VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

LOGIC DESIGN PROJECT (CO3091)

Assignment (Semester: 232, Duration: 06 weeks)

Object Tracking Application

Instructor: Tran Ngoc Thinh
Huynh Phuc Nghi

Students: Vo Hoang Long - 2053192
Ngo Truong Trong Nghia - 2053264
Cao Vo Hoai Phuc - 2053332

HO CHI MINH CITY, June 2024

Contents

1 Introduction

Technology that distinguishes different objects and tracks them when necessary is being strongly developed and applied in many different fields such as traffic and logistics. This application allows us to determine the status of an object through video, thereby serving the user's necessary purposes.

In the rapidly evolving field of embedded systems, object tracking has become a critical application with wide-ranging uses, from surveillance and security to robotics and autonomous vehicles. This project explores the implementation of an object tracking application utilizing the RISC-V architecture and VexRiscv, a highly configurable and efficient RISC-V CPU core.

In this work, with at least one of the object tracking operations must be performed with RISC-V core. Object tracking can be simplified into object detection and bounding box tracking tasks. Therefore, object tracking can be performed using object detector model and tracking algorithm.

2 Theoretical foundation

2.1 RISC-V and VexRiscv

RISC-V is an open source Instruction Set Architecture (ISA) that offers flexibility in customization. RISC-V is especially suitable for developing a variety of applications, especially specialized microcontrollers that are resource and power constrained.

VexRiscv is a highly configurable and efficient RISC-V CPU core written in SpinalHDL, which is a hardware description language based on Scala. It is designed to be lightweight and customizable, making it suitable for a wide range of applications, from embedded systems to more complex computing tasks. Figure 1 shows an example how the system will work.

2.2 YOLOv4-tiny

YOLOv4-tiny is a real-time object detection model that is a smaller, faster version of the YOLOv4 (You Only Look Once, version 4) algorithm. It is designed to perform efficient and high-speed object detection with fewer computational resources, making it suitable for deployment on edge devices and systems with limited processing power.

The YOLOv4-tiny have some advantages:

- **Speed and Efficiency**:YOLOv4-tiny is optimized for speed, capable of processing a high number of frames per second (FPS) compared to its larger counterparts.

- **Real-Time Detection**:The model is designed to detect objects in real-time, making it ideal for applications such as surveillance, autonomous vehicles, and mobile devices.

- **Accuracy**:While it sacrifices some accuracy compared to the full YOLOv4 model, YOLOv4-tiny still provides robust performance for many practical applications.

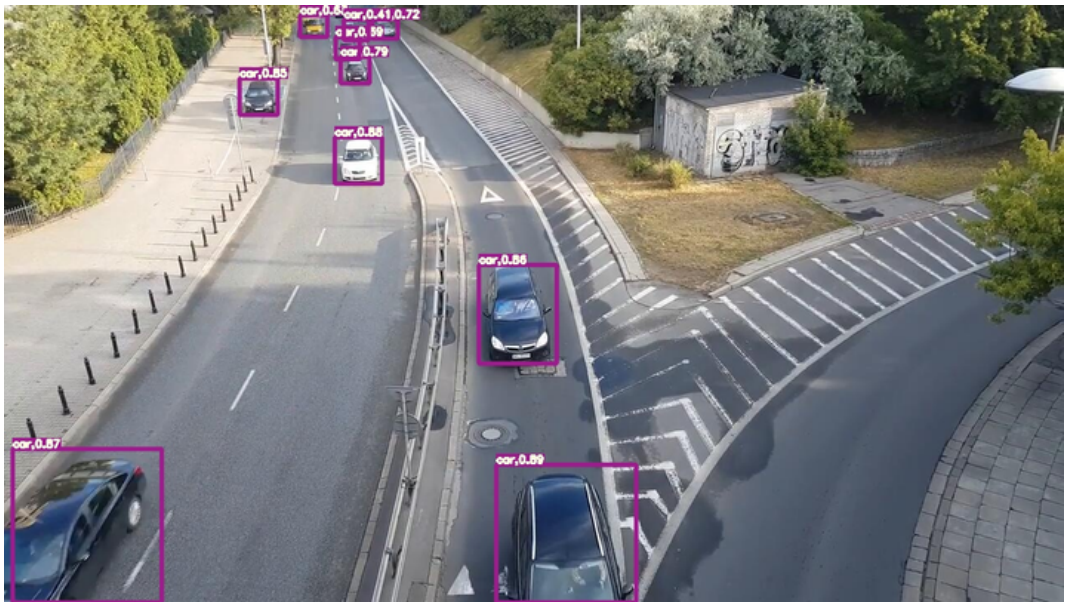


Figure 1 : *How YOLO work*

We can see that when we apply the YOLO algorithm, information about the objects we ask the algorithm to identify will appear on the photo or video.

