Basic Image Processing and XSCT Data transfer Implementation over FPGA

Seminar Report

by

Vinaykumar Yadav
(23m1201)

Under the guidance of

Prof. Sachin Patkar

Department of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Mumbai - 400076, India

November, 2024

Abstract

This seminar report explores the implementation of basic image processing techniques, specifically edge detection, and data transfer between DDR and System using the Xilinx Software Command-line Tool (XSCT). The report describes the development of a Zynq-based image processing system leveraging a combination of hardware and software to achieve efficient image convolution. The system architecture includes components such as the Processing System (PS), Direct Memory Access (DMA) controller, and Image Processing RTL on the Programmable Logic (PL). Edge detection is implemented using convolution operations of image data with standard Edge detection kernel.

Furthermore, This report demonstrates basic data transfer between DDR and PC over XSCT. The XSCT workflow includes transferring binary files to DDR memory, monitoring buffer memory addresses, and retrieving processed output. This project underscores the potential of FPGA-based platforms for scalable and real-time image processing applications, paving the way for advancements in 2D and 3D mapping.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Image processing plays a crucial role in a wide range of modern applications, from autonomous systems and robotics to medical imaging and video surveillance. This report explores the implementation of edge detection, one of the fundamental techniques in image processing, which is used to identify object boundaries and features within images. By leveraging a Zynq-based platform, the project demonstrates how hardware and software can work together to achieve efficient image processing.

The system architecture integrates a Processing System (PS) for managing operations and a Programmable Logic (PL) for executing Image Processing IP. Data is exchanged between the DDR memory and the image processing modules using a Direct Memory Access (DMA) controller, ensuring seamless and fast communication. Edge detection is implemented through hardware convolution, which processes a 3x3 pixel window to highlight changes in intensity and detect edges efficiently.

The Data Transfer between a DDR and PC can be enhanced using XSCT of which basic implemnetation is shown in this Report.

This Report demonstrates not only the successful implementation of edge detection but also the practical integration of programmable hardware and software tools.

The approach highlights the potential of FPGA-based systems for scalable and high-performance image processing, paving the way for applications in real-time environments. This report details the system's development process, from architecture design and module configuration to results validation, offering a comprehensive overview of the work done.

Chapter 2

Literature Review

2.1 Introduction to Stereo Vision

Stereo vision systems aim to estimate depth by analyzing the disparity between two images captured from slightly different perspectives. Using stereo cameras, depth maps are generated to create 3D reconstructions. Applications include autonomous vehicles, UAV navigation, and robotic vision.The Project Sinha (2023) shows a complete process of implementing SGM over FPGA using HLS tools.

2.2 Semi-Global Matching (SGM)

Semi-Global Matching (SGM) is a disparity estimation algorithm that aggregates matching costs along multiple paths. It minimizes a cost function:

$$\begin{aligned} L_r(p,d) = C(p,d) + \min \Big(& L_r(p-r,d), \\ & L_r(p-r,d-1) + P_1, \\ & L_r(p-r,d+1) + P_1, \\ & \min_i L_r(p-r,i) + P_2 \Big) - \min_k L_r(p-r,k) \end{aligned} \tag{2.1}$$

where:

- $C(p,d)$: Matching cost between the left and right images at a pixel p and disparity d.
- P_1, P_2: Penalties for minor and major disparity changes, promoting smooth disparity transitions.

SGM balances the trade-off between local and global methods, providing high accuracy while being computationally intensive.

2.3 More Global Matching (MGM)

MGM is an enhancement of SGM, designed to reduce streaking artifacts by aggregating costs from fewer paths and averaging them:

$$L_r(p,d) = C(p,d) + \frac{1}{n}\sum_{x\in\{r_n\}} \min\Big(L_r(p-x,d),$$
$$L_r(p-x,d-1)+P_1,$$
$$L_r(p-x,d+1)+P_1,$$
$$\min_i L_r(p-x,i)+P_2\Big) - \min_k L_r(p-x,k) \quad (2.2)$$

where n is the number of paths considered.

