Image Coprocessor: A Real-time Approach Towards Object Tracking

Muhammad Shahzad, Saira Zahid
Next Generation Intelligent Networks Research Center (nexGIN RC),
National University of Computer and Emerging Sciences (NUCES-FAST),
Islamabad 44000, Pakistan.
{muhammad.shahzad, saira.zahid}@nexginrc.org

Abstract

Real time object tracking finds its applications in diverse fields. An online embedded system for image tracking should be fast, accurate, robust and efficient. This paper presents a hardware based architecture and implementation of an Image Coprocessor on FPGA using Verilog Hardware Description Language. The core concept is to ensure a fast and memory efficient dedicated hardware which could increase the efficiency manifolds and enhance the overall output, thereby making real time detection and tracking possible at much higher rates than achievable using C++. The design was simulated using Xilinx ISE Simulator and ModelSim. An embedded system was later on developed for complete implementation of Image Coprocessor on FPGA, using Xilinx Embedded Development Kit. Six modules including Edge Image, Bhattacharya Coefficient, Histograms, Epanechnikov Kernel, Translation and Rotation have been implemented on FPGA, and a comparison has been made with their implementation in C++. The results demonstrate up to 66 times improved performance of these individual modules. Thus, an efficient Image Coprocessor has been developed.

1. Introduction

1Real-time object detection and tracking is a critical task in many computer vision applications such as surveillance, driver assistance, gesture recognition and man machine interface. Research has been carried out for many years in this field. In this sequel, algorithms have been developed and tested on desktop machines [1]. However the overhead of the implemented schemes is excessive enough to hinder the successful implementation of these algorithms.

Due to its programming flexibility, easy availability and final design compactness FPGA technology has been used by many researchers to implement object tracking. The computational complexity of object tracking algorithms is quite high. The major contribution of this paper is implementation of six functions used in object tracking using Verilog Hardware Description Language (VHDL) and their performance comparison with the functions implemented using standard C++. The first attempt in this development was real-time frame grabbing which was done successfully. Video camera output was digitized, decoded, stored in SRAM and was displayed on the monitor. Second step was to incorporate a processor in this design for running complicated video processing algorithms e.g. object detection and tracking routines that cannot be directly translated into VHDL. So Xilinx soft core 32-bit processor known as Microblaze was chosen for this purpose. This microprocessor supports higher level languages e.g. C/C++.

The main focus of this paper is on describing an Image Coprocessor that works in parallel with Microblaze. This coprocessor has been implemented by using VHDL and it ensures that the computation of any feature requires least possible time and memory resources. Section 2 gives an overview of the complete embedded system used for object tracking. Section 3 describes the modules implemented in Image Coprocessor. Section 4 shows the performance improvement and finally Section 5 gives the conclusion and future work.

2. Architecture

The main diagram of the complete embedded system is shown in the Fig. 1. Analog composite video signal is con-
verted to digital form by video decoder card. Frame grabber module decodes this digital video signal at pixel level and stores in the memory shared with the microprocessor.

Video Display Unit is designed to display graphics, text and mouse cursor overlaid on the incoming streaming video. Note that streaming video is not displayed by reading back the memory, it is directly passed to video multiplexer module which overlay Video Display Unit module output on it and outputs the overlaid video to the video encoder module which again converts it to composite video signal and it is displayed on TV.

Microprocessor communicates with the Video Display Unit by Data Local Memory Bus (DLMB). Microprocessor also processes the frame stored in the shared memory. It accesses the memory by Xilinx Memory Link (XML) bus also called Xilinx Cache Link (XCL).

A dedicated hardware, image coprocessor, has been developed for applying a modular approach in hardware leading to overall enhanced efficiency. Instructions are exchanged with this Coprocessor via FSL (Fast Simplex Link). While LMB is used for data transfer and fetching.

We will explain the image coprocessor part in detail and indicate the improvement in performance resulting due to its use below.