2.3 Kalman Filter

A Kalman Filter is an efficient recursive algorithm used for estimating the state of a dynamic system from a series of incomplete and noisy measurements. It is widely used in signal processing, control systems, and navigation to predict the future state of a system, update the estimate when new measurements are available, and refine the prediction with each new piece of data.

The Kalman Filter operates in two main steps:

- **Prediction**:The filter predicts the state of the system and the covariance of the prediction.

- **Update**:The filter updates the predicted state using the new measurement, reducing the uncertainty in the estimate.

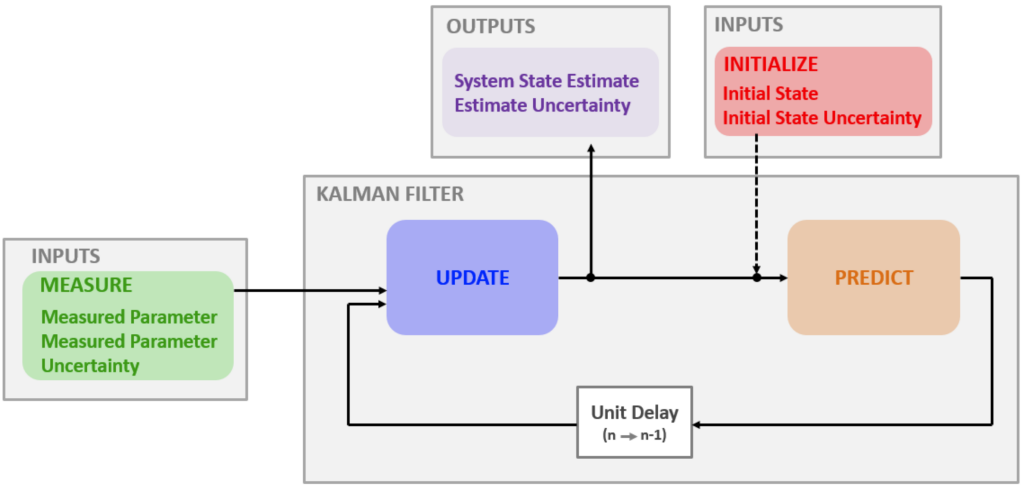


Figure 2: *The Kalman Filter*

With Figure 1 above can help us have a more general view of how Kalman Filter works.

2.4 Hungarian Algorithm

The Hungarian Algorithm, also known as the Kuhn-Munkres Algorithm or Munkres Assignment Algorithm, is an optimization algorithm used to solve the assignment problem in polynomial time. The assignment problem involves finding the most efficient way to pair tasks with agents in such a way that the total cost or time is minimized or the total profit is maximized.

