Optimized Multi-Processor System-on-Chip (MPSoC) Design for Low-Resource JPEG Encoding

Kanishka Gunawardana, Chandula Adhikari, Isuru Nawinne
Department of Computer Engineering Faculty of Engineering, University of Peradeniya, Sri Lanka
e19129@eng.pdn.ac.lk, e19008@eng.pdn.ac.lk, isurunawinne@eng.pdn.ac.lk

***Abstract*—This paper presents the design and optimization of a Multiprocessor System-on-Chip (MPSoC) architecture for low-resource JPEG encoding. The initial design employs six Central Processing Units (CPUs) to implement a pipelined JPEG encoder, divided into stages: color space conversion, level shifting, discrete cosine transform (DCT), quantization, Huffman encoding, and output buffering. The pipeline is then optimized using custom hardware components, including First-In-First-Out (FIFO) queues, custom instruction blocks, and superscalar pipelines to improve performance and efficiency. The paper details the design process, various optimization strategies, implementation challenges, and the resulting performance improvements. The system is implemented and evaluated using the Cyclone IV E Field Programmable Gate Array (FPGA) platform with Nios II/e processors, and the design was developed using the Quartus II software, demonstrating significant gains in throughput and resource efficiency.**

***Index Terms*—Pipelined MPSoC, JPEG Encoding, FPGA, Optimization.**

I. Introduction

JPEG encoding is essential for compressing digital images, reducing storage requirements and transmission bandwidth without significantly compromising image quality. The encoding process involves several computationally intensive stages, making hardware acceleration critical for achieving real-time performance in applications such as digital cameras, mobile devices, and multimedia systems.

This research focuses on implementing a pipelined JPEG encoder using a Multiprocessor System-on-Chip (MPSoC) architecture, optimized for low resource usage and higher throughput. The JPEG encoding process is divided into six stages, as illustrated in Fig. 1(a): color space conversion, level shifting, discrete cosine transform (DCT), quantization, Huffman encoding, and output buffering.

Field Programmable Gate Arrays (FPGAs) provide a flexible and reconfigurable platform for implementing such computationally demanding tasks. This study utilizes the Altera DE2-115 development board, equipped with a Cyclone IV FPGA, to implement a pipelined JPEG encoder based on an MPSoC architecture. Initially, each stage is managed by a separate Nios II/e soft processor core, integrated using the Quartus II software, with hardware FIFOs facilitating communication between stages, as shown in Fig. 1(b).

The primary goal is to optimize the MPSoC architecture for JPEG encoding by addressing performance bottlenecks, enhancing memory management, and integrating custom hardware components. Specific optimizations include strategic allocation of on-chip memory, custom FIFO queues to replace certain processing stages, and the implementation of superscalar pipelines for parallel data processing. This paper details the design and implementation of an efficient low-resource MPSoC JPEG encoder, highlighting the challenges and solutions developed. Performance evaluations demonstrate significant improvements in throughput and processing time, illustrating the potential of FPGA-based MPSoC architectures for efficient JPEG encoding.

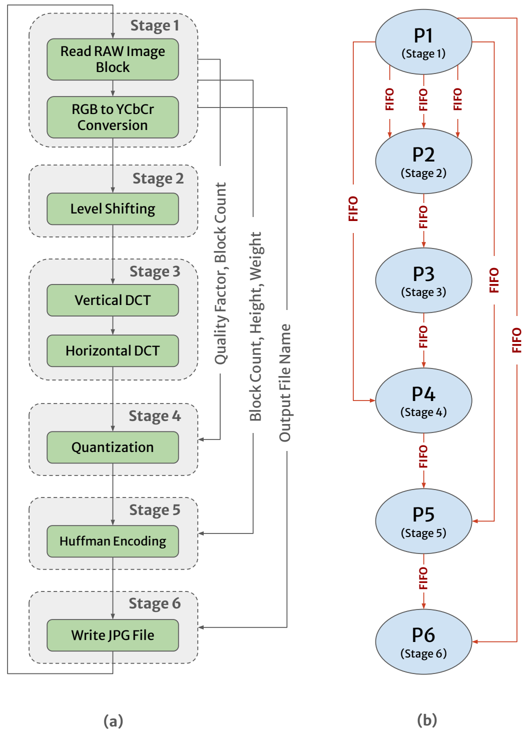

Fig. 1. (a) JPEG Encoding Process Stages Before Optimization. (b) Six processors are configured in a pipeline with one processor for each stage

The paper is organized as follows: Section 2 reviews related work, Section 3 describes the system design, Section 4 covers optimization techniques, Section 5 presents the experimental setup, Section 6 analyzes the results, and Section 7 concludes.