3. Image Coprocessor

To implement the Coprocessor we have used 11 Kilo Bytes of Block Random Access Memories (BRAMs). These BRAMS are used to share the input and output data between processor and Co-Processor. The command instructions are given to the coprocessor using Fast Simplex Link (FSL). The overall architecture of Coprocessor is shown in Fig. 2.

There are six modules implemented in Image Coprocessor which are as follows.

1) Edge Image
2) Bhattacharya Coefficient
3) Histogram
4) Epanechnikov Kernel
5) Rotation
6) Translation

The Edge Image module, Translation module and Rotation module each reads the image from BRAM1, computes the edge image, translated image and rotated image respectively and stores the resulting image in BRAM2. BRAM1 and BRAM2 are 4KB in size each. Bhattacharya module reads the histograms from BRAM4 and BRAM5 and sends the coefficient value through FSL link to MicroBlaze. Histogram module reads the image from BRAM1 and has the option to store the calculated histogram in either BRAM4 or BRAM5. Epanechnikov Kernel module utilizes all the BRAMs to calculate the kernel profile. BRAM3, BRAM4 and BRAM5 all are 1KB each. So the total memory that our system uses is just 11KB.

3.1. Edge Image

To make the edge image we have used Robert’s cross edge detection operator [4]. The Roberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. It thus highlights regions of high spatial frequency which often correspond to edges. In its most common usage, the input to the operator is a grayscale image, as is the output. Pixel values at each point in the output represents the estimated absolute magnitude of the spatial gradient of the input image at that point.

Our aim was to accomplish the task in minimum number of cycles and using as lesser a memory as possible. Fig. 3 gives the Asynchronous State Machine (ASM) chart of edge calculation algorithm. Busy pin indicates that the edge image is being calculated. Addr1, Dout1, Din1, Addr2, Dout2, and Din2 are the Address lines, Data output lines and Data input lines of BRAMs 1 and 2 respectively.

3.2. Bhattacharya Coefficient

The Bhattacharya coefficient [5] is an approximate measurement of the amount of overlap between two statistical samples. The coefficient can be used to determine the relative closeness of the two samples being considered. Bhattacharya Coefficient is widely used for similarity measure [6]. The similarity function defines a distance between two probability density functions. In our case probability density functions are actually quantized histograms. Histogram description will follow this section. The sample estimate of
Bhattacharyya Coefficient between two histograms \( p \) and \( q \) with \( m \) elements each, is defined in (1):

\[
Bhattacharyya \ Coefficient = \sum_{u=1}^{m} \sqrt{p_u q_u} \tag{1}
\]

Where \( u \) is the number of histogram bins. \( p \) actually represents the histogram of target model and \( q \) represents the histogram of target candidate [1]. The value of Bhattacharyya Coefficient lies between 0 and 1. Greater the overlap between the two images, larger is the value of the Coefficient. The ASM chart is shown in Fig. 4:

3.3. Histogram

An image histogram is type of histogram which acts as a graphical representation of the tonal distribution in a digital image [2]. It plots the number of pixels for each tonal value. By looking at the histogram for a specific image a viewer is able to judge the entire tonal distribution at a glance. The horizontal axis of the graph represents the tonal variations, while the vertical axis represents the number of pixels in that particular tone. The left side of the horizontal axis represents the black and dark areas, the middle represents medium grey and the right hand side represents light and pure white areas. The vertical axis, on the other hand, represents the size of the area which is captured in each one of these zones. The ASM for histogram implemented in FPGA is given in Fig. 5

3.4. Epanechnikov Kernel

The kernel that has been used in the Image Coprocessor is Epanechnikov kernel[6]. Its profile is given by:

\[
k(x) = \begin{cases} \frac{1}{2}c_d^{-1}(d + 2)(1 - x) & \text{if } x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{2}
\]

The kernel given in (2) is used to weight the window that contains the target. After using this kernel the distance minimization and finding of target in new frame is accomplished.