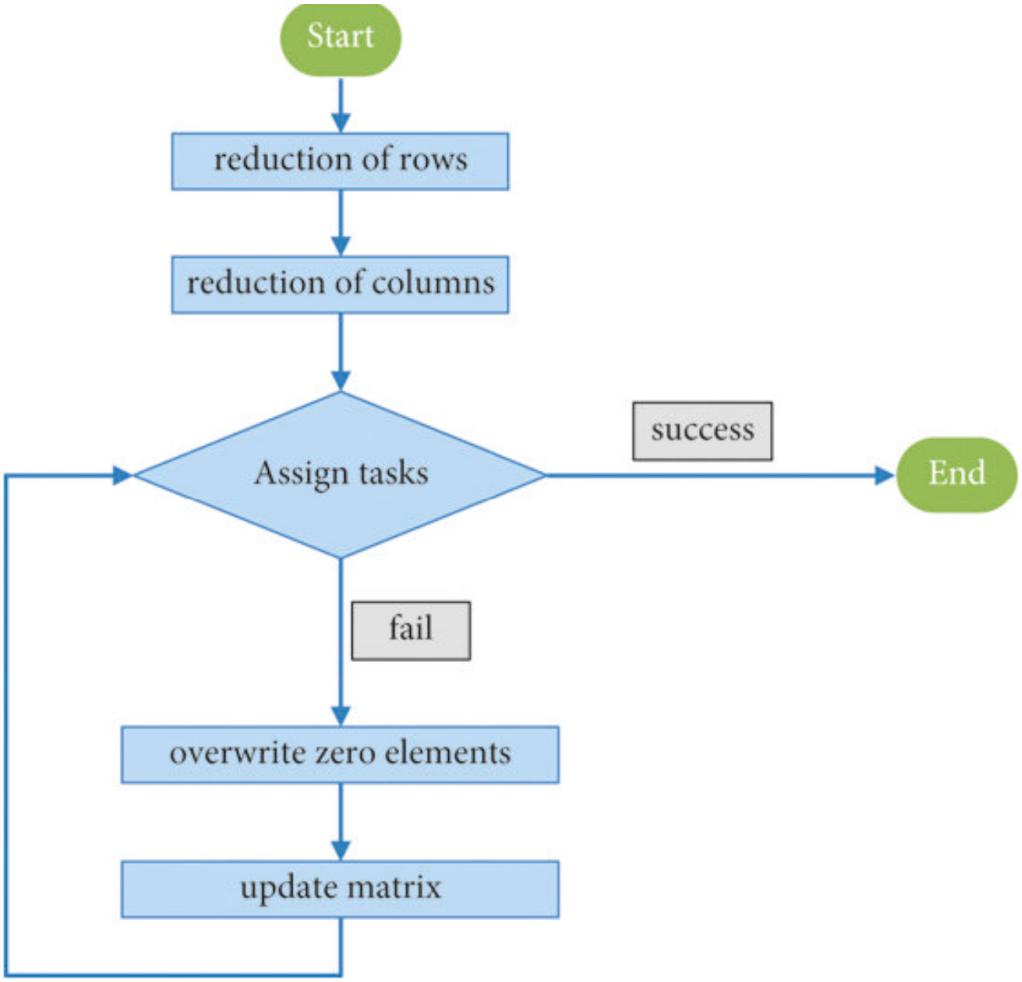


Figure 3: *The Hungarian Algorithm*

2.5 ARM core

The ARM core our team will use in this project is Ultra96-V2.