II. Past Related Works

Research in MPSoC architectures for JPEG encoding has provided a wealth of insights, guiding our work toward an optimized, resource-efficient pipeline design.

Shee et al. [1] explored heterogeneous multiprocessor configurations for JPEG encoding in 2006, employing two parallelization models: a master-slave model and a pipeline model. Using Tensilica's Xtensa LX processors with queues, they achieved up to 4.6X speed-up on a seven-core system, while only increasing the area by 3.1X. With processor utilization rates between 50-80%, their work demonstrated the potential for JPEG parallelization but highlighted a trade-off between performance gains and hardware area expansion. Our design takes this further by addressing hardware efficiency in resource-constrained environments through custom components and fine-grained parallelism.

Building on this, Javaid and Parameswaran [2] introduced an Integer Linear Programming (ILP) based methodology for synthesizing heterogeneous pipelined multiprocessor systems, tailored for JPEG encoding. Their ILP model allowed rapid exploration of over 4.2×10^{13} system configurations, using pruning algorithms to identify pseudo Pareto-optimal designs in less than 100 seconds. This inspired our approach to systematic optimization, although we focus more on custom instruction sets to achieve greater resource efficiency.

Wu et al. [3] contributed to automated design space exploration with ILP, incorporating profiling to locate bottlenecks in data processing and communication. This profiling tool provided valuable insights into which stages of JPEG encoding require acceleration, showing up to 4.72X speed-up with task-level parallelism. However, predefined architectures limited flexibility in customizing stages. This informed our decision to integrate superscalar pipelines and stage-specific hardware components for greater adaptability, thereby optimizing resource use and enhancing efficiency in constrained environments.

Lee et al. [4] advanced MPSoC performance through a scalable NoC-based approach that achieved a 373.6% speed-up, leveraging message-passing on RISC processors within a 2x2 grid. Their design demonstrated the scalability potential of NoC platforms for distributed tasks. We gained insight into the benefits of NoC-based models but saw the need for stage-specific optimizations in our pipeline design, as general-purpose RISC processors proved insufficient for resource-constrained tasks. Consequently, we focused on custom instructions and hardware to optimize the DCT and quantization stages.

Wang et al. [5] and Grubišić and Zadrija [7] highlighted the power of hardware accelerators for computationally intensive JPEG stages like DCT-2D and quantization. Their focus on heterogeneous configurations yielded substantial speed-ups, with Wang et al. achieving a 26.01x increase. Grubišić and Zadrija's replication of pipeline stages showed promise in reducing bottlenecks, though their pipelines were relatively simplistic. These studies underscored the value of task-specific acceleration, influencing our decision to incorporate a flexible six-stage pipeline with custom components for critical JPEG encoding functions, as well as advanced memory management for smoother task transitions.

Kunz et al. [6] provided insights into hardware transactional memory (HTM), showcasing its efficiency over traditional synchronization methods with performance gains of up to 30% and 32% energy savings. Their model, while innovative, was tested on low-synchronization tasks using Java processors. This pointed us toward the applicability of HTM in MPSoC systems but highlighted the need for its implementation in synchronization-heavy environments.

Salamy et al.[8] and Kluter et al.[9] concentrated on memory management and communication efficiency. Salamy's use of scratchpad memory (SPM) emphasized predictable memory allocation for real-time applications, enhancing system throughput. Similarly, Kluter's Architecturally Visible Communication (AVC) buffers achieved a 4.2X speed-up in communication-bound tasks. These approaches underscored the importance of memory predictability and communication efficiency, shaping our pipeline design with custom memory management to maintain high throughput and low overhead during data-intensive stages of JPEG encoding.

Our work extends these previous studies by combining the strengths of heterogeneous multiprocessing, custom hardware components, and advanced pipeline structures. We aim to achieve significant performance improvements while maintaining a focus on resource efficiency, particularly suitable for low-resource environments.

III. System Design

The system design of the JPEG encoder involve a pipelined architecture that utilizes multiple Nios II processors, each dedicated to a specific stage of the JPEG encoding process. This section provides a detailed description of the architecture and hardware components used to enhance performance.