2.3.1 Comparison of SGM and MGM

Aspect	**SGM**	**MGM**
Path Count	Typically 8	Reduced (e.g., 4)
Memory Usage	High	Lower
Accuracy	High in textured areas	High overall
Artifacts	Streaking prone	Reduced streaking
Computation Time	Higher	Lower

Table 2.1: Comparison of SGM and MGM.

2.4 Process Flow Diagram

The summarized process flow for the system mentioned in the project is as follows:

1. **Capture Images**: Stereo cameras acquire left and right views.
2. **Rectification**: Correct distortion and align images.
3. **Compute Disparity**: FPGA computes disparity maps using Semi-Global Matching (SGM) or Modified Global Matching (MGM).
4. **Generate Depth Map**: Disparity maps are converted to depth values.
5. **3D Map Visualization**: Octomap generates point clouds for visualization.
6. **Application Integration**: Disparity maps are used for tasks like visual odometry.

2.5 Hardware Implementation on FPGA

2.5.1 Platform Details

The disparity computation using SGM/MGM was tested on FPGA platforms such as:

- **Zedboard:** Achieved 10.5 fps with power consumption of 0.72 W.
- **Ultra96:** Enhanced pipelining enabled 16 fps.
- **ZCU104:** Achieved 84 fps with optimized resource usage.

2.5.2 Challenges in Implementation

- Managing high memory requirements for SGM.
- Balancing accuracy and resource utilization on FPGA.
- Adapting algorithms to hardware constraints while ensuring real-time performance.

2.6 Future Directions in Literature

- **Enhanced SLAM:** Integration of SLAM for real-time mapping and navigation.
- **Dynamic Scene Adaptation:** Improved algorithms for handling texture-less and dynamic environments.
- **Algorithmic Optimizations:** Exploration of hybrid methods combining SGM/MGM with machine learning.

Chapter 3

Edge Detection Implementation

3.1 Introduction

Edge detection is a key technique in image processing and computer vision, used to identify the boundaries within images. This chapter explores the implementation of edge detection using a Zynq-based image processing system Xilinx (2024b). The system combines the strengths of both the Processing System (PS) and the Programmable Logic (PL) to handle image processing tasks efficiently.

By utilizing Direct Memory Access (DMA) for fast data transfer and performing convolution operations in the PL, the system can detect edges in faster and efficiently. The chapter provides an overview of the different components of the system, the data flow, and the specific modules that make edge detection possible.

