4.2 Design Exploration

4.2.1 Design Decisions

In this section, we elaborate on the critical choices that shape our solution's framework and execution. These decisions, driven by our goal to efficiently utilize the Xilinx Kria K26 board, revolve around optimizing DPU resource sharing, managing memory effectively, and implementing parallel processing to meet our throughput requirements. Each decision is instrumental in ensuring the success and efficiency of our project.

1. DPU Resource Sharing

- Decision: To implement two methods for DPU resource sharing: a software only approach and an approach using the Zynq UltraScale+'s onboard hardware mutex for enhanced control.
- Software Approach: We will designed a function, use_DPU(), to manage DPU calls via execute_async(), incorporating a queuing system with priority scheduling. This ensures organized access to the DPU, prioritizing tasks based on their urgency and importance, which may vary as we analyze performance outcomes.
- Hardware Mutex Integration: In conjunction with our software strategy, leveraging the Zynq UltraScale+'s hardware mutex provides a robust mechanism for mutual exclusion at the hardware level, offering a failsafe against potential software bottlenecks.
- Importance: Efficient DPU resource sharing is crucial for maintaining high performance and responsiveness of our system, particularly when running multiple algorithms that depend on this shared resource.

2. Memory Management

- Decision: Adopting a memory affinity approach to divide the available 5GB of DDR memory into distinct banks for dedicated process usage.
- Importance: This strategy ensures that each algorithm has access to its required memory resources without interference, reducing contention and latency. It's a pivotal decision for optimizing memory utilization and performance, given the physical memory constraints of our FPGA board.

3. Parallel Processing Implementation

- Decision: To run the blink detection and eye tracking algorithms on separate threads, with eye tracking further split across two threads to handle different frames simultaneously.
- Importance: This approach is aimed at maximizing throughput and minimizing latency. By employing parallel processing, we can address the inherently slow nature of the eye tracking algorithm and ensure that the system meets our throughput requirement of <5ms. Parallelism allows us to leverage the full computational potential of the FPGA board, ensuring that each component operates efficiently and contributes to the overall speed and reliability of the system.

4.2.2 Ideation

1. Change Hardware

- One of the idea on changing hardware could be increasing RAM on the board. By having more RAM allows each thread to have a larger memory allocation without contention for resources. This can prevent slowdowns caused by memory thrashing and improve overall system responsiveness.
- Instead of using AMD Kria board, we could switch to NVIDIA Jetson AGX Xavier. It features an integrated NVIDIA Volta GPU with 512 CUDA cores, along with an eight-core ARMv8.2 CPU complex, making it well-suited for both compute-intensive and graphics-intensive tasks. NVIDIA GPUs are equipped with CUDA cores and Tensor Cores, which are highly parallelized processing units optimized for different types of workloads. Depending on your application's requirements, these specialized cores can accelerate computations and speed up processing tasks.

2. Parallelism

- Parallelism is one way for us to process multiple tasks together. By executing multiple tasks concurrently, parallelism allows for overlapping computation, communication, and I/O operations. This can lead to improved overall throughput and reduced latency, as tasks can progress simultaneously rather than waiting for one another to complete.
- Due to the architecture of Kria board of having four separated DDR4 memory, data can be processed independently across different subsets or partitions of the dataset. Parallelism enables data parallelism by distributing these data partitions across multiple processing units, allowing for simultaneous processing of different parts of the dataset and reducing overall processing time.

3. Memory Management

- In a multi-threaded environment, concurrent access to shared resources without proper synchronization can lead to data corruption or inconsistent states. By using locking mechanisms such as mutexes (mutual exclusion locks), semaphores, or read-write locks, developers can ensure that only one thread or process accesses a shared resource at a time, preventing data corruption and maintaining data consistency.

- Memory compression algorithms compress data stored in RAM, allowing more data to fit within the available memory capacity. This can reduce the frequency of paging (swapping memory contents between RAM and disk), which is a costly operation in terms of performance. By keeping more data in RAM and reducing the need for frequent disk accesses, memory compression can lead to faster overall performance.