A. *JPEG Encoding Stages*

1) **Color Space Conversion**: Converts raw image data from RGB to YCbCr format.
2) **Level Shifting**: Adjusts DC coefficients by subtracting 128 from each component value.
3) **Discrete Cosine Transform (DCT)**: Transforms the spatial domain data into the frequency domain.
4) **Quantization**: Reduces the precision of the DCT coefficients using quantization tables.
5) **Huffman Encoding**: Compresses the quantized DCT coefficients using Huffman coding.
6) **Output Buffering**: Writes the compressed data to the output JPEG file.

B. *MPSoC Design*

The multiprocessor system-on-chip (MPSoC) architecture is implemented on the FPGA using the following components:

1) **Nios II/e Processors**: The Nios II/e (economy) processor was selected because it requires the least amount of FPGA resources, making it well-suited for low-resource implementations. Although it has a limited set of user-configurable features, its free availability in Quartus II software makes it a practical choice for this design.

2) **Hardware FIFO Queues**: Hardware FIFO queues facilitate data transfer, with each neighbouring processor acting as both a producer and a consumer, ensuring smooth communication between pipeline stages.
3) **Memory Modules**: On-chip memory is strategically allocated to meet the memory requirements of each processor as mentioned in the Memory Management Strategy section.
4) **Resource Allocation**:
   - **Memory Sizes**: 128 KB for the first processor and 32 KB for each of the remaining five processors.
   - **FIFO Depth**: Set to 128 to accommodate the 8x8 block.
   - **Clock Speed**: The clock speed is set to 50 MHz to ensure balanced performance and power consumption across the system.

IV. Optimization

Optimizing the MPSoC architecture for JPEG encoding involves several key strategies aimed at improving performance, reducing resource usage, and ensuring efficient memory management. The optimizations implemented can be categorized into memory management, custom hardware components, and pipeline enhancements. This section details each of these optimizations.

A. Memory Management Strategy

Efficient memory management is crucial for optimizing the performance of the JPEG encoding process, particularly in resource-constrained environments. The design of our MPSoC architecture for JPEG encoding incorporates several strategies to ensure optimal memory usage and reduce latency. A two-step memory allocation strategy is employed:

1) **Reducing Code Memory Footprint**: To fit the memory requirements of the processing stages within the available on-chip memory, we minimized the code memory footprint. Here we applied various BSP (Board Support Package) configuration flags to reduce unnecessary functionalities and optimize the code for memory usage. These included(Based on our experimental setup):
   - **Reduced Device Drivers**: Limits device drivers to save memory (enable reduced device drivers set to 1).
   - **Small C Library**: Uses a smaller C library to reduce memory usage (enable small c library set to 1).
   - **C++ Support**: Disables C++ functions in HAL footprint (enable c plus plus set to 0).
   - **Clean Exit**: Disables clean exit functionality (enable clean exit set to 0).
   - **JTAG UART Driver**: Enables a compact driver for JTAG UART interface (altera avalon jtag uart driver: enable small driver set to true).
   - **Lightweight API**: Enables a lightweight device driver API (enable lightweight device driver api set to 1).
   - **BSP Compilation Optimization**: Sets to -O2 for balance between optimization and compilation time (make bsp cflags optimization set to ”-O2”).
2) **On-Chip Memory Utilization**: With the reduced code footprint achieved through the first step, it became possible to relocate memory-intensive stages, such as Color Space Conversion, entirely to on-chip memory. Initially, SDRAM was used for these stages due to their high memory demands, which introduced a bottleneck in throughput. By leveraging the optimizations from the first step, we moved the Color Space Conversion stage to on-chip memory, enabling both the first processor and subsequent stages to operate within the on-chip memory, further boosting overall performance.
   For stages 1 and 6, we’ve chosen not to change the enable small c library and enable light-weight device driver api settings because we need to get input at stage 1 and give output at stage 6. This decision ensures that the necessary functionality for input/output operations is maintained while still optimizing other areas to reduce the overall memory footprint.