3.5. Rotation

Image stabilization (IS) is a technique to increase the stability of an image [3]. It is used in image-stabilized binoculars, photography, video-graphy, and astronomical telescopes. With video cameras, camera shake causes visible frame-to-frame jitter in the recorded video. As a result distortion of the image occurs.
Distortion of the images in the video in each frame causes Loss of Data, Imprecise Tracking and Vague Vision. Distortion can be of two types, translation and rotation.

In case of rotational distortion, the image gets rotated from center by a certain angle. Each pixel gets rotated by a certain angle. Gyroscopes are used to detect the hand jitter, in terms of measuring the degree of linear and angular displacement, the image is simply shifted in software by a certain number of pixels by this degree calculated.

The purpose of this routine is to rotate the image by a certain degree of angle which is input to the system. We know that image can be distorted both linearly and rotationally. In case of rotational distortion, the image gets displaced by a certain angle from the center. This routine takes this angle as the input to the system and displaces it back to its original position. The flow of the routine is same as that of translational routine. Image is first read from the first BRAM. Position and angle of each pixel is stored. Then the image is rotated by certain angle which is required to maybe stabilize the image. However the difficulty here is that in case of rotation, the stabilized address of pixels in the second BRAM requires the calculation of COS and SIN trigonometric calculations which are complex and not available in hardware. For this reason, CORDIC math function support available in Xilinx ISE software, i.e. the software we used to implement the system has been used. As a result a successful rotated image is achieved. Fig. 7 is the block diagram and ASM of the Rotation routine.

### 3.6. Translation

In this case of distortion the frame gets translated by a certain degree. Each pixel in the image gets shifted in horizontal, vertical or both directions. This routine has been written to translate the image by a certain value of displacement. This value of displacement can be an input to the system from a gyroscope or any other device that detects degree of displacement. In this routine the image is read from a BRAM, i.e. position and value of each pixel is stored. Now each pixel is translated to a new position in the second BRAM by the value of displacement measured by an external device. This translation of the image can be horizontal, vertical or both. As currently our system has been designed for a 64x64 image, so we used a single 4k Bram for both reading the distorted image and writing the translated or stabilized image. The result is a successful translated image, which can be used to stabilize the image. ASM of the translation routine is shown in Fig. 8.
Table 1. Timing comparison table

<table>
<thead>
<tr>
<th></th>
<th>Cycles</th>
<th>Coprocessor</th>
<th>C++</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge Image</td>
<td>8195</td>
<td>418895</td>
<td>51.1 times faster</td>
<td></td>
</tr>
<tr>
<td>Bhattacharya Coefficient</td>
<td>4352</td>
<td>72231</td>
<td>16.6 times faster</td>
<td></td>
</tr>
<tr>
<td>Histogram</td>
<td>12288</td>
<td>127047</td>
<td>10.3 times faster</td>
<td></td>
</tr>
<tr>
<td>Epanechnikov Kernel</td>
<td>18432</td>
<td>1234221</td>
<td>66.9 times faster</td>
<td></td>
</tr>
<tr>
<td>Translation</td>
<td>10124</td>
<td>108412</td>
<td>10.7 times faster</td>
<td></td>
</tr>
<tr>
<td>Rotation</td>
<td>13412</td>
<td>132224</td>
<td>9.8 times faster</td>
<td></td>
</tr>
</tbody>
</table>

improved as compared to their implementation in C++. There is an average increased improvement of 36 times which helps in online, faster image tracking once it runs in parallel with the tracking algorithm. Moreover our implemented image co-processor uses only 11KB of memory which is nominal, making it applicable in memory constraint environments. Therefore we optimized time and memory resource utilization simultaneously which can increase the overall performance of any tracking algorithm.

5. Conclusion and Future Work

We have proposed a hardware architecture that will be run in parallel with the software and will speed up the computation of various functions used in the algorithms for tracking of objects in real time. The performance of our proposed architecture implemented using VHDL is way better then the performance of the implementation of these modules in High Level Languages(C++ in our case). We further plan to add more modules in our coprocessor so that it can be used by all the available object tracking algorithms making them fast and efficient.

References


