ECE532 Final Report
Motion detection with FPGAs

Da Kang
Yi Qi Cao

Table of Contents

1. Overview

1.1 Goal

The goal of this project is that an FPGA monitors video signal from a camera and checks for motion. If motion is detected, the video is streamed to a web server for storage and later viewing. The server may receive video from multiple FPGAs and the server can do all sorts of post-processing according to various needs. This hardware can be used for security camera and smart home applications.

1.2 Block diagram

- Camera & Video Decoder: These IPs are from Nexys Video HDMI Demo.
- Motion detector: custom motion detection algorithm IP with AXI Lite interface.
- DDR memory: M_t, V_t and frame buffer are fed to motion detection algorithm IP.
- Ethernet Controller & Network & Server: Streaming frames from motion detection algorithm IP. over the ethernet.

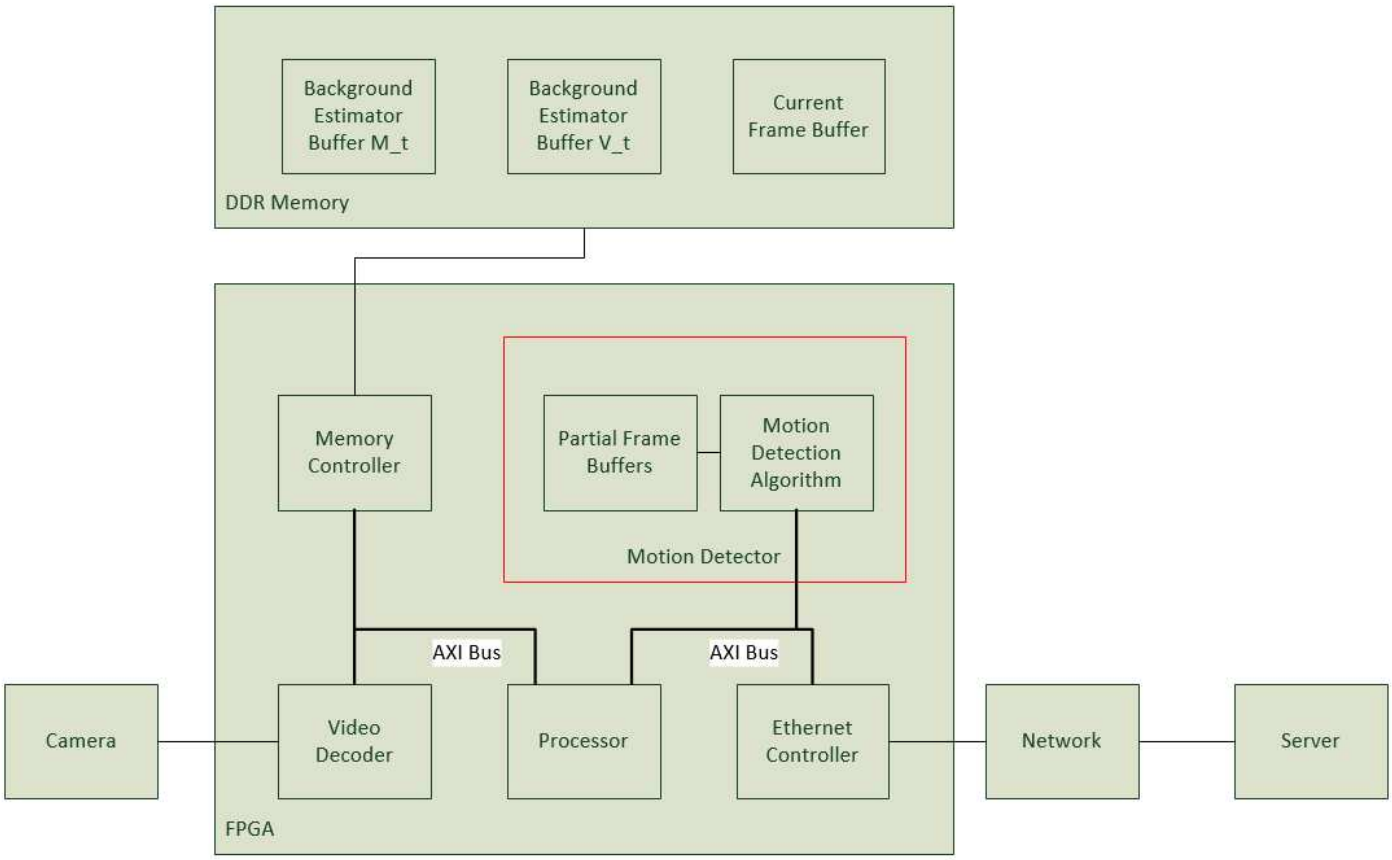

Figure 1. Block diagram

1.3 Brief description of IP

This table mainly shows IPs that our team added based on Nexys Video HDMI Demo.