B. Custom Hardware Components

To further enhance performance, custom hardware components were integrated into the MPSoC architecture:

1) **Custom FIFOs for Stage 2 (Level Shifting)**:
   - **Implementation**: Hardware FIFOs with integrated level-shifting functionality replaced the dedicated CPU initially assigned to this stage.
   - **Rationale**: Level shifting, which involves adjusting pixel values by subtracting 128, is a straightforward yet repetitive task. Offloading this operation from the CPU to custom hardware FIFOs eliminates the need for a dedicated processor, freeing up resources for other stages and reducing overall CPU load. This approach streamlines the pipeline, enhances data throughput, and allows more efficient use of processing power. By integrating the level-shifting functionality directly into the hardware FIFOs, we ensure that the data is pre-processed and ready for subsequent stages, improving the overall system efficiency and speed.
2) **Custom Instruction Hardware Block for stage 3 (DCT)**:
   - **Implementation**: A custom instruction hardware block was developed to support the Discrete Cosine Transform (DCT) calculations in Stage 3.
   - **Rationale**: Through performance profiling, we identified that Stage 3, responsible for both Vertical and Horizontal DCT, was the most time-consuming stage in our processing pipeline. By adding a custom hardware block to handle the computationally intensive DCT operations, we offload these tasks from the CPU’s core logic, significantly improving the performance of Stage 3. This customization leads to

a substantial increase in throughput, enhancing the overall efficiency of the JPEG encoding process.

3) **Custom Instruction for Multiplication in Stage 4 (Quantization)**:
   - **Implementation**: A custom hardware instruction for multiplication was developed specifically to handle the frequent multiplications required in Stage 4 of the quantization process.
   - **Rationale**: Multiplication is a computationally intensive task, especially in the quantization process, where it is frequently applied to frequency domain coefficients. Since the Nios II/e processor lacks a built-in multiplication instruction, this custom hardware block became essential to efficiently support the quantization stage. The custom instruction significantly reduced the load on the CPU by offloading these computations, thereby enhancing the processing speed of Stage 4. By addressing the absence of a native multiplication instruction in the Nios II/e, this optimization minimized processing time and improved the overall throughput of the JPEG encoding pipeline, effectively mitigating a key bottleneck and enabling faster, more efficient data processing.

C. Pipeline Enhancements

Enhancements to the pipeline architecture focused on improving data flow and parallel processing capabilities:

1) **Superscalar Pipelines for Stage 3 (DCT)**:
   - **Implementation**: Superscalar pipelines were introduced for the DCT stage, allowing parallel processing of Y, Cb, and Cr data planes.
   - **Rationale**: DCT is a critical bottleneck due to its computational intensity. By enabling parallel processing, we significantly increase throughput and reduce the time required to process each image block. uperscalar pipelines allow multiple instructions to be processed simultaneously, leveraging the inherent parallelism in the DCT stage. This approach enhances the efficiency of the pipeline, leading to faster encoding times and improved overall performance of the JPEG encoder.
2) **Superscalar Pipelines for Stage 4 (Quantization)**:
   - **Implementation**: Superscalar pipelines were also introduced for the Quantization stage, facilitating parallel processing of quantization tasks.
   - **Rationale**: Similar to the DCT stage, quantization involves computationally intensive operations. By implementing superscalar pipelines, we enable parallel processing of quantization tasks, thereby reducing the time required for this stage and improving the overall throughput of the JPEG encoder. This parallel processing capability ensures that the quantization stage does not become a bottleneck, contributing to more efficient and faster JPEG encoding.
3) **Hardware FIFO Depth Adjustment**:
   - **Implementation**: The depth of each hardware FIFO queue was adjusted to 128 to meet the 8x8 block processing requirements.
   - **Rationale**: Setting the FIFO depth to 128 ensures smooth data flow and prevents bottlenecks between stages, as it accommodates the 8x8 data blocks effectively. This adjustment minimizes idle times, improving overall throughput and efficiency in JPEG encoding by maintaining continuous data transfer across the pipeline.

Fig 2 below illustrates the fully optimized pipeline, highlighting the improvements achieved through the implementation of the superscalar pipeline and other optimizations discussed.