3.1.1 System Architecture

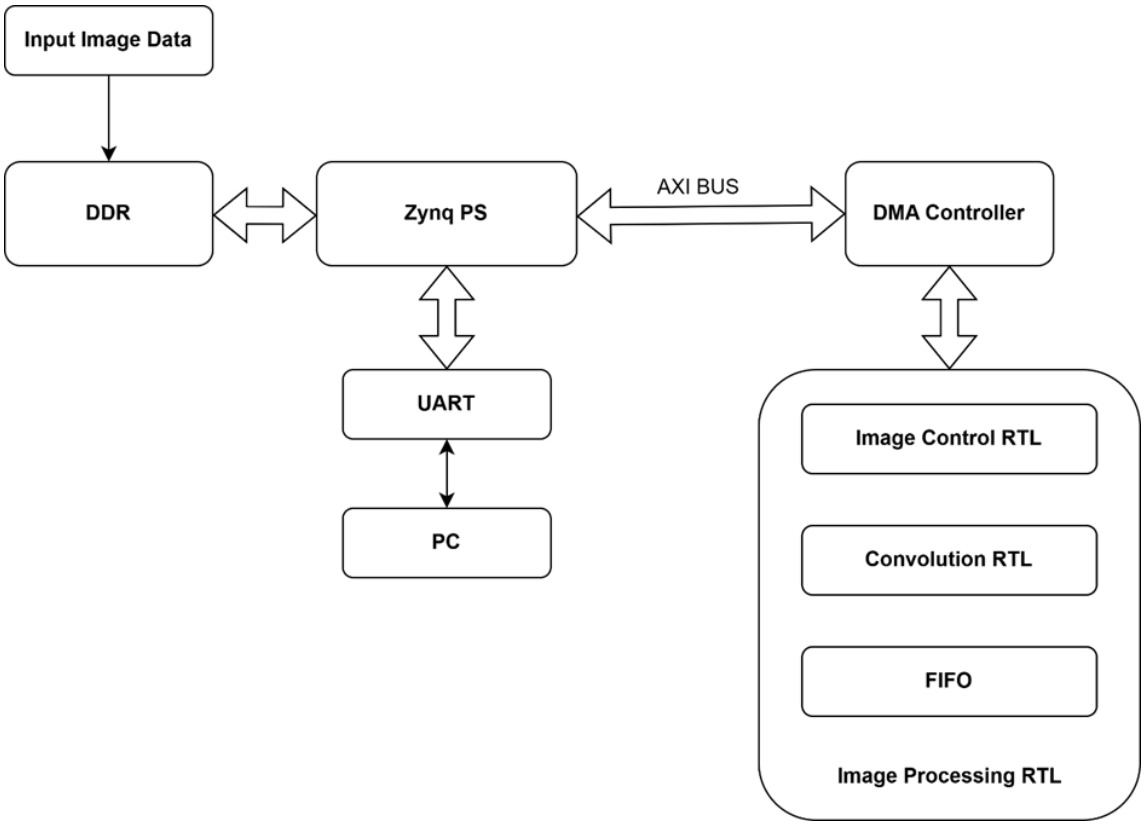

FIGURE 3.1: System Architecture

This System architecture illustrates the Zynq-based image processing system. Below is a brief explanation of the architecture:

1. **Input Image Data:**
    - The system starts with an input image stored in the DDR memory. This image is used as the source data for the image processing task.
2. **DDR Memory:**
    - The DDR memory acts as the main storage for both the input image and the processed image. It is accessible by both the Processing System (PS) and the Programmable Logic (PL).

3. **Zynq PS (Processing System):**

   - The Zynq Processing System (PS) serves as the control center. It manages communication between the DDR, UART, and the DMA controller.
   - The PS configures the AXI DMA controller to handle data transfers and triggers the image processing pipeline.

4. **DMA Controller:**

   - The DMA (Direct Memory Access) controller facilitates high-speed data transfer between the DDR and the Image Processing RTL in the PL without involving the CPU.
   - **Data flow:**
     - Input image data is sent from DDR to the Image Processing RTL through the DMA-to-Device transfer.
     - The processed image is received back from the Image Processing RTL and stored in DDR through the Device-to-DMA transfer.

5. **Image Processing RTL:**

   - The Image Processing RTL in the Programmable Logic (PL) performs the actual image processing task . It consists of three key blocks:
     - Image Control RTL: Manages data flow and controls the image processing pipeline.
     - Convolution RTL: Implements the convolution operation or other image processing algorithms.
     - FIFO: Buffers the data to ensure smooth and efficient processing.

6. **UART:**

    - The UART (Universal Asynchronous Receiver/Transmitter) serves as a communication interface between the Zynq PS and an external PC.
    - Once the image processing is complete, the PS retrieves the processed image data from DDR and sends it to the PC via UART.

7. **PC:**

    - The external PC acts as the final destination where the processed image data is displayed or analyzed. The processed data is transmitted byte-by-byte over the UART connection.

3.2 Edge Detection IP Working

RTL of Edge Detection IP is based on the Neighborhood Image Processing concept. it's a method in digital image processing where the pixel values of an image are modified based on the values of their neighboring pixels. This approach involves applying a filter or kernel (a small matrix, such as 3x3 or 5x5) over the image in a sliding window manner. The kernel defines the relationship between a pixel and its neighbors to achieve specific effects. Below is the key concepts involved in Neighborhood Image Processing concept.

- **Neighborhood**: The group of pixels around a target pixel, often defined by the size and shape of the kernel (e.g., 3x3 square).
- **Kernel Operations**: Mathematical operations (like averaging, weighting, or edge detection) applied to the pixel and its neighbors.

- **Applications**:
  - **Smoothing**: Reduces noise (e.g., using a mean or Gaussian filter).
  - **Edge Detection**: Highlights edges in an image (e.g., Sobel or Laplacian filter).
  - **Sharpening**: Enhances details and boundaries in an image.
- **Local Processing**: Modifications occur locally, ensuring that each pixel's new value depends only on its neighborhood.