4. DPU Acceleration

- DPUs are often used to accelerate machine learning inference tasks, such as image classification, object detection, and natural language processing. By leveraging dedicated hardware resources optimized for matrix operations and neural network inference, DPUs can significantly speed up the process of executing machine learning models, leading to faster inference times and improved overall performance.
- Instead of using DPU to infer machine learning algorithms, ASICs could be used in this application. ASICs are custom-designed hardware accelerators optimized for specific tasks or algorithms. ASICs can offer superior performance and energy efficiency compared to general-purpose CPUs or GPUs for specialized tasks like blink detection and eye tracking. Designing and manufacturing ASICs can be expensive and time-consuming, but they can provide significant performance benefits for specific workloads.

5. Different Image Processing Techniques

- Instead of processing the entire image, focus on extracting and processing only relevant features or regions of interest. Techniques such as region-based processing, saliency detection, and object detection can help narrow down the processing scope, leading to faster execution times.
- Downsampling involves reducing the resolution of an image, typically by averaging or subsampling pixels. This technique can significantly reduce the computational complexity of subsequent image processing tasks while maintaining essential information.

4.2.3 Decision-Making and Trade-Off

DPU Resource Sharing Options

Options:

Software-only approach: Utilize a software-based queue and priority scheduling system.

Combined Software and Hardware Mutex approach: Integrate the software solution with the hardware mutex provided by Zynq UltraScale+ for added efficiency and reliability.

Analysis:

The software-only approach offers simplicity and flexibility in implementation but might face scalability and efficiency issues under high demand.

The combined approach adds complexity but promises improved reliability and control, ensuring that DPU access conflicts are minimized.

Decision:

We opted for the combined approach, as the added control and reliability from the hardware mutex significantly outweigh the increased complexity. This choice was driven by our priority for system robustness and performance under varying loads.

Memory Management Options

Options:

Unified Memory Pool: A single, shared memory pool accessible by all processes.

Memory Affinity: Dividing the available memory into dedicated banks for specific processes.

Analysis:

A unified memory pool simplifies memory management but can lead to contention and inefficient use of resources.

Implementing memory affinity introduces complexity in management but ensures dedicated resources, reducing contention and potentially increasing performance.

Decision:

The memory affinity strategy was selected for its direct benefits in reducing latency and increasing the predictability of memory access times, crucial for meeting our throughput targets.

Parallel Processing Strategies

Options:

Sequential Processing: Running algorithms one after another, focusing on simplicity.

Concurrent Execution with Limited Parallelism: Introducing basic parallelism while maintaining some sequential operations.

Full Parallel Processing: Maximizing parallel execution across all algorithms and processes.

Analysis:

Sequential processing simplifies development but does not utilize the full capabilities of our hardware, leading to potential bottlenecks.

Limited parallelism offers a balance but may still underutilize available resources.

Full parallel processing maximizes hardware utilization but requires sophisticated control mechanisms to manage resource sharing effectively.

Decision:

We embraced full parallel processing, accepting the challenge of complexity for the sake of maximizing throughput and efficiency. This strategy aligns with our goal to leverage the FPGA board's capabilities fully, ensuring that each component contributes optimally to the system's overall performance.

Decision-Making Tools

To facilitate our decision-making, we employed a weighted decision matrix, assigning values to key criteria such as performance efficiency, complexity, scalability, and reliability. This quantitative analysis supported our qualitative assessments, guiding us toward choices that best align with our project goals and constraints.

4.3 PROPOSED DESIGN

4.3.1 Overview

Our project involves creating a system on the Kria K26 board that can understand where someone is looking and infer their emotions from their eye movements. Imagine it as a smart camera that doesn't just see you but tries to understand how you're feeling by paying close attention to your eyes.

Key Components of Our Design

DPU: This part of the FPGA is like an accelerator for machine learning applications.

