Real Time Motion Detection with FPGAs

Gradeigh D. Clark
Xianyi Gao
Elie Rosen
Eric Wengrowski

December 17, 2013

Abstract

Computer Vision offers researchers powerful tools to extract meaningful data from complex images. However, Computer Vision algorithms are often encumbered by high computational complexity, poor scalability, and the need for preprocessing. Using hardware description language, and the FPGA as a canvas to prototype our design, we have created a real-time motion detection implementation. By designing the application in hardware, we are able to exploit bit, instruction, and task-level parallelism without explicit constraint on the number of concurrent processes or threads.

Contents

1 Introduction

Conceptually, motion detection refers to measuring the relative positional change of an object to its background or measuring the change of a background to an object. Motion detection finds a plethora of applications in areas relating to security, smart homes, and lighting; a very common example being the IR or laser sensors used to turn lights on and off upon detecting motion (or the absence of it).

There are a variety of ways to implement motion detection, as listed below.

1.1 Motion Detection Types

- Infrared - These detection sensors work by measuring the infrared light emitted from an object; motion is detected if the gradient of that measured light as it moves is above some threshold.

- Optics - Optical motion detection is done by using a camera or video system and performing processing on the individual video frames or pixel data to determine whether or not motion has occurred.

- Radio - This refers to using electromagnetic waves emitted in the radio frequency band to determine if there is motion; the most common example of this would be to emit waves in this frequency range and measuriung the frequency shift between the received and transmitted waves.

- Sound - Motion can be detected with sound by setting some threshold noise level and registering if motion has occurred through analyzing subsequent sound measurements to see if there is some noise levels higher than the base line.

- Vibration - If an object is moving, it can cause displacements in the area around it. A seismometer can be used to detect the changes between one instance and the next to see if an object is moving enough; an object moving in the area around the seismometer can disrupt it, indicating motion. These see typical application in earthquakes.

To diverge into all of the different analysis methods for each of these motion detectin schema would be beyond the scope of this report. We will truncate this and cut to the chase: the ideal choice for motion detection is to use optics. Optical motion detection provides visual cues for users and has more readily verifiable results than the other methods (simply check the video to determine whether something was actually moving). Methodologies for optical motion detection can be subdivided into three broad analysis types:

1.2 Optical Detection Techniques

1. Differential Frame Analysis - This analysis type exploits temporal locality of the frame processing buffer to measure the amount of difference between sequential frames in the datapath and establish motion levels based on thresholding.

2. Background Modeling - Similarly to differential frame analysis, this also examines the magnitude of the difference between frames but it does not use locality in the same way. Background modeling requires that a frame be permanently stored in memory that is known as "the background" and examines the difference between the background and the incoming frame while employing simple modeling to update the background as the scenery changes.

3. Counting Motion Detectors - This is an extension of background modeling where the detector functionality is extended to counting blobs of the objects in the frame in order to determine what is moving and what is not by examining the position and size of the detected blobs.

We now have a grasp on detection structures to form a skeleton and a methodology to act as the skin. What is missing now is the connecting tissue – what is the mechanism by which we can link the optics to the analysis? There are two solutions:

1.3 Connecting Tissue

1. Software Solutions - These are programs such as OpenCV or MATLAB where there are built in toolboxs and libraries to accelerate the analysis of video frame data. Software solutions require that the information from the camera be written to memory prior to processing and read from memory after processing. Since the camera itself is a discrete component along with the display screen, a computer acts as the arbiter between the two.

2. Hardware Solutions - There is a way around offloading the image between the capture and sending phases to computer for software processing. Hardware could be used as an inbetween where the raw camera data is intercepted as it is sent to the display, being temporarily hijacked and processed before being sent to the display unit. Hardware solutions are broadly split into two further classes:

   - Application Specific Integrated Circuit (ASIC) - This is a hardware solution that is fabricated by a manufacturer to perform the one task that it is constructed to do – the market is dominated by these. ASICs are ubiquitous in all aspects of modern electronics; they are microprocessors, memory blocks, controllers, et cetera. A distringuishing mark of the ASIC is that is is built for one purpose only.
   - Programmable Logic Device (PLD) - Programmable Logic Devices are a collection of electronic components that are packaged together and meant to be used by the consumer to design a circuit that an ASIC would normally be used for. PLDs are composed of many arrays of digital logic components that can be programmed by the end consumer to build a logical circuit to perform a wide variety of tasks.