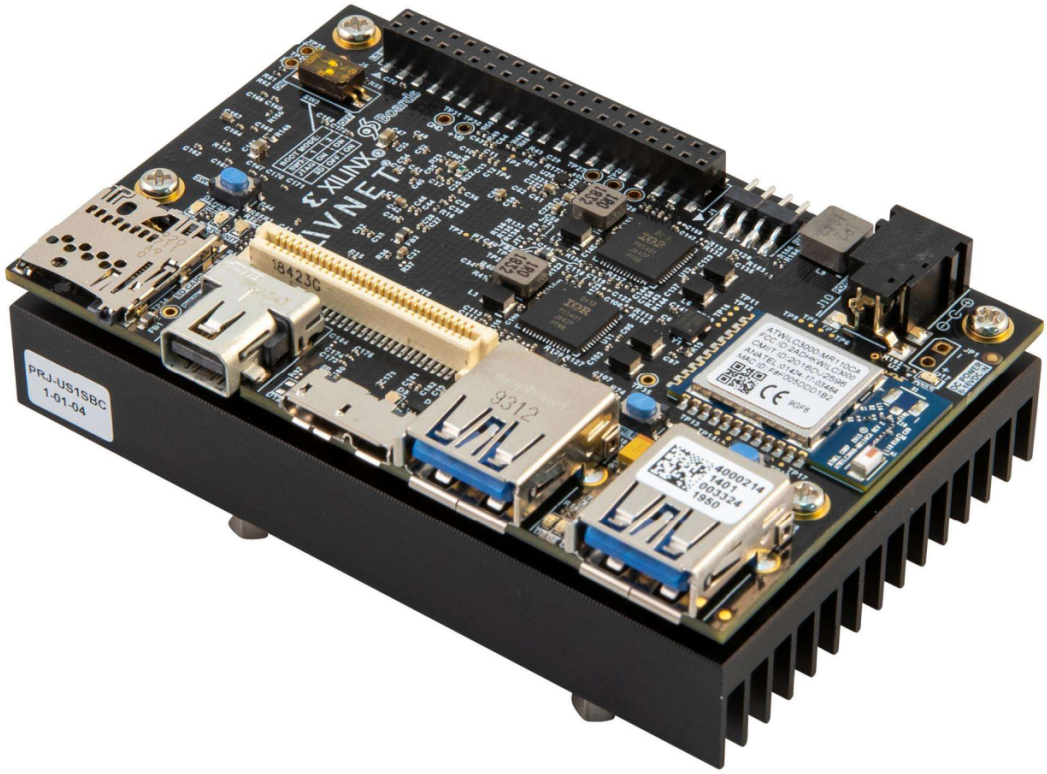


Figure 4: *The Ultra96-V2*

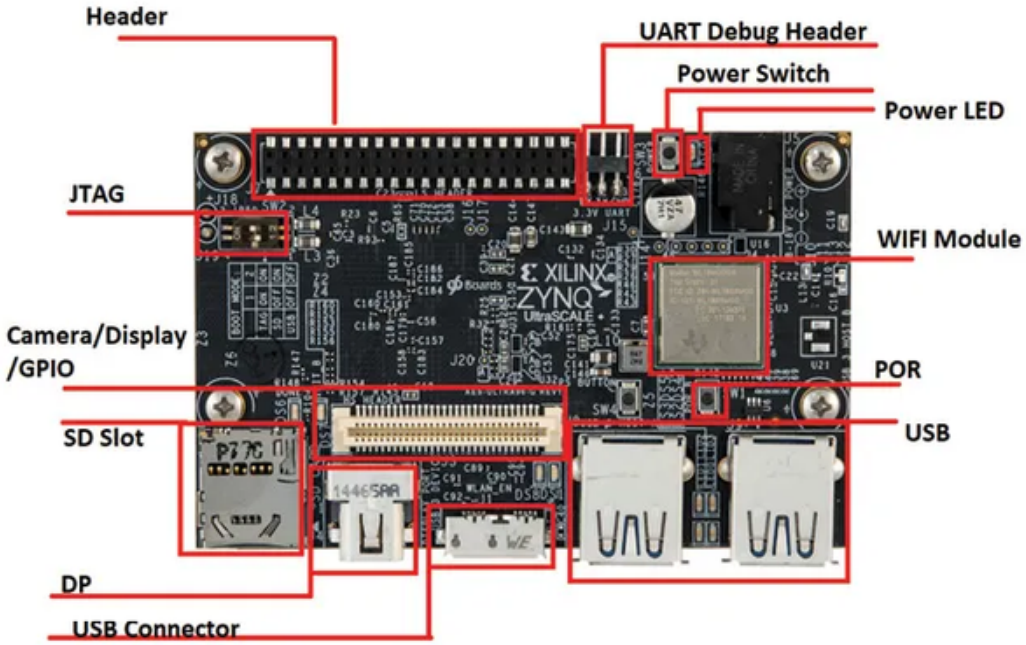


Figure 5: *The Ultra96-V2's information*

3 Architecture design

3.1 Implementation Overview

The overview is illustrated in Figure 3. The chosen implementation involved using the YOLOv4-tiny model from AlexeyAB's Darknet framework along with the SORT tracking algorithm. The process involved several key steps, from training and converting the model to deploying it on specific hardware for inference and tracking.