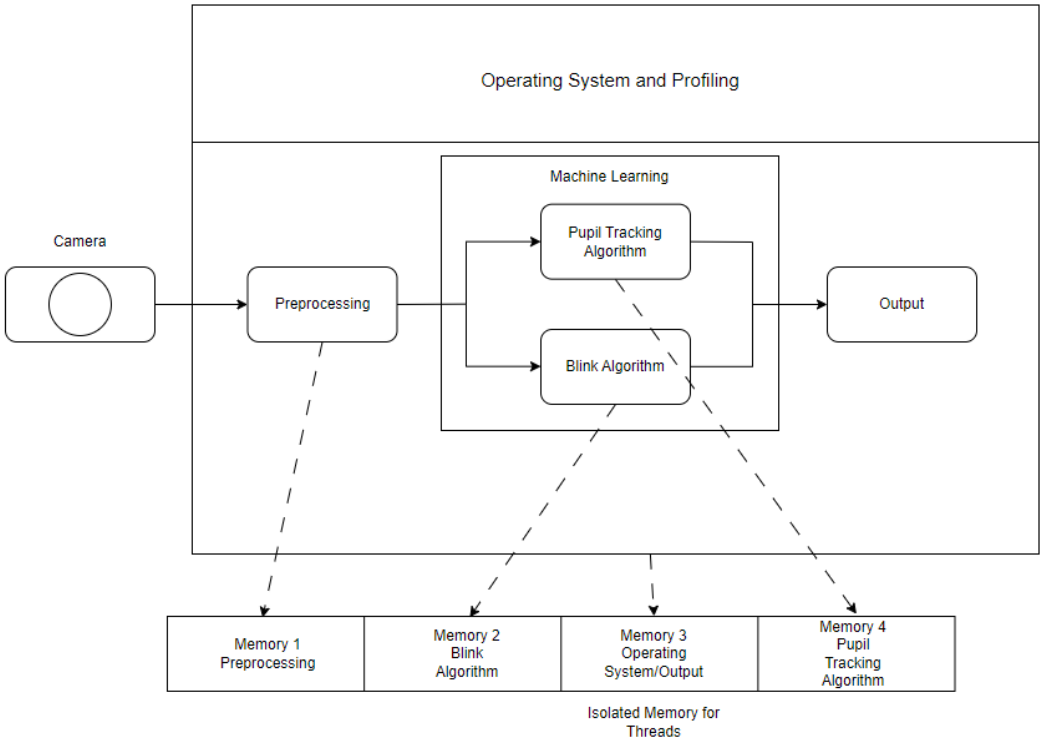
Memory Management: Because memory is limited, we've set up a system where the memory is divided into sections, each dedicated to a specific task. This way, everything runs smoothly without any hold-ups, ensuring that the system can keep up with real-time analysis without any lag.

Parallel Processing: To make the system faster, we can run multiple algorithms at the same time, i.e. multitasking. It can run several tasks at the same time (like watching for blinks and tracking eye movements) without getting mixed up. This is key to making the system fast and responsive.

How It All Comes Together

All these parts work together like a well-coordinated orchestra. The DPU is the main processing unit, and the DPU scheduling unit ensures that all algorithms share the resource as needed. The memory management unit also ensures that all algorithm has sufficient memory to run. Finally, parallel processing allows the system to process multiple operation at the same time, achieving a higher throughput.

The result? A system that can quickly and accurately understand where you're looking and how you're feeling, just by watching your eyes. This technology could have many uses, from helping doctors understand their patients better to making computers and gadgets more responsive to our needs and emotions.



4.3.2 Detailed Design and Visual(s)

High-Level Overview

Our design is centered around a highly efficient FPGA-based eye-tracking system, specifically utilizing the Kria K26 FPGA board. Below is a detailed overview of our system:

- FPGA Board (Xilinx Kria K26): Serves as the core platform, integrating all components and algorithms.
- Deep Learning Processing Unit (DPU): A specialized processing IP within the FPGA for executing deep learning algorithms.
- Memory Banks: The FPGA's memory is strategically divided into separate banks, each allocated to different processes (e.g., image preprocessing, blink detection, eye tracking) to optimize memory usage and system performance.
- Image Preprocessing: This subsystem prepares raw eye images for analysis, improving the accuracy of subsequent detection and tracking algorithms.
- Blink Detection and Eye Tracking Algorithms: Core algorithms that analyze preprocessed images to detect blinks and track eye movements, respectively. These algorithms are critical for determining the user's gaze direction and emotional state.
- DPU Resource Sharing Mechanism: Incorporates a software-based queue with priority scheduling and hardware mutex to manage access to the DPU among different algorithms.
- Parallel Processing Threads: To enhance system throughput and responsiveness, blink detection and eye tracking algorithms are executed on separate threads, with eye tracking further split to handle different frames simultaneously.

Sub-System and Component Descriptions

Image Preprocessing

- Operation: Enhances image quality for accurate analysis, involving noise reduction, contrast adjustment, and scaling. This step is crucial for the reliable performance of downstream algorithms.
- Technical Requirement: Must process images within milliseconds to ensure real-time performance, integrating seamlessly with the eye tracking and blink detection algorithms.