To implement this Neighborhood Image Processing concept , Edge Detection IP consist of three different modules :

3.2.1 ImageControl Module

- **Purpose:** Manages 3x3 pixel windowing for convolution.
- **Working:**
  - Buffers incoming pixel data into 4 line buffers (lineBuffer instances) and extracts 3x3 pixel blocks (`o_pixel_data`).
  - Uses pixel Counter and line buffer control signals to determine read/write operations.
  - Issues an interrupt (`o_intr`) when enough data has been processed or buffer conditions are met.
  - **Outputs:**
    - `o_pixel_data`: 3x3 pixel block (72 bits) for the conv module.
    - `o_pixel_data_valid`: Indicates validity of the 3x3 block.

3.2.2 Convolution Module

- **Purpose:** Performs convolution on 3x3 pixel data using two predefined kernels.
- **Working:**
  - Multiplies each pixel in the 3x3 block with corresponding kernel values.
  - Computes the sum of products for each kernel, squares the sums, combines them, and thresholds the result.
  - **Outputs:**
    - `o_convolved_data`: Binary result (edge-detected pixel).
    - `o_convolved_data_valid`: Indicates the validity of convolution output.

3.2.3 OutputBuffer Module

- **Purpose:** Buffers and streams convolution results to the master interface.
- **Working:**
  - Stores convolution outputs (`convolved_data`) until the master interface (`o_data_ready`) is ready to accept data.
  - Handles handshake signals for efficient data transfer.
  - **Outputs:**
    - `o_data`: Final processed data.
    - `o_data_valid`: Indicates availability of output data.

3.2.4 Top-Level Module imageProcessTop

- **Purpose:** Integrates the imageControl, conv, and outputBuffer modules.
- **Working:**
  - Receives pixel data (`i_data`) via a slave interface.
  - Passes the data to imageControl for windowing and then to conv for convolution.
  - Sends the convolved data to outputBuffer for output streaming.

3.3 Hardware Block Diagram - Vivado

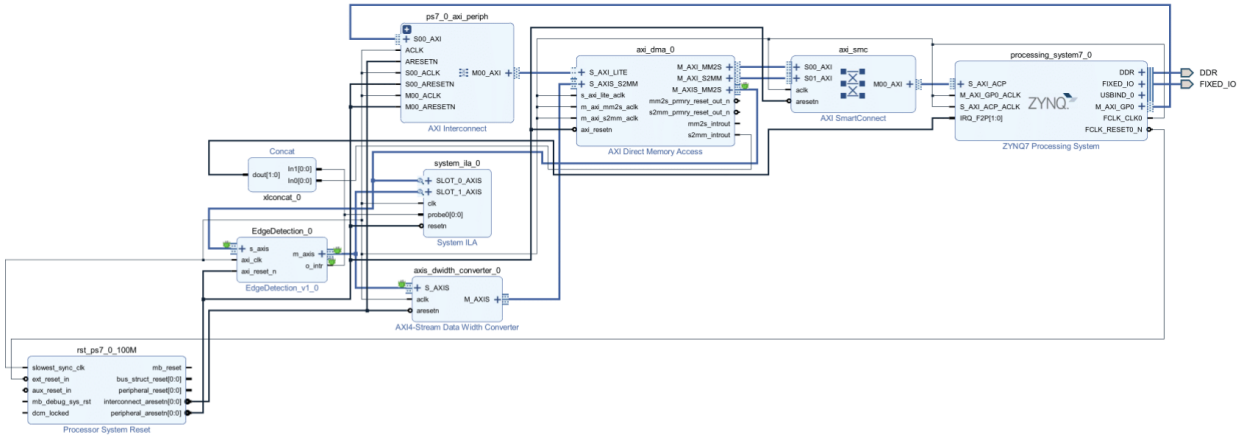

FIGURE 3.2: Hardware Block diagram

This is a Hardware block diagram made in Vivado which incorporates the Edge detection IP along with Zynq processor and DMA controller.Below is the process flow of this Block Diagram