1.4 Project Goals

The goal of our project is to implement **real time** motion detection with a camera. Real time motion detection eliminates quite a few variables from the design; we cannot rely explicitly

on software solutions nor can we subject ourselves to analysis methods that require I/O operations. This leaves us needing to implement **differential frame analysis** using a **hardware solution**. Why?
Background modeling and, by extension, counting motion detectors will not be feasible for real time detection since they require that a background frame be stored for the long term and be continuously flagged for updates after the scenery changes. Long term storage may not turn into a problem but the constant checks for updates along with this could turn into an issue – the real time analysis constraint means that the camera cannot be allowed to be limited by RAM or bandwidth bottlenecks on the hardware solution side. Differential frame analysis stands above this through its use of temporal locality – subsequent frames that are close to each other are examined to determine the motion detection, eliminating the need for long term memory storage.
As for eliminating software solutions, this should be more apparent. Again, the real time constraint steps in to squash the software problem; software solutions require that the capture data be sent to a computer for processing before being sent back to the display unit. This inherently introduces delays:

- Write Delay
- Processing Delay
- Read Delay

which hampers the intended purpose of **real time** motion detection. This leaves us with hardware solutions. But of those solutions, as we noted earlier, there are two types to choose from. Which one is the better option?
Without a doubt, the ASIC solution is always the superior option. ASICs, by the nature of their design, are highly optimized systems; they are configured and built for the one task they are produced for. As such, they will always be faster than PLDs, less expensive for mass production, less power consuming, and less difficult to configure with a hardware description language. However, PLDs offer the distinct advantage of being reprogrammable. PLDs are the ideal building block to construct the motion detection system on in the prototype staging; an ASIC circuit would be used to replace the PLD only at the very end once it is being sent to market. A PLD can allow the user to see the modeling of motion detection as they iterate their design without waiting for a foundry to churn out the IC every so many weeks. So we intend to prove the concept of motion tracking in real time on a PLD device. The PLD we have on hand is a **Field Programmable Gate Array** or an **FPGA**.

1.5 FPGA Characteristics

The FPGA in use with this project is a **Cyclone IV EP4CE115** on an **Altera DE2-115** board. The board has the following characteristics:

Resources	EP4CE115
Logic Elements	114,480
Embedded memory	3,888
18x18 Multipliers	266
General-purpose PLLs	4
Global Clock Networks	20
User I/O Banks	8
Maximum user I/O	528

Table 1: FPGA Resources Count

This is a highly advanced board and is more than adequate for the task given the abundant resources it has to bear, as seen in Table 1.

2 Motion Detection Algorithm

2.1 Thresholding

Differential frame analysis requires that we take advantage of the subsequent nature of incoming frames and examine the magnitude of the difference between these frames. Mathematically, we can quantify the analysis thusly:

$$|T_i - T_j| \geq \delta, \quad \forall (i,j) \in \mathbb{N} * \mathbb{N}, \quad i \neq j, \quad \delta \geq 0 \tag{1}$$

Where, notationally, we have the following:

- T_i is the ith frame stored at memory location one
- T_j is the jth frame stored at memory location two
- δis the threshold parameter that the difference must be compared to in order to register whether motion has occured.

The intensity of the detected motion is highly sensitive to the thresholding parameter. Depending on whether or not the threshold is set appropriately, the output could be extremely noisy.