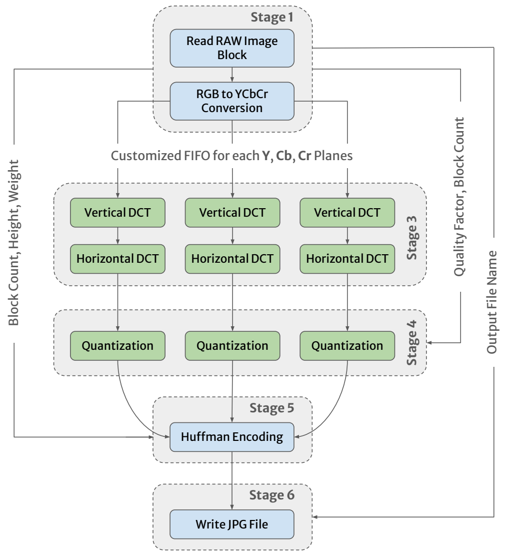

Fig. 2. JPEG Encoding Process Stages After all the Suggesting Optimizations

V. Experimental Setup

The MPSoC hardware design was developed in Quartus II, freely available software provided by intel. Within Quartus II, the Qsys system integration tool was used to construct the MPSoC hardware, leveraging pre-built IP blocks for various hardware components and utilizing Qsys's capability to create custom hardware blocks. After creating the MPSoC architecture in Qsys, the design was imported into Quartus II, where pin mappings were applied, and the system was analyzed, synthesized, and programmed onto the Cyclone IV E FPGA on the DE2-115 board[10]. TABLE I shows the flow summary of the final optimized design with superscaling.

For the software setup, Nios II Software Build Tools for Eclipse were employed to develop and run the software on each processor stage within the FPGA. Separate application software and Board Support Packages (BSPs) were generated for each processor, with the memory management strategies described in Section 3 applied through the BSP editor for each processor.

TABLE I
FPGA SYNTHESIS RESOURCE UTILIZATION ON CYCLONE IV E

Resource	Utilization
Total logic elements	25,552 / 114,480 (22%)
- Total combinational functions	23,633 / 114,480 (21%)
- Dedicated logic registers	11,695 / 114,480 (10%)
Total registers	11,712
Total pins	58 / 529 (11%)
Total virtual pins	0
Total memory bits	3,136,512 / 3,981,312 (79%)
Embedded Multiplier 9-bit elements	258 / 532 (48%)
Total PLLs	1 / 4 (25%)

For data transfer and timing measurements, a set of sample images with resolutions of 512x512, 256x256, 128x128, 64x64, and 512x256 were transmitted from a host computer to the FPGA board. Over 250 images were processed to calculate average throughput. Encoded images were sent back to the host computer for storage, while processing times for each stage and the overall pipeline were measured during execution. This setup enabled the evaluation of throughput and processing efficiency across various image sizes and configurations, demonstrating the impact of optimization strategies in a resource-constrained environment.

VI. RESULTS

The optimization of the MPSoC architecture for JPEG encoding on the Altera DE2-115 FPGA yielded significant performance improvements. This section presents the performance evaluation results of the initial setup and the implemented optimisations' impact.

A. *Initial System Performance*

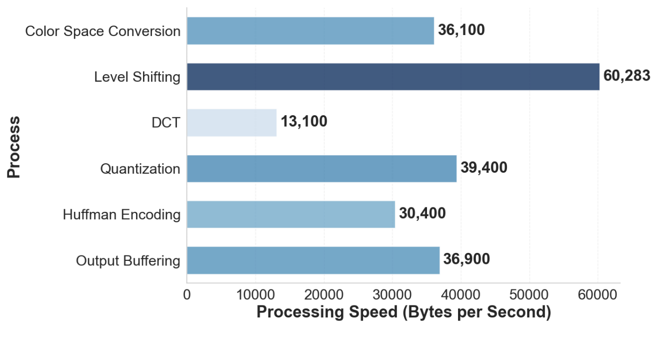

Fig. 3. Processing Speeds of each stage in Bytes per Second - **before optimization**

Above Fig. 3 presents the processing speeds of each stage in the initial, unoptimized JPEG encoding pipeline. As can be observed, the DCT stage emerges as the most significant bottleneck, hindering overall system performance. Also drawing insights from Wallace et al.[11], we recognized the critical role of the DCT stage optimization in the JPEG encoding process.

The overall system throughput was approximately **11,100 bytes per second**.

B. *Optimized System Performance*

To evaluate the impact of the proposed optimizations, we measured the processing speeds of each stage after their implementation. As depicted in Fig. 4 and Fig. 5, the optimizations resulted in a substantial increase in throughput, particularly for the DCT and quantization stages.

Below Fig. 4 illustrates the processing speeds of each stage in the JPEG encoding pipeline after the introduction of custom hardware components.