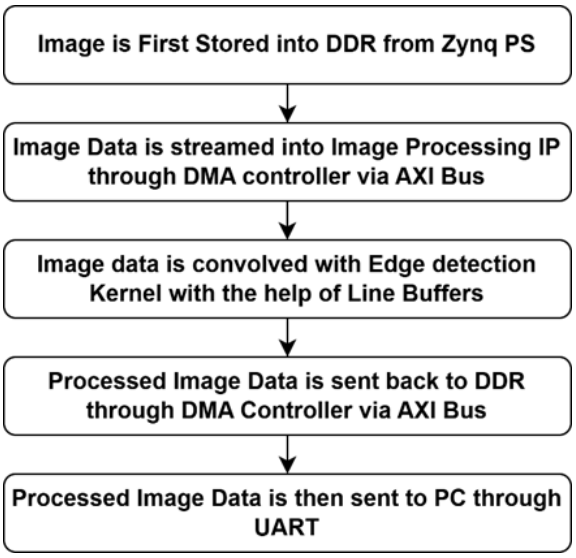

FIGURE 3.3: Process Flow

3.4 Zynq PS Software Logic

The Zynq PS sends the image Data to DDR and then it is transferred to Edge detection IP via AXI bus . The processed image is then sent to DDR again via AXI bus from Edge detection IP to DDR. And then the processed image is sent to PC through USB port. below is the detailed explaination of Code flow :

1. UART Initialization

- Configures UART using `XUartPs_LookupConfig` and `XUartPs_CfgInitialize`.
- Sets the baud rate to 115200.

2. DMA Initialization

- Configures the AXI DMA using `XAxiDma_LookupConfigBaseAddr` and `XAxiDma_CfgInitiali`

- Enables interrupts for the DMA transfer completion (`XAXIDMA_IRQ_IOC_MASK`).

3. Interrupt Controller Initialization

- Configures the interrupt controller using `XScuGic_LookupConfig` and `XScuGic_CfgInitialize`.
- Sets interrupt priorities and types for:
  - Edge Detection IP core interrupt.
  - AXI DMA interrupt.
- Registers handlers `imageProcISR` and `dmaReceiveISR`.

4. Enable Interrupts

- Initializes and enables interrupts using `Xil_ExceptionInit`, `Xil_ExceptionRegisterHandler`, and `Xil_ExceptionEnable`.

5. DMA Transfers

- Starts an initial transfer of the image data to the AXI DMA for:
  - **Device-to-DMA**: Transfers processed image data back to the memory.
  - **DMA-to-Device**: Sends image data to the Edge Detection hardware.

6. Wait for Processing Completion

- The program waits in a `while (!done)` loop until the `dmaReceiveISR` signals completion.

7. UART Data Transmission

- Transmits processed image data over UART byte by byte using `XUartPs_Send`.
- Uses a delay (`usleep`) between transmissions to ensure proper communication.

8. Helper Functions

1. checkIdle

- Checks whether the DMA is idle by reading the specified register and masking it with `XAXIDMA_IDLE_MASK`.

9. Interrupt Service Routines (ISRs)

1. imageProcISR

- Handles the interrupt from the Edge Detection hardware.
- Ensures the DMA is idle before starting the next data transfer.
- Continues transferring image data in chunks (one row at a time).

2. dmaReceiveISR

- Handles the interrupt from the AXI DMA.
- Disables DMA interrupts, acknowledges the interrupt, and signals that processing is complete by setting `done = 1`.
- Re-enables the DMA interrupt for future operations.

For transferring the data to PC, Tera Term Software is taken in usage whose COM port and Baud rate is first defined and then the log file is saved as output.bmp.Once the image is fully transferred (512 x 512) 438 Bytes Header from the original gray scale image is then added in the header section of processed output image. Input image Hex file need to be stored in Zynq PS code as a character format which can be generated by simulating Image detection IP and then it can be edited as a character format and then can be included in C file of Zynq PS.

3.5 Result

FIGURE 3.4: Input Image

FIGURE 3.5: Output Image

The image on the left corresponds to the input image which was stored in DDR and the Right Image corresponds to the Edge Detection Output which is sent back to the PC via UART.