Blink Detection and Eye Tracking

- Operation: Utilize machine learning models to interpret preprocessed images, identifying blinks and tracking eye movements. Blink detection prioritizes rapid identification of eye closures, while eye tracking focuses on determining gaze direction.
- Integration: Both algorithms request DPU access via the resource-sharing mechanism, with scheduling managed to balance urgency and computational load.

DPU Resource Sharing Mechanism

- Software Queue with Priority Scheduling: Algorithms queue for DPU access, with priority given based on predefined criteria, ensuring critical tasks receive timely processing.
- Hardware Mutex: Enhances control over DPU access, preventing conflicts and ensuring smooth operation across concurrent tasks.

Memory Management

- Affinity Approach: Allocates separate memory banks for each major process, reducing contention and speeding up access times, which is critical for maintaining high throughput and system responsiveness.

Parallel Processing Threads

- Implementation: Designed to execute multiple algorithms in parallel, significantly reducing processing time and increasing system throughput. Special consideration is given to the eye tracking algorithm, which is split across threads to handle different frames, addressing its computationally intensive nature.

This technical description, accompanied by the detailed block diagram, provides a comprehensive understanding of our design. It outlines the critical components, their functions, and how they are integrated to create a high-performance eye-tracking system. This information should enable another senior design team to grasp the intricacies of our solution and consider its implementation.

4.3.3 Functionality

In a Healthcare Setting

User Action: A patient sits in front of a diagnostic monitor equipped with our eye-tracking system during a mental health assessment.

System Response: As the patient views various stimuli on the screen, the system analyzes their gaze and blinking patterns. It provides real-time feedback to healthcare professionals about the patient's emotional state and engagement, assisting in diagnosing or tailoring treatment plans.

In an Educational Environment

User Action: A student interacts with an educational software application on a tablet that incorporates our eye-tracking technology.

System Response: The system monitors the student's eye movements to gauge where their attention is focused and how they react to different educational content. This information helps the software adapt in real-time, offering a personalized learning experience by emphasizing topics that captivate the student's interest or revisiting areas where their attention wanes.

In Personal Computing

User Action: A user navigates a website using a computer equipped with our eye-tracking system.

System Response: The system detects the user's gaze direction, allowing for hands-free navigation based on where the user is looking. It could also adjust content display based on the user's emotional response to different elements, enhancing the browsing experience.

4.3.4 Areas of Concern and Development

1. High Throughput and Low Latency: The system's capability to process data at a high rate, with a target throughput of <5ms, ensures real-time responsiveness. This is crucial for applications requiring immediate feedback based on eye movement analysis, such as adaptive learning software or patient emotional state monitoring in healthcare settings.

2. Efficient Resource Utilization: Through intelligent design choices like DPU resource sharing, memory affinity, and parallel processing, our system optimizes the limited resources of the FPGA board. This ensures that the device operates smoothly, even under the demand of running multiple complex algorithms simultaneously.

3. Scalability and Flexibility: The modular approach to algorithm implementation and resource management allows for flexibility in system deployment across various contexts, from healthcare to personal computing. This adaptability ensures that our design can evolve to meet emerging user needs and technological advancements.

Primary Concerns for Delivering a Product/System

1. Algorithm Efficiency and Accuracy: The effectiveness of the eye-tracking and blink algorithms heavily depend on the accuracy and efficiency of the underlying software system. Ensuring these algorithms can perform under real-world conditions without significant errors or delays is crucial.

2. Hardware Limitations: While we've designed our system to optimize resource utilization on the Kria K26 board, physical constraints such as memory capacity and DPU availability remain a challenge.

4.4 Technology Considerations

We are using the AMD Kria KR260, a development platform for Kria K26 SOMs, in our design. The Kria K26 System-on-Module (SoM) by AMD offers a compact and integrated solution for embedded applications, providing a balance of performance, power efficiency, and versatility.

Strengths

- The KR260 is equipped with high-performance interfaces tailored for robotics and industrial applications. These interfaces likely include GPIO, UART, I2C, SPI, and CAN ports, facilitating seamless integration with various sensors, actuators, and control systems.

- The KR260's support for Kria K26 SOMs offers versatility, allowing developers to choose the appropriate SOM configuration based on their application requirements. This scalability ensures that the platform can address a wide range of robotics and industrial use cases.

- Leveraging the capabilities of the Kria K26 SOMs, the KR260 platform offers scalability in terms of processing power, memory, and connectivity options. This scalability enables developers to scale their robotic systems to meet evolving performance demands.