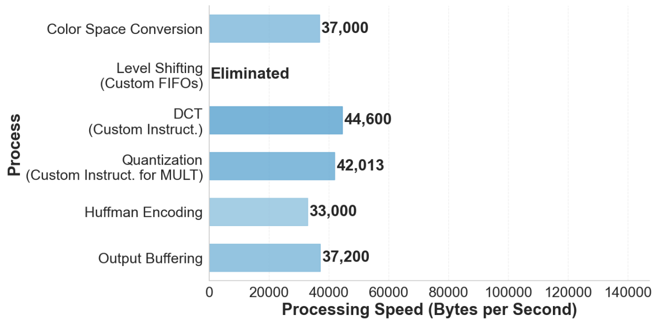

Fig. 4. Processing Speeds of Each Stage After Introducing Custom Instructions

The Fig. 5 depicts the processing speeds of each stage in the JPEG encoding pipeline after incorporating pipeline enhancements, such as superscalar pipelines, in addition to custom hardware components. The impact of superscalar pipelines on the DCT and quantization stages is evident in the increased throughput.

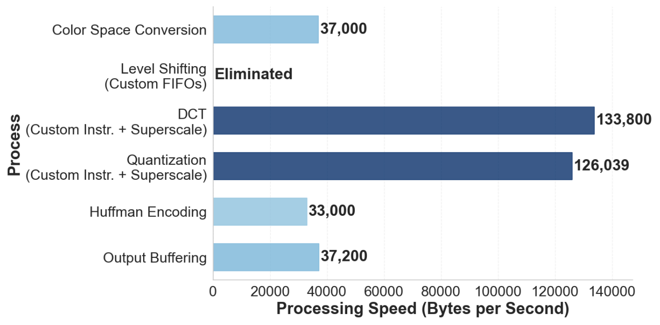

Fig. 5. Processing Speeds of Each Stage After Including Pipeline Enhancements

Here, we can now identify the bottleneck: the Huffman encoding stage. If we can find a way to optimize this stage, we could achieve a slight overall performance increase.

As we can see in Fig. 6 The overall system throughput increased to **31,000 bytes per second**, representing an approximate 2.78x increase compared to the initial design.

Additionally, we observed that integrating superscale in both the DCT and Quantization stages did not lead to significant improvements in overall performance. This lack of impact is likely due to the influence of other stages.

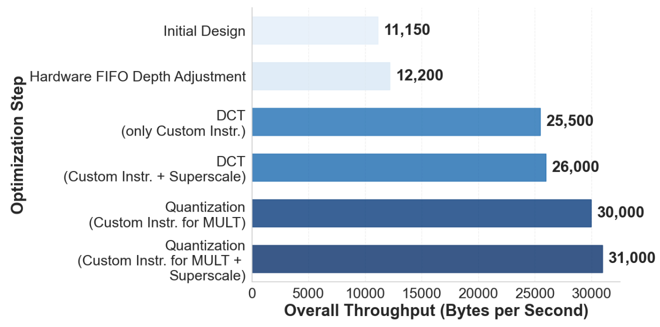

Fig. 6. Overall System Performance After Each Optimization

As a result, we have decided not to include the superscaling pipeline in our final design (Fig. 7). The implementation would require substantial hardware costs, and the performance gains would not justify the investment.

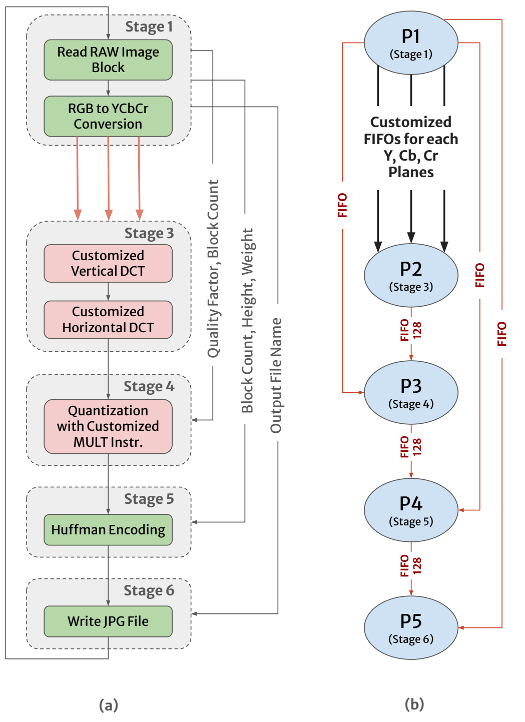

Fig. 7. (a) JPEG Encoding Process Stages in Final Optimized Pipeline. (b) Five processors and three customized FIFOs are configured in the pipeline

VII. Conclusion

The optimization of the Multi-Processor System-on-Chip (MPSoC) architecture for JPEG encoding demonstrates significant performance improvements and resource efficiency. By strategically addressing bottlenecks in the encoding pipeline and leveraging custom hardware components, we achieved a substantial increase in throughput while minimizing resource usage.