Chapter 4

XSCT - Standard Example Implementation

4.1 Hardware Block Diagram

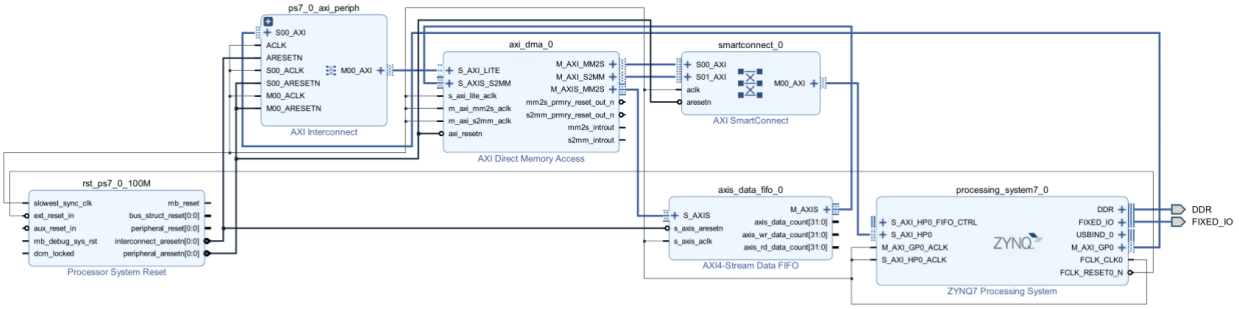

FIGURE 4.1: Hardware block Diagram - XSCT

The block diagram illustrates the integration of a Zynq-7 Processing System with an AXI Direct Memory Access (DMA) controller acting as a basic Hardware for XSCT example design. The Zynq Processing System (`processing_system7_0`) provides the central processing and control, interfacing with DDR memory, Fixed IO, and AXI

interconnects. The AXI DMA (`axi_dma_0`) enables high-speed data transfer between memory and peripherals using AXI Stream interfaces. The `axis_data_fifo_0` acts as a buffer for streaming data, while the `smartconnect_0` ensures efficient routing of AXI transactions. The system reset logic (`rst_ps7_0_100M`) manages initialization and synchronization of components. This design demonstrates efficient data handling with minimal CPU involvementXilinx (2024a).

4.2 Software Logic for XSCT Data Transfer

This section describes the software logic for implementing AXI DMA data transfer in XSCT. The logic consists of the following key functions:

4.2.1 Main Function

- Initializes the program and runs the `XAxiDma_SimplePollExample` function.
- Prints whether the example succeeded or failed.
- Returns a status code (`XST_SUCCESS` or `XST_FAILURE`).

4.2.2 XAxiDma_SimplePollExample Function

This core function manages the AXI DMA configuration and data transfer process:

1. **Initialize Variables:** Sets up pointers for transmit (`TxBufferPtr`) and receive (`RxBufferPtr`) buffers using predefined memory addresses.
2. **Configuration Lookup:** Retrieves the DMA hardware configuration using `XAxiDma_LookupConfig`. Exits with an error if not found.

3. **Initialize DMA:** Configures the DMA hardware with `XAxiDma_CfgInitialize` and ensures it is in simple mode.

4. **Disable Interrupts:** Configures the DMA for polling mode by disabling interrupts.

5. **Data Preparation:** Populates the transmit buffer (`TxBufferPtr`) with a sequence of values and flushes the data cache for consistency.

6. **DMA Data Transfers:** For the specified number of transfers:

   - Sets up data transfers using `XAxiDma_SimpleTransfer` for transmission and reception.
   - Polls `XAxiDma_Busy` to wait for transfer completion.

7. **Data Verification:** Calls the `CheckData` function to ensure the received data matches the transmitted data.

4.2.3 CheckData Function

Validates the data after the DMA transfer:

1. **Cache Handling:** Invalidates the destination buffer to ensure correct data is read from memory.

2. **Data Comparison:** Compares the transmit and receive buffers. Logs errors for mismatches or success for correct matches.

3. Returns `XST_SUCCESS` if all data matches, otherwise exits with an error.

This software logic ensures efficient DMA-based data transfers with verification, using polling mode for simplicity and reliability.