2.2 Grayscale Interpolation

For our purposes, it is necessary to convert to grayscale so we can use half the available memory – this is how we're able to store two frames. We reduce the information down with grayscale interpolation and then segment into separate frames. The conversion formula is as follows:

$$Gray_{x,y} = \frac{299 * Red_{x,y} + 587 * Blue_{x,y} + 114 * Green_{x,y}}{1000} \tag{2}$$

3 Datapath

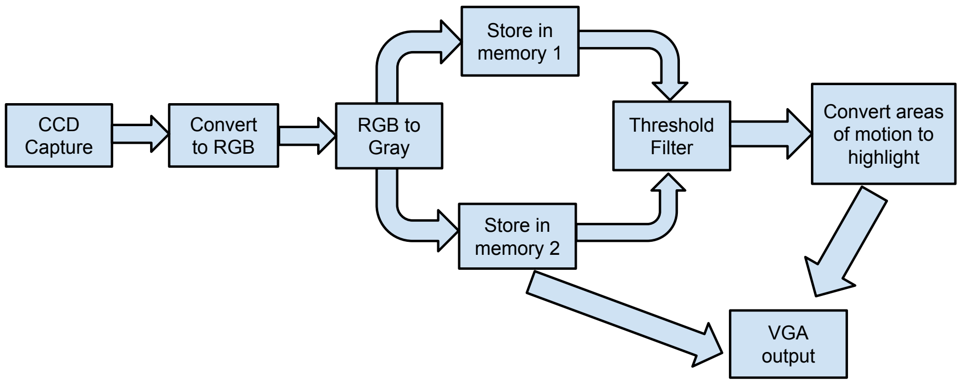


Figure 1: Project Overview

Our proposed implementation is illustrated in Figure 1. Each block performs a vital function in accomplishing the end goal of motion detection:

- CCD Capture - This block is where the raw image data from the camera is sent to the FPGA board; this is the first instance where usable data can be found.
- Convert to RGB - The raw data is taken and converted to RGB.
- RGB to Gray - We take our RGB values and weigh them such that the image is converted to a grayscale one.
- Memory Store - The image frame, depending on a counter, is stored in either an 'A' or 'B' register; these will hold at *least* the current frame and one prior to that.
- Threshold Filter- Differential frame analysis is performed on the two frames simultaneously.
- Highlight - Based on the output of the analysis, the portions of the screen where motion is detected is highlighted.
- VGA Output - The final output is sent to the VGA controller to be outputted to the screen.