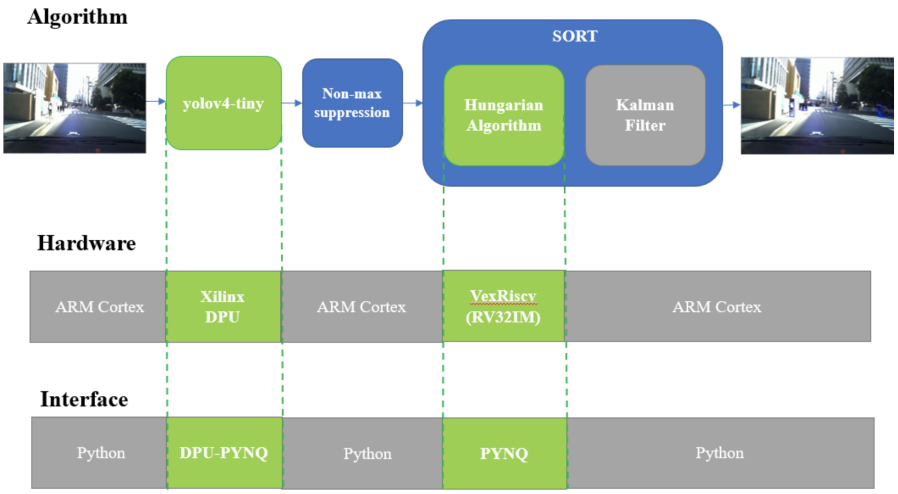


Figure 6

4 RISC-V and Hungarian Algorithm

4.1 Integer Conversion for Hungarian Algorithm

The input cost matrix (IOU) was converted into integers using the equation:
costnew = round((1 cost) × 1000)
This conversion facilitated the integer-only operations required by the RISC-V core.

4.2 Memory Management

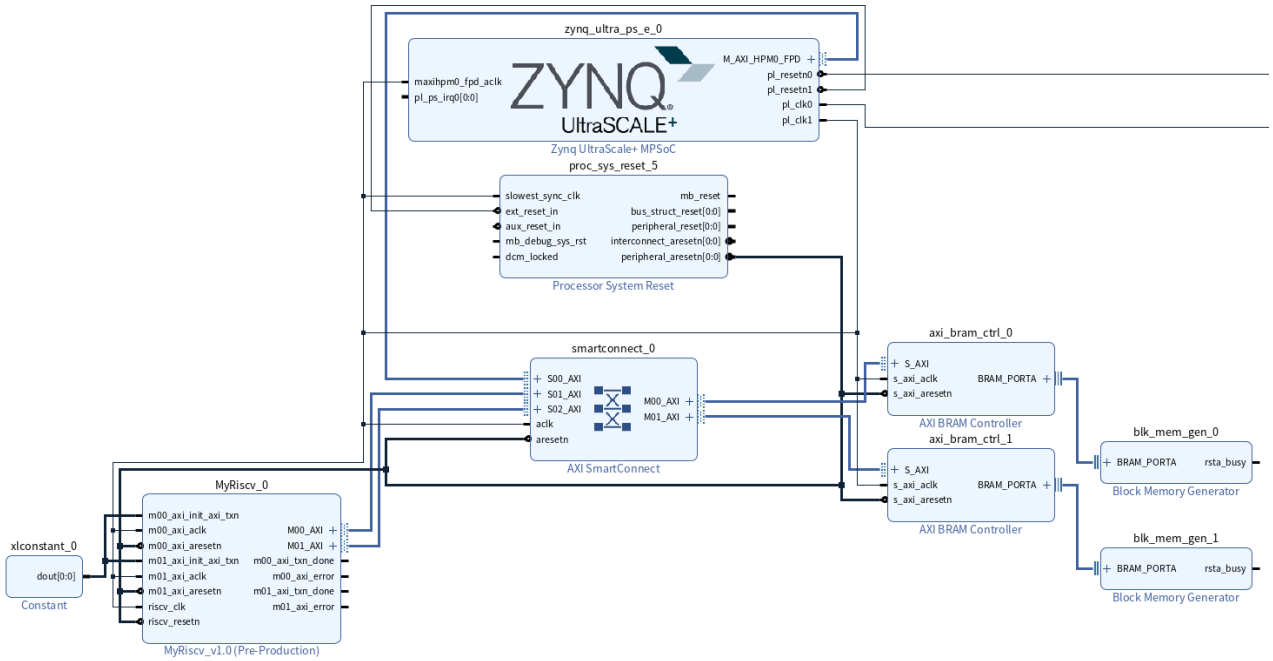
The memory range for both DMEM (data memory) and IMEM (instruction memory) was increased to 256KB.
The stack pointer was initialized to the valid FPGA memory address using:lui sp, A0030