IP instance name	Description	Source
motion_detector_final_0	Motion detection algorithm custom IP with AXI Lite interface.	Our Team
axi_ethernet_0	Ethernet controller	Xilinx
axi_ethernet _0_dma	Direct memory access for axi_ehternet_0	Xilinx
Rest of the IPs are all from Nexys Video HDMI Demo.		

2. Outcome

2.1 Results

The goals of this project are successfully completed. The motion detection algorithm IP is able to capture motion and FPGA could stream certain frames over the ethernet. The only bottleneck of this project is AXI Lite interface of the motion detection algorithm in IP can only transfer 3MB/sec. We tried AXI-full interface, but there were timing failures and it didn't work really well.

2.2 Possible Further improvement

- Motion detection algorithm IP interface could be changed to AXI FULL interface. It could significantly improve data transfer rate.
- Ethernet data transfer speed could be improved. Right now, we can only transfer 3MB/sec. Ideally, we can stream entire motion video.

3. Project schedule

Milestone #	Proposed	Delivered
1	Shown that motion detection algorithm works without post processing	Motion detection algorithm works without post processing
2	Shown Camera and Video decoder working.	Camera and Video decoder works.
3	Sending video over the network to the server	Ethernet part was not fully working.
4	Shown that motion detection algorithm works with post processing	Motion detection algorithm works with post processing and ethernet part works.
5	Controlling the motion detector component with	Wrote motion detection IP AXI lite interface

	the processor	
6	Testing and final integration#1	Integration and changed motion detection IP interface to AXI Full (Verified on behavioral simulation)
7	Testing and final integration#2	Final Integration and changed back motion detection IP interface to AXI Lite.

4. Description of the blocks

4.1 Block Design

The major of the IP blocks used in the project were sourced from Diligent tutorials: the Nexys Video HDMI Demo and the Getting Started with Microblaze Servers tutorials. The HDMI demo provided a zip file containing a tcl script and several IP blocks that were used to generate a basic project that performed HDMI capture and output. The servers tutorial was then followed to add Ethernet support by adding the AXI 1G/2.5G Ethernet Subsystem IP block and another DMA. Our motion detection custom IP block was then added to the project to complete the block design.

4.2 Motion Detection Algorithm

foreach *pixel* x **do** [step #1: M_t estimation]
  if $M_{t-1}(x) < I_t(x)$ **then** $M_t(x) \leftarrow M_{t-1}(x) + 1$
  if $M_{t-1}(x) > I_t(x)$ **then** $M_t(x) \leftarrow M_{t-1}(x) - 1$
  otherwise $M_t(x) \leftarrow M_{t-1}(x)$
foreach *pixel* x **do** [step #2: O_t computation]
  $O_t(x) = |M_t(x) - I_t(x)|$
foreach *pixel* x **do** [step #3: V_t update]
  if $V_{t-1}(x) < N \times O_t(x)$ **then** $V_t(x) \leftarrow V_{t-1}(x) + 1$
  if $V_{t-1}(x) > N \times O_t(x)$ **then** $V_t(x) \leftarrow V_{t-1}(x) - 1$
  otherwise $V_t(x) \leftarrow V_{t-1}(x)$
  $V_t(x) \leftarrow max(min(V_t(x), V_{max}), V_{min})$
foreach *pixel* x **do** [step #4: $\hat{E}_t$ estimation]
  if $O_t(x) < V_t(x)$ **then** $\hat{E}_t(x) \leftarrow 0$ **else** $\hat{E}_t(x) \leftarrow 1$

Figure 2. Motion detection algorithm.

The motion detection custom IP block is a direct translation of the basic motion detection algorithm in the paper found here: Motion Detection: Fast and Robust Algorithms for Embedded Systems. The algorithm operates on a pixel by pixel and frame by frame basis. It uses intermediate data called background estimators to check if a given pixel has changed. The estimator M_t tracks the expected value of the pixel, and the estimator V_t is the tolerance for difference in pixel intensity. The algorithm essentially checks for the absolute difference between current pixel intensity and M_t, and compares it to V_t. If the different is higher than the threshold

V_t, then the pixel is said to have changed. The background estimators are then updated: M_t is incremented or decremented to move it closer to the current pixel intensity, and V_t is adjusted upwards if the different in pixel intensity is big enough. Note that the algorithm input is pixel intensity and not RGB, so the video input must first be converted to black and white before it can be processed.