4.3 XSCT Commands - for Data Transfer

The following XSCT commands are used for transferring data to and from the DDR memory in the Zynq system:

- **Send Data to DDR:**

  ```
  dow -data <filename> <address>
  ```

  This command transfers a binary file (`<filename>`) to a specified address in DDR memory (`<address>`).

- **Receive Data from DDR:**

  ```
  mrd -size b -bin -file OutData.bin <address> <length>
  ```

  This command reads data from DDR memory starting at the specified address (`<address>`) and of the specified length (`<length>`), saving it as a binary file (`OutData.bin`).

4.4 Result

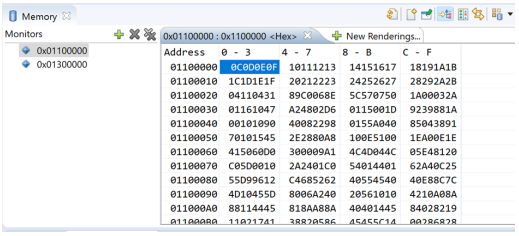

FIGURE 4.2: Initial DDR Data

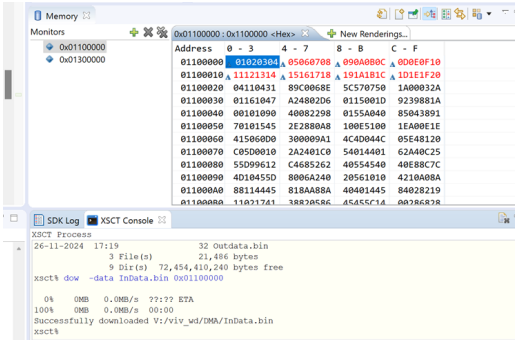

FIGURE 4.3: DDR Data after receiving Data from InData bin

Fig 4.2 corresponds to the Initial DDR Data and Fig 4.3 corresponds to DDR Data after 20 bytes was transferred to DDR from InData.bin file. This File was generated with the help of HXD Software.

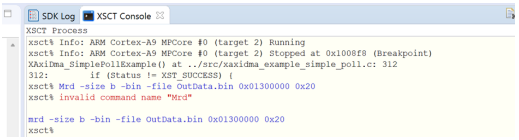

FIGURE 4.4: XSCT Console with read Command

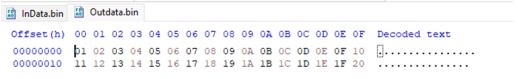

FIGURE 4.5: Outdata bin after DDR transfer

Fig 4.4 corresponds to XSCT console image in which the command for memory read is sent and the OutData.bin is generated in the PC and the data Outdata.bin is shown in FIg4.5 which is same as data of InData.bin .

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The Hardware Implementation of Basic Image Processing - Edge Detection and Basic XSCT based data transfer between DDR and PC has been presented. The integration of Edge Detection IP with the DMA controller ensures high-speed data transfers between the DDR and the programmable logic, while UART enables reliable communication with an external PC for processed data retrieval.

5.2 Future Scope

- This system can be further extended by using XSCT instead of UART for transferring Data from DDR to PC for High Speed Data transfer..
- Implementation of optimized Stereo Visual Odometry (SVO) on the FPGA for better real-time performance. The SGM Implementation study described in Literature Survey processes disparity maps on the FPGA, but odometry

calculations are done on the ARM processor.Porting odometry algorithms like feature tracking, pose estimation, or simultaneous localization and mapping (SLAM) to the FPGA can significantly speed up the system.

References

Sinha, Y., 2023. Phase 1 report on fpga implementation. URL: https://www.ee.iitb.ac.in/~hpc/yashwant_ddp_phase1.pdf. accessed: 2024-11-27.

Xilinx, 2024a. XSCT Reference Guide. https://users.ece.utexas.edu/~mcdermot/arch/articles/Zynq/ug1208-xsct-reference-guide.pdf.

Xilinx, 2024b. Zynq-7000 soc overview. Online. https://www.xilinx.com/support/documentation/data-sheets/ds190-Zynq-7000-Overview.pdf.

Image Processing Tutorial Retrieved from https://www.youtube.com/@Vipinkmenon

Prathmesh Sawant, FPGA Implementation of a Real Time Stereo Vision System, Mtech Dissertation Report.