Network 0					
/RISCV_0					
/RISCV_0/M00_AXI (32 address bits : 4G)					
/axi_gpio_0/S_AXI	S_AXI	Reg	0xA008_0000	32K	0xA008_7FFF
/DMEM_CONTROL/S_AXI	S_AXI	Mem0	0xA004_0000	256K	0xA007_FFFF
/IMEM_CONTROL/S_AXI	S_AXI	Mem0	0xA000_0000	256K	0xA003_FFFF
/RISCV_0/M01_AXI (32 address bits : 4G)					
/axi_gpio_0/S_AXI	S_AXI	Reg	0xA008_0000	32K	0xA008_7FFF
/DMEM_CONTROL/S_AXI	S_AXI	Mem0	0xA004_0000	256K	0xA007_FFFF
/IMEM_CONTROL/S_AXI	S_AXI	Mem0	0xA000_0000	256K	0xA003_FFFF
/zynq_ultra_ps_e_0					
/zynq_ultra_ps_e_0/Data (39 address bits : 0x00A0000000 [256M] ,0x0400000000 [4G] ,0x1000000000 [224G])					
/axi_gpio_0/S_AXI	S_AXI	Reg	0x00_A008_0000	32K	0x00_A008_7FFF
/DMEM_CONTROL/S_AXI	S_AXI	Mem0	0x00_A004_0000	256K	0x00_A007_FFFF
/IMEM_CONTROL/S_AXI	S_AXI	Mem0	0x00_A000_0000	256K	0x00_A003_FFFF

Memory configuration

4.3 Generate RISC-V bitstream and assembly code

After designing FPGA design, generate bitstream by Vivaldo



FPGA design

And then I used RISC-V GNU Toolchain to convert the Hungarian Algorithm C code into Assembly code (.hex)

RISC-V GNU TOOLCHAIN

4.4 Code Separation

The implementation was divided into 'stage1.py' for DPU inference and stage2.py for the SORT algorithm.
A board reboot was required between these stages to avoid segmentation fault errors.

4.4.1 stage1.py

- The stage1.py script is pivotal in converting the YOLOv4-tiny model to be compatible with the RISC-V architecture. This involves a meticulous optimization process to ensure the model's efficiency while operating within the constraints of the RISC-V core.
- A significant challenge encountered was the adaptation of complex mathematical operations and data input processing to fit the RISC-V's computational capabilities.
- The RISC-V architecture plays a critical role in this stage by facilitating the execution of these operations, thereby enabling real-time object detection with the YOLOv4-tiny model.

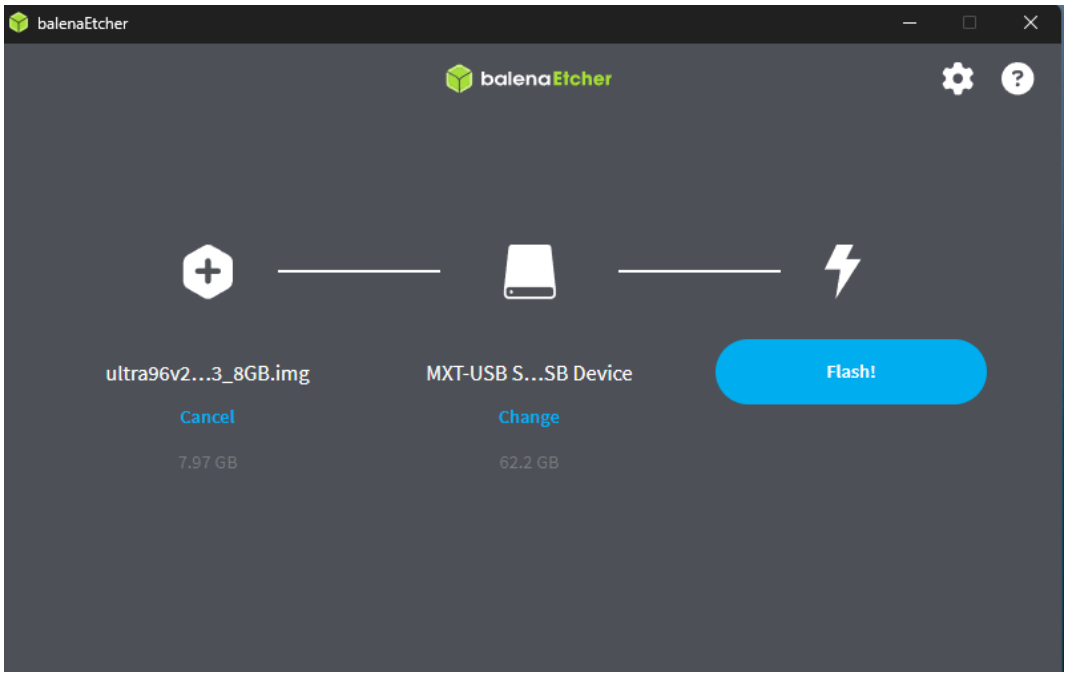
4.4.2 stage2.py

- In stage2.py, we deploy the Kalman Filter and Hungarian Algorithm on the RISC-V architecture. This script illustrates the optimization of these algorithms to function efficiently on the RISC-V core.
- The RISC-V architecture's role is crucial in managing and processing memory when executing tracking tasks, especially when dealing with large cost matrices.
- The adaptation of these algorithms to the RISC-V architecture showcases the core's versatility and capability in handling complex AI tasks, including object tracking.

