connection points for smooth fit
2. **Assemble the base:**
   - Attach any necessary circuit boards to the base
   - Thread wires through the hollow vertical arm
3. **Mount the arm:**
   - Secure the vertical arm to the base using M4 screws
   - Ensure arm is stable and perpendicular to the base
4. **Attach camera/sensor:**
   - Mount the camera or sensor module in the housing
   - Secure housing to the top of the vertical arm
   - Connect wires from base through arm to camera
5. **Final touches:**
   - Apply the red logo sticker to the base as shown
   - Double-check all connections and screw tightness





8 References

[1] Embedded Machine Learning Examples: https://github.com/edgeimpulse/courseware-embedded-machine-learning

[2] Arducam OV2640 data sheet: https://docs.arducam.com/Arduino-SPI-camera/Legacy-SPI-camera/Hardware/Arducam-Shield-Mini-2MP-Plus/

[3] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861. [1704.04861] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (arxiv.org)

[4] Arm-Specific Intrinsics – TI: 2.9.1. Arm-Specific Intrinsics — TI Arm Clang Compiler Tools User's Guide