Weaknesses

- The advanced features and capabilities of the KR260 platform comes with a higher upfront cost compared to simpler development platforms. This could be a limiting factor for developers with budget constraints or hobbyists exploring robotics projects.

- Developing applications for robotics and industrial applications can be complex, requiring expertise in hardware, software, and system integration. While the KR260 platform aims to simplify development with native ROS 2 support and high-performance interfaces, there may still be a learning curve for developers new to robotics.

Trade-offs

Performance vs. Power Consumption

The high-performance interfaces and processing capabilities of the KR260 platform may result in higher power consumption, especially in battery-powered robotics applications. Developers must balance performance requirements with power efficiency to ensure optimal system operation and longevity.

Versatility vs. Specialization

While the KR260 platform offers versatility in supporting various robotics and industrial applications, it may be optimized for specific use cases within these domains. Developers should assess whether the platform's features align with their project requirements or if a more specialized development platform would be more suitable.

Solutions

1. NVIDIA Jetson Series

- NVIDIA Jetson series of embedded platforms, such as Jetson Nano, Jetson Xavier NX, and Jetson AGX Xavier, offer high-performance GPU acceleration suitable for running multiple machine learning algorithms concurrently.
- These platforms feature CUDA support, allowing developers to leverage GPU parallelism for accelerated inference tasks.
- The Jetson series also provides support for popular machine learning frameworks like TensorFlow, PyTorch, and ONNX, facilitating easy deployment of machine learning models.

2. Google Coral Dev Board

- The Google Coral Dev Board is another option for accelerating machine learning inference tasks at the edge.
- It features Google's Edge TPU (Tensor Processing Unit) for high-performance AI acceleration with low power consumption.
- The Coral Dev Board supports TensorFlow Lite and TensorFlow Lite Micro, making it suitable for running multiple machine learning algorithms concurrently in resource-constrained environments.

Design Alternatives

1. Distributed Computing Architecture

- Instead of relying on a single powerful device, distribute the computational load across multiple edge devices interconnected in a network. Each edge device can be responsible for processing a subset of the input data or running specific machine learning algorithms. Use communication protocols such as MQTT or gRPC for inter-device communication and coordination. This approach allows for scalability and fault tolerance, as well as efficient utilization of resources across the network.

2. Custom ASIC-based Solution

- Design custom Application-Specific Integrated Circuits (ASICs) tailored to the specific requirements of your machine learning algorithms. Develop dedicated hardware accelerators optimized for parallel execution of inference tasks. Leverage the high performance and power efficiency of ASICs to achieve the desired throughput and latency targets. This approach requires significant upfront investment in ASIC design and fabrication but offers the potential for unparalleled performance and energy efficiency.

4.5 Design Analysis

Our team has not implemented anything on the board. We only tested the eye tracking algorithm from previous team on the board. Our team has built the environment on the board and set up a workstation that could communicate with the board. As for now, the image semantic segmentation has been tested on the PC. In the upcoming days, we will implement blink algorithm onto the board.

Here's a breakdown of the situation and potential plans for future design and implementation work:

1. Validation on Target Hardware: The first step would be to validate the proposed design on the Xilinx Kria K26 FPGA board. This involves porting the implemented algorithms and components onto the board and testing their performance in a real-world environment.

2. Identifying Challenges: Once the design is tested on the board, it's crucial to identify any challenges or discrepancies between the expected and observed performance. This could include issues related to resource constraints, hardware limitations, or unexpected behavior.

3. Iterative Optimization: Following validation and identifying challenges, an iterative optimization process would be necessary to address any issues and fine-tune the implementation. This may involve optimizing algorithms for hardware acceleration, adjusting resource allocation strategies, or refining parallel processing techniques.

4. Testing and Validation: Rigorous testing and validation on the target hardware platform are essential to ensure that the system meets performance requirements and functions reliably under various conditions.

5. Documentation and Reporting: Comprehensive documentation of the implementation process, including challenges faced and solutions developed, should be maintained. This documentation will serve as a valuable resource for future iterations of the project and knowledge transfer.

6. Feasibility Assessment: Throughout the implementation and optimization process, ongoing feasibility assessments should be conducted to evaluate whether the design goals are achievable within the constraints of the hardware platform.

7. Addressing Build Issues: If any build issues are encountered during the implementation process, they should be addressed promptly through troubleshooting, debugging, and potentially revising the design or implementation approach.