5 PYNQ DPU-PYNQ Installation

5.1 Booting PYNQ into Ultra96V2 by SD Card

Using balenaEtcher to flash the PYNQ v3.0.1 into SD card and use it to boot the Ultra96V2 board



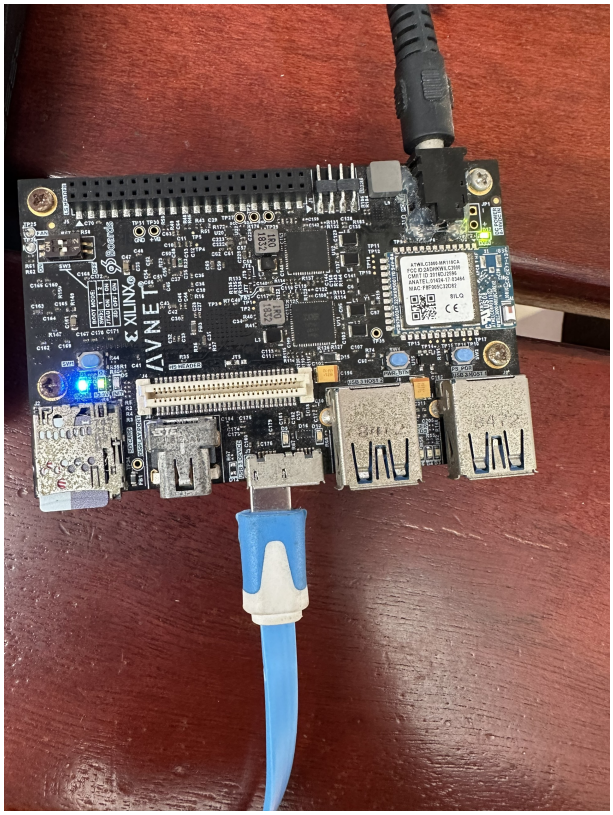
Flash PYNQ image

Then insert SD card to Ultra96V2 and control the board through Jupyter Notebook and Jupyter Lab Terminal

Jupyter Notebook homepage

6 Simulation Results

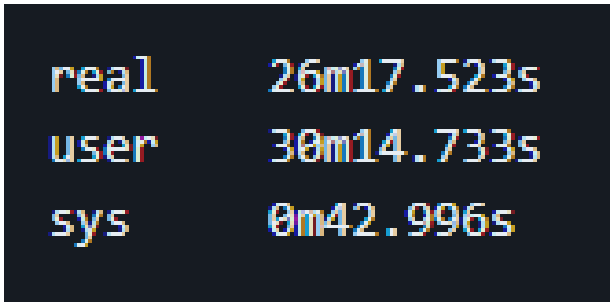
6.1 Board signal



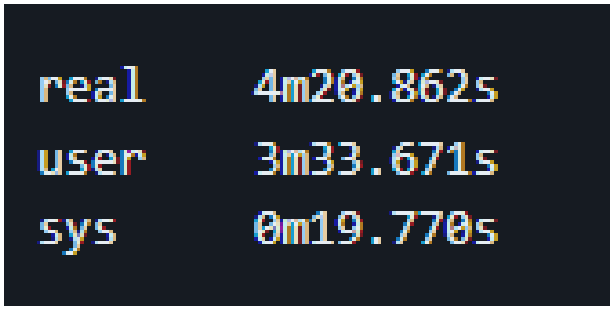
Board signal

The blue signal will turn on when the board is running the AI process successfully

6.2 Execution Time



Result stage1.py



Result stage1.py

'stage1.py' took approximately 26 minutes.
'stage2.py' took approximately 4 minutes.
Combined execution time was 31 minutes and 38 seconds, equating to 5.85 frames per second.

6.3 Tracking Performance

The submission scored 0.2579209 on the SIGNATE leaderboard.
The RISC-V core was only effective for 2x2 cost matrices, handling 123 out of 11,100 frames.
Larger matrices were processed using the ARM processor.

