Real Time Hot Spot Detection Using FPGA

Sol Pedre, Andres Stoliar, and Patricia Borensztejn

Departamento de Computación, Facultad de Ciencias
Exactas y Naturales, Universidad de Buenos Aires
{spedre,astoliar,patricia}@dc.uba.ar

Abstract. Many remote sensing applications require on-board, real time processing with low power consumption. Solutions based in FPGA implementations are common in these cases to optimize the processing resources needed. In this paper we describe an FPGA based solution for a remote sensing application that processes real time video from an infrared camera in order to identify hot spots. The solution reduces the information in each frame to the location and spatial configuration of each hot spot present in the frame. The proposed method successfully segments the image with a total processing delay equal to the acquisition time of one pixel (that is, at the video rate). This processing delay is independent of the image size. The solution is not tied up to one specific camera, and may be used with several infrared cameras with minor adjustments. FPGA area equations are also presented in order to calculate the needed FPGA size for a particular application.

Keywords: real time image processing, FPGA, remote sensing, hot spot detection, embedded computing.

1 Introduction

Many remote sensing applications require on board, real time processing with low power consumption. For many of these embedded digital signal processing applications, today´s general purpose microprocessors are not longer able to handle them [1]. Functional Programming Gate Array (FPGA) offer a highly parallel hardware architecture with low power consumption that is an alternative for such digital signal processing implementations.

Field Programmable Gate Arrays, or FPGA, are devices made up of thousands of logic cells and memory. Logic cells, memory and interconnections between them are software programmable using a standard computer. Therefore, these devices offer a fast and cheap prototype solution for an embedded product, and when the production scale is small, they also offer a fine final solution.

In this paper we present and describe a real time image processing algorithm, hot spot detection, implemented on an FPGA device. This solution was developed for an Unmanned Aerial Vehicle (UAV) System of the Department of Computer Architecture, Escola Politècnica Superior de Castelldefels, Universitat Politècnica de Catalunya. The aim of the algorithm is to identify fire embers (that is, hot spots) in the images captured by an infrared video camera on the UAV. The location and characteristics of

E. Bayro-Corrochano and J.-O. Eklundh (Eds.): CIARP 2