Initially, the system's performance was constrained by the high memory demands and communication delays associated with the SDRAM. By reallocating memory to on-chip resources and optimizing the hardware FIFO depths, we eliminated these bottlenecks and enhanced data flow between stages.

Implementing custom instructions for the Discrete Cosine Transform (DCT) and quantization stages further accelerated the computationally intensive operations, significantly boosting the processing speeds. The introduction of superscalar pipelines allowed for parallel processing, effectively tripling the throughput in the DCT and quantization stages.

Our optimized design achieved an overall system throughput of 31,000 bytes per second, representing a 2.78x improvement over the initial design. This work underscores the potential of FPGA-based MPSoC architectures for efficient and high-performance JPEG encoding, making it a viable solution for real-time image processing applications in resource-constrained environments.

Future work will explore further optimizations and the integration of additional custom hardware blocks to enhance performance and extend the capabilities of the JPEG encoder. The methodologies and results presented in this paper provide a robust framework for developing efficient MPSoC systems for various computationally intensive applications.

References

[1] A. Erdos, S. Parameswaran and S. L. Shee, "Heterogeneous multiprocessor implementations for JPEG:: a case study," Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '06), Seoul, Korea (South), 2006, pp. 217-222, doi: 10.1145/1176254.1176307.

[2] H. Javaid and S. Parameswaran, "Synthesis of heterogeneous pipelined multiprocessor systems using ILP: JPEG case study," Proceedings of the 6th International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2008, Atlanta, GA, USA, October 19-24, 2008, pp. 1–6, Oct. 2008, doi: 10.1145/1450135.1450137.

[3] J. Wu, J. Williams, N. Bergmann and P. Sutton, "Design Exploration for FPGA-Based Multiprocessor Architecture: JPEG Encoding Case Study," 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines, Napa, CA, USA, 2009, pp. 299-302, doi: 10.1109/FCCM.2009.7.

[4] Y. -L. Lee, J. -W. Yang and J. M. Jou, "Design of a distributed JPEG encoder on a scalable NoC platform," 2008 IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 2008, pp. 132-135, doi: 10.1109/VDAT.2008.4542430.

[5] C. Wang, X. Li, P. Chen, and X. Zhou, "A case study of parallel JPEG encoding on an FPGA," Journal of Parallel and Distributed Computing, vol. 78, pp. 1–5, Oct. 2014, doi: 10.1016/j.jpdc.2014.09.010.

[6] L. Kunz, G. Girão, and F. R. Wagner, "Evaluation of a hardware transactional memory model in an NoC-based embedded MPSoC," pp. 85–90, Sep. 2010, doi: /10.1145/1854153.1854177.

[7] Roko Grubišić and V. Zadrija, "Design of a System-Level Pipelined JPEG Coder for a Homogeneous Multiprocessor Platform Using Replication," May 01, 2009.

[8] H. Salamy, "An Effective Technique to Higher Throughput for Streaming Applications on an MPSoC," Journal of Computers, vol. 14, no. 8, pp. 507–518, Jan. 2019, doi: 10.17706/jcp.14.8.507-518.

[9] T. Kluter, P. Brisk, E. Charbon, and P. Ienne, "MPSoC Design Using Application-Specific Architecturally Visible Communication," Lecture Notes in Computer Science, pp. 183–197, 2009, doi: 10.1007/978-3-540-92990-1_15.

[10] "DE2-115 User Manual." Available: https://www.terasic.com.tw/attachment/archive/502/DE2_115_User_manual.pdf

[11] G. K. Wallace, "The JPEG still picture compression standard," in IEEE Transactions on Consumer Electronics, vol. 38, no. 1, pp. xviii-xxxiv, Feb. 1992, doi: 10.1109/30.125072.