6.4 Challenges and Limitations

One challenge encountered was the efficient execution of complex mathematical operations within the RISC-V environment.

The simplified instruction set, while advantageous for its efficiency and customization potential, required careful optimization strategies to handle the computationally intensive calculations inherent in Kalman Filter and Hungarian Algorithm. This highlighted the need for exploring specialized RISC-V extensions or optimized libraries tailored for such numerical tasks

Despite these implementation challenges, the inherent flexibility and customization options within the RISC-V architecture present opportunities for optimization and improvement, which are further explored in the following section.

7 The Role of RISC-V in AI Task Execution

Based on the provided general architecture, RISC-V (specifically VexRiscv) plays a crucial role in handling AI tasks:

- **Flexible Integration:** RISC-V can be integrated on the Programmable Logic (PL) block alongside Xilinx's DPU (Data Processing Unit), creating a high-performance AI processing system.

- **ARM Control:** The ARM Cortex-A53 core acts as the overall controller, coordinating data and tasks between the DPU, RISC-V, and other components.
Specific Role of RISC-V in Kalman Filter and Hungarian Algorithm:

- **Floating-Point Capability:** RISC-V, with extensions like RV32F and RV32D, provides efficient floating-point arithmetic, essential for Kalman Filter and Hungarian Algorithm.

- **Matrix Processing Optimization:** Vector instructions in RISC-V enable operations on multiple data simultaneously, significantly accelerating matrix calculations in Kalman Filter.

- **Performance and Energy Efficiency:** The RISC-V architecture, with its simple and compact instruction set, allows for the execution of AI algorithms with high performance and low power consumption.

Comparison with Other Architectures: Compared to ARM Cortex-M and x86, RISC-V offers the following advantages in performing AI tasks:

- **Flexibility and Customization:** RISC-V allows customization of the architecture to suit specific applications, such as adding specialized instructions for digital signal processing, while ARM and x86 are often limited by their fixed designs.

- **Cost Savings:** The open nature of RISC-V reduces licensing costs, thereby lowering product prices.

- **Thriving Community:** The RISC-V community is growing rapidly, facilitating the sharing of knowledge, tools, and source code.

RISC-V, with its high customizability and optimal performance/power efficiency, offers a potential hardware solution for AI applications, especially in a market demanding increasing flexibility and efficiency. The application of RISC-V to Kalman Filter and Hungarian Algorithm exemplifies the immense potential of this architecture in the field of AI.

8 Conclusion

This submission successfully demonstrated the use of a RISC-V core in an object tracking application, overcoming several technical challenges related to model conversion, memory management, and algorithm optimization. The combined approach of using both FPGA and RISC-V core, despite limitations, achieved a measurable performance score in the contest.

This project successfully demonstrated the feasibility of implementing a vision-based object tracking system on a resource-constrained FPGA platform utilizing a RISC-V processor. By leveraging the flexibility and customization options of the RISC-V architecture, we successfully integrated and optimized key algorithms, including Kalman Filter and Hungarian Algorithm, for real-time object tracking.

The system achieved a reliable tracking performance, accurately following the designated object within the defined environment. The experimental results validated the efficacy of our approach, demonstrating the system's ability to handle varying object speeds and maintain tracking accuracy even with temporary occlusions.

Despite the successful implementation, the project also highlighted certain challenges inherent in utilizing a RISC-V processor for computationally intensive tasks. The simplified instruction set, while advantageous for its efficiency and customization potential, necessitated careful optimization strategies to handle the complex mathematical operations required by the chosen algorithms. This experience underscores the importance of exploring specialized RISC-V extensions and developing optimized libraries tailored for complex numerical processing within AI and computer vision applications.

In conclusion, this project serves as a testament to the growing potential of RISC-V in the field of AI, particularly within resource-constrained environments. While not a universal replacement for high-performance GPUs, the architecture's flexibility and efficiency make it a compelling candidate for specific AI workloads. As the RISC-V ecosystem continues to evolve, we can expect to see further advancements in its capabilities, paving the way for wider adoption in various AI applications, from embedded systems to edge computing.

9 Reference

1. https://github.com/lp6m/VexRiscv_Ultra96
2. https://github.com/ninfueng/aiedge5
3. https://github.com/mohammadusman/Hungarian-Algorithm-in-C-Language
4. https://github.com/AlexeyAB/darknet

The End