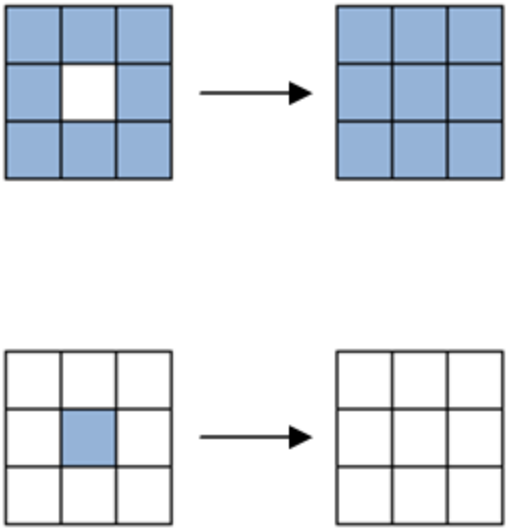

Figure 3. Post processing algorithm.

Our custom IP implements this motion detection algorithm for every pixel in a 16x8 pixel block. The output for each pixel is then post processed: standalone pixels are ignored, while unchanged pixels surrounded by changed pixels are considered changed. The borders of the block are not post processed, to make things simpler. The number of changed pixels is then counted to decide if the video frame has changed.

0x4410000	Data segment selection
0x4410004	Data type selection
0x4410008	Changed pixel count
0x441000C	Unused registers
0x4410020	Memory mapped data registers

Figure 4. AXI lite interface registers.

To transfer data to and from our custom IP block, we use an AXI lite interface. Our custom IP block has 3 buffers: one for the RGB video input, one for the M_t values, and one for the V_t values. The output values are not explicitly buffered: the output wires of the combinatorial logic in the algorithm are treated as the output values. The interface uses 3 registers for control and 8 registers for data. The 3 control registers are at the start of the address space, with the 1st register for data segment selection, the 2nd register for data type selection, and the 3rd register for the changed pixel count. The data registers are the 9th to 16th registers, which get memory mapped to different parts of the buffers. The 1st and 2nd register control which position and which buffer is being written to or read from: the data registers can be moved to access any part of the internal buffers.

This memory mapping scheme was chosen to reduce the number of writes needed to control which part of which buffer was being accessed, while keeping the number of registers required to a minimum. This scheme also allows for better data packing due to direct memory mapping: the RGB input for example has 3 bytes per pixel. Writing each pixel individually would waste 1 out of every 4 bytes. Data packing while writing non-aligned data like RGB data one word at a time could be very complicated to implement, making multiple registers a more attractive option. Side note: the block size was initially chosen as 16x9 to match the aspect ratio of the image, but was reduced to 16x8 to better fit the memory mapping scheme.

Testing the algorithm was relatively straightforward. The more complicated operations in the algorithm like absolute were split into a separate module that could be tested individually. The overall integrated IP block was tested by using an AXI VIP to control it, and by comparing the result with a Python implementation of the motion detection algorithm algorithm. The block design also allowed for direct checking of the buffers and algorithm outputs. The final design was tested by adding printing functions to the control code.

4.2 Microblaze Control Code

The Microblaze control code is based on the code provided by the HDMI demo, although the final design only uses the initialization code and option menu. The Ethernet and custom IP control code are integrated as a new option in the menu.

The HDMI control code used is just the video capture and output initialization code from the GDMI demo. Once the option to start the motion detection algorithm is chosen, the video output is disabled and no other video related code is run, aside from pausing video capture to send complete frames.

The Ethernet control code is almost entirely original, with the initialization code sourced from the SDK's built-in LwIP echo server project and the LwIP tutorial serving as a reference for the rest of the code. The code establishes a TCP connection with the server, and when connected sends an entire frame to the server one line of pixels at a time, utilizing the TCP send buffer as much as possible.

The custom IP block control code consists mostly of reading and writing from the address range of the custom IP block. A frame and its background estimators are written to the custom IP and the results read one 16x8 block at a time. Since the data transfer is the bottleneck and the custom IP is entirely combinatorial aside from some registers to meet timing, no real control commands are needed. While processing each block, the number of changed pixels is summed across all blocks. If the number of changed pixels meets a 10000 pixel threshold, then the frames are sent to the server.

The server code is an extension of the Python server provided for the warmup project. It simply receives data into a buffer the size of the frame, and when full dumps it into a bmp file with the correct file header.

5. Description of the design tree

- The Nexys-Video-HDMI directory contains the main project. This directory was the result of unzipping the zip file from the HDMI tutorial. The proj subdirectory contains the final block design, and the proj/HDMI.sdk subdirectory contains the videodemo project that is used to control the design.
- The motion_detector_2.0 directory contains the motion detection custom IP.
- The vip_testing directory contains the block design and testbench that use an AXI VIP to test the custom IP.
- The ethernet_tutorial directory contains the block design and sdk project that was used to implement and test sending video frames over Ethernet. The main sdk project used is named ethernet_tutorial.
- The python directory contains the Python server that receives the data sent by the FPGA, a script that converts video frame memory dumps to bmp files, and the software implementation of the motion detection algorithm.

6. Tips

Nexys Video HDMI Demo and LwIP tutorial are useful to get our design finished faster and we re-use their block design.