3.1 CCD Capture

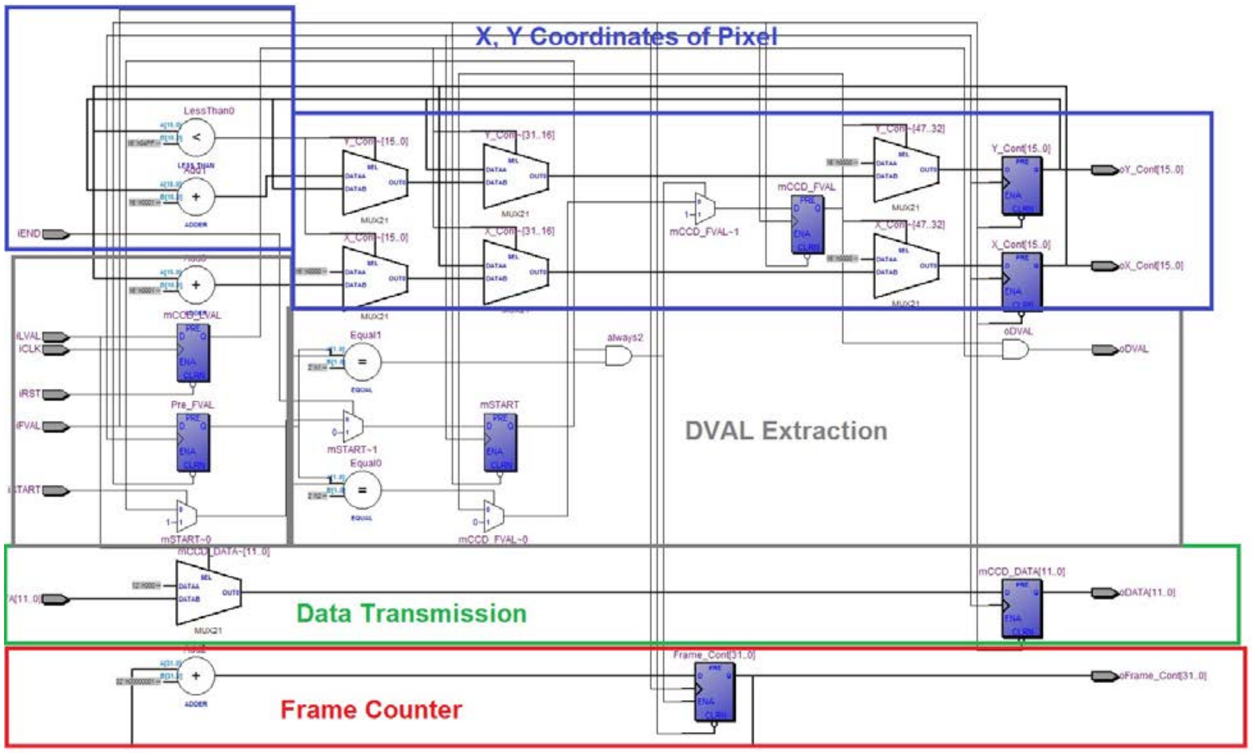


Figure 2: Logic Diagram of the CCD_Capture Block

The CCD Capture mechanism, illustrated in Figure 2, is how we receive camera data. CCD operation, at a very basic level, converts light energy into distributed charge across the cell. An analog to digital converter is what converts the pixel value to digital by reading it at each array point of the CCD and sends it to board.
The inputs are as follows:

- iCLK: Clock signal of the FPGA
- iRST: Reset signal of the FPGA
- iStart: The starting signal; signals data transfer from the CCD camera
- iEnd: Indicates the end of data transfer
- iFVAL: A verification value on the frame transfer; = 1 if valid, 0 if invalid
- iLVAL: A verification value on the line transfer; =1 if valid, 0 if invalid
- iDATA: Raw data from the camera

And the outputs are for:

- DVAL: Pixel validation value

- DATA: Raw data of the actual pixel
- X_Cont: X coordinate of the pixel
- Y_Cont: Y coordinate of the pixel
- Frame_Cont: The number of the captured frame

3.2 Convert to RGB

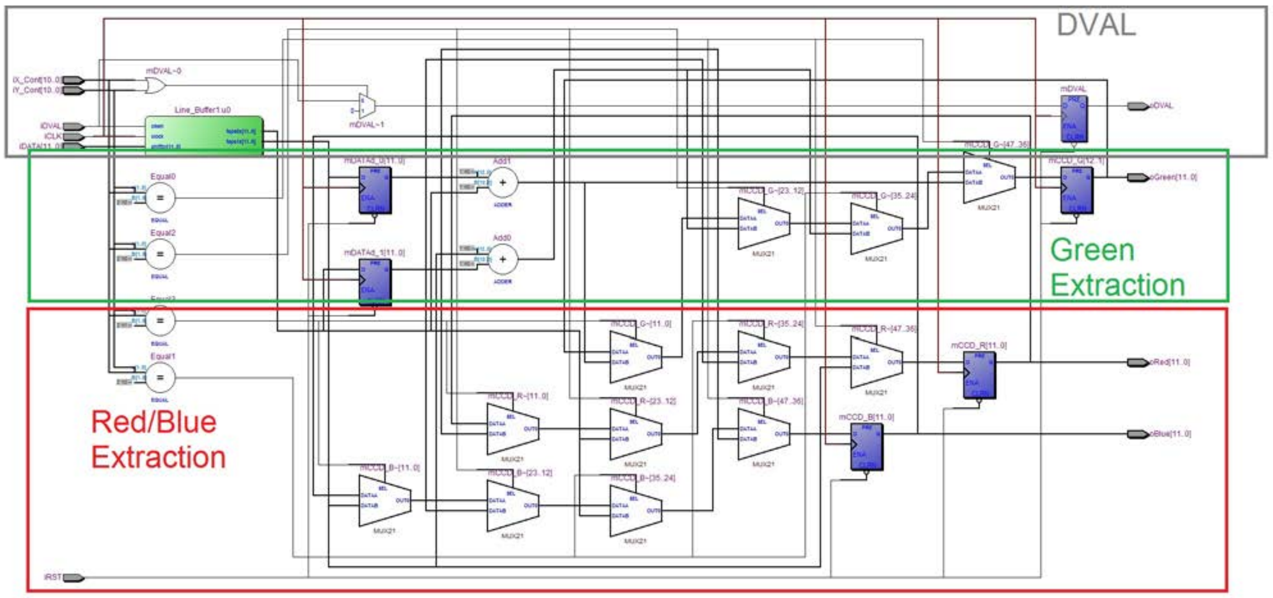


Figure 3: Logic Diagram of the RAW2RGB Block

This conversion block is meant to take the raw X,Y pixel data and convert that to RGB values. The inputs are:

- iCLK: Clock signal of the FPGA
- iRST: Reset signal of the FPGA
- DVAL: Pixel validation value
- DATA: Raw data of the actual pixel
- X_Cont: X coordinate of the pixel
- Y_Cont: Y coordinate of the pixel

And the outputs are:

- DVAL: Pixel validation value

- iRed: Red pixel value
- iBlue: Blue pixel value
- iGreen: Green pixel value

3.3 RGB to Gray Conversion

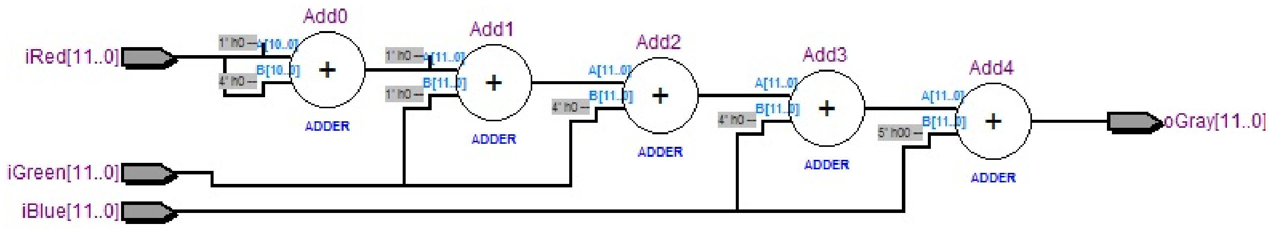


Figure 4: Logic Diagram of the RGB2Gray Block

This block performs the RGB to grayscale conversion. The inputs are:

- iRed: Red pixel value
- iBlue: Blue pixel value
- iGreen: Green pixel value

And the outputs are:

- Gray - The converted grayscale pixel value

3.4 Memory Switching

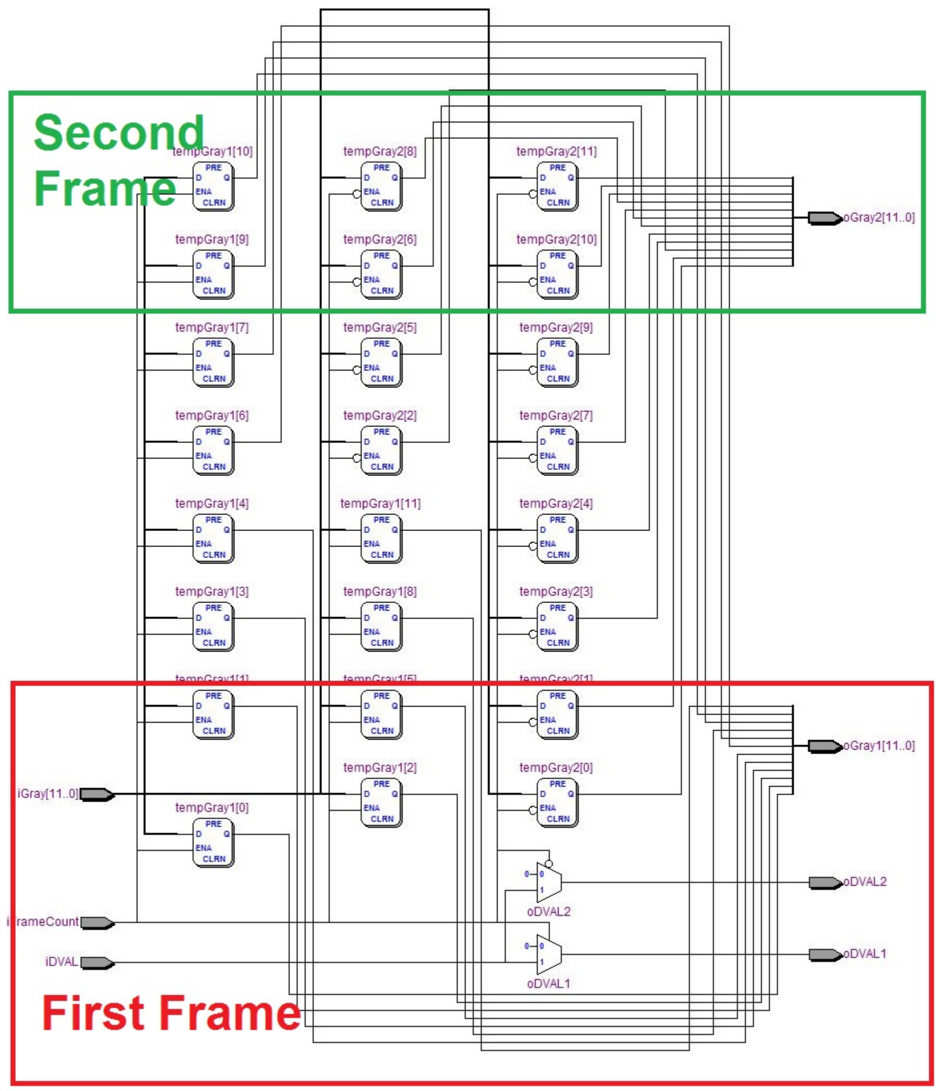


Figure 5: Logic Diagram of the MemSwitch Block

Pictured is the memory switching block. To perform differential frame analysis, we need access to at least two frames. This memory structure represents our ability to do that; a frame is loaded into the first memory location and then the frame counter is incremented to load the incoming frame at the second memory location. The inputs are:

- iGray - The current grayscale pixel value
- FrameCount - A counter that decides which memory location to store a frame at
- DVAL - Validation on the pixel

And the outputs are:

- iGray1 - The current grayscale pixel value at location one
- iGray2 - The current grayscale pixel value at location tweo
- DVAL1 - Validation on the first pixel
- DVAL2 - Validation on the second pixel

3.5 VGA Controller

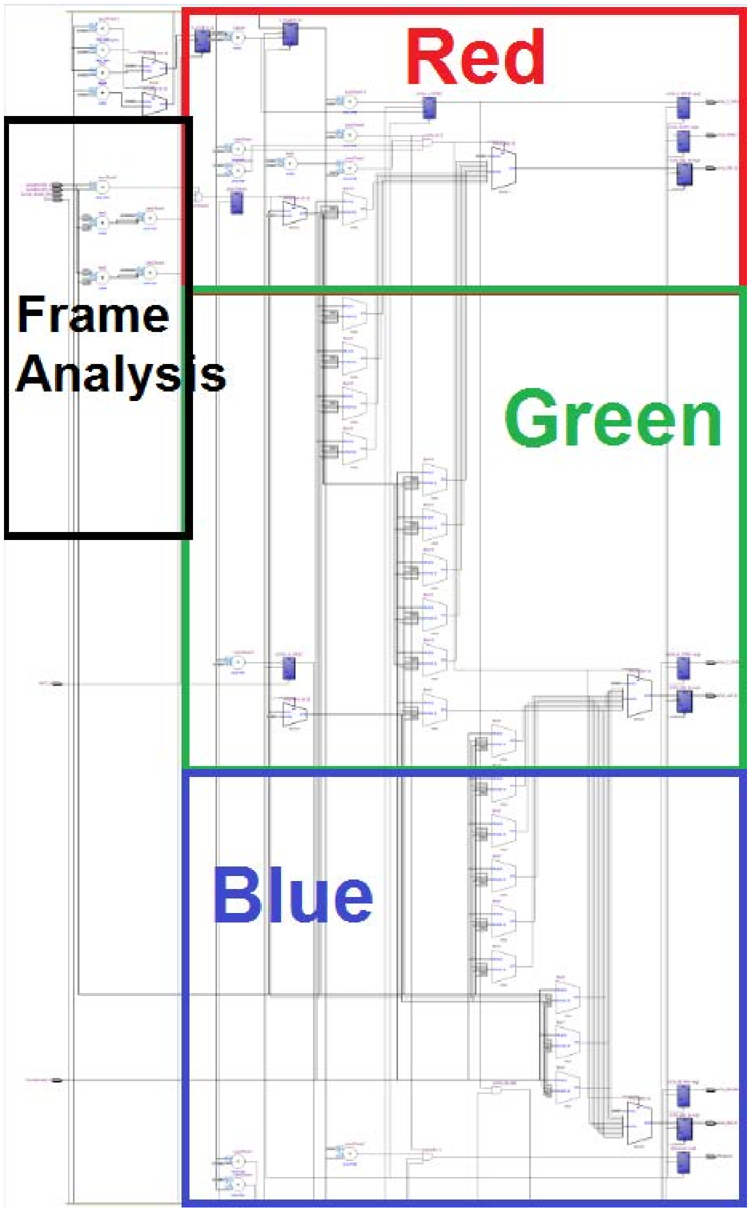


Figure 6: Logic Diagram of the VGA Controller Block

The VGA controller is a standard package that comes with the FPGA board. The Red, Blue, and Green VGA outputs are highlighted in the appropriate boxes and our frame analysis portion is highlighted in the black box. Our frame analysis is done entirely by making modifications to the standard controller; the two frames are simultaneously inputted into the VGA controller and analyzed via thresholding. The detection output is then sent to the VGA controller instead of normal camera motion. The important inputs are:

- iGray1 - The current grayscale pixel value at location one
- iGray2 - The current grayscale pixel value at location twe
- iTrackMode - Control bits for the different operating modes of motion detection

And the important outputs:

- VGA_R - R value of outputted pixel to the screen
- VGA_G - G value of outputted pixel to the screen
- VGA_B - B value of outputted pixel to the screen

4 Functionality

To showcase all the abilities of the project, the following items can be preformed on the FPGA to demonstrate results:

- VGA Output
  - The most current processed image is converted to signals that control the VGA protocal. On screen images display the current opperation of the algorithim.
- Genral Purpose Input / Output (GPIO)
  - The 5 Mega Pixel camera is connected through the 40 pin GPIO port which contains signals for interfacing. Some of these signals include I2C communication for sending pixel data and a clock for syncing frames
- Switches
  - Switches 0 through 2 control the current opperating mode.
    - SW0 enables the display of motion data in red
    - SW1 enables the display of motion data overlayed with the current frame
    - SW2 enables the display of motion data in cyan.
    - If none of these switches are enabled, the current frame is displayed
  - Switches 4 through 7 control the thresholding amount
    - SW4 turns off thresholding
    - SW5 sets thresholding to 4 bits of noise (16)
    - SW6 sets thresholding to 5 bits of noise (32)
    - SW7 sets thresholding to 6 bits of noise (64)

        * If none of these switches are enabled, thresholding is set to 50
    - Switch 16 enables a built in zoom function
    - Switch 17 changes exposure increase / decrease

- LEDs
    - Green LEDs 0 through 5 display the amount of activity between frames, more LEDs being turned on corresponds to increased activity in the frame
    - Green LED 8 displays when the next frame is available
- Hex Displays
    - The hex displays show a counter for the number of frames captured since the device being powered on
- Keys
    - KEY0 resets the device
    - KEY1 in conjuction with the setting of SW17 increases / decreases frame exposure time

5 Results

In this section, we will summarize the results of our motion detection implementation on the FPGA.

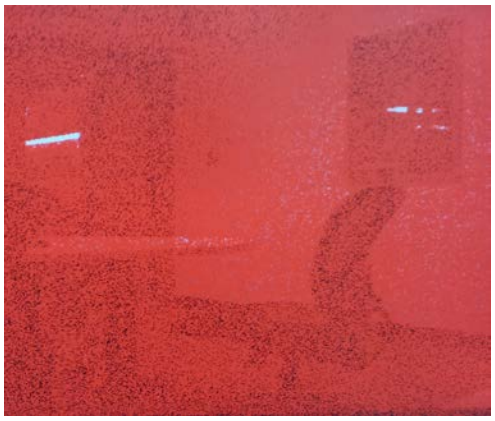
5.1 Resources

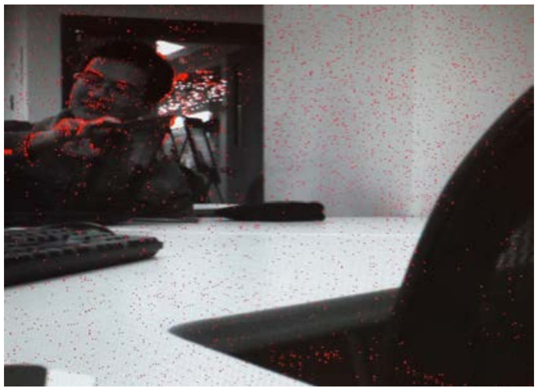
Resources	Total	Used
Logic Elements	114,480	1,914 (2%)
Embedded memory	3,888	1,608 (1%)
General-purpose PLLs	4	1 (25%)
Maximum user I/O	528	428 (81%)
Combinational functions	114,480	1,608 (1%)

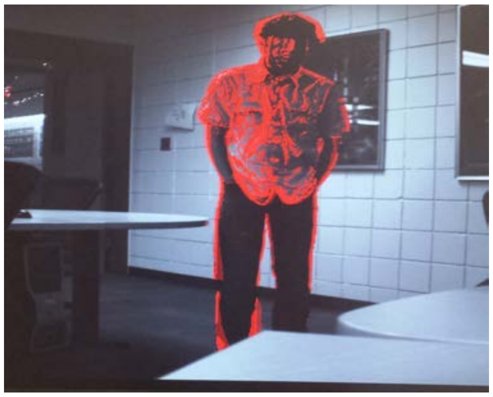
Table 2: FPGA Resources Results

Table 2 summarizes the effect our implementation has on the FPGA device itself. In general, the solution is of low cost to the the FPGA; we use a very small number of logical elements (2%), combinational functions (1%), and embedded memory (1%). The reason why the maximum user I/O is so high is because this is not an ASIC solution; an ASIC solution would only have the number of pins required for the camera. Because the FPGA is mounted on the Altera board, the user pins need to be used up by it for other functionality.

5.2 Thresholding


(a) Threshold = 0


(b) Threshold = 16


(c) Threshold = 32

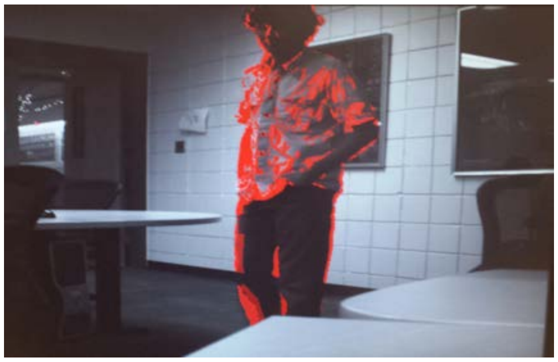

(d) Threshold = 64

Figure 7

The results of our thresholding analysis can be seen in the above figure. As we can see, the differential frame analysis is very sensistive to the value of the threshold parameter as described in the Motion Detection Algorithm section. With no thresholding parameter and accepting all motion, the output is horribly noisy. As the threshold is increased, the noise is reduced and the red filling that describes the detected motion is reduced and sharpened.
There is an inherent tradeoff between increasing the threshold and losing valuable motion data. There are merits to some of the "noisier" threshold values, depending on the application. For instance, if you are more interested in outputting a more continuous region of detected motion than eliminating outliers, then a lower threshold isn't a bad idea. Especially because techniques such as RANSAC can be used in postprocessing if the application allowed it. This further justifies our design to allow user switching of the intensity threshold. Through heuristics, it was determined that a threshold value of 50 provided the best balance between data loss and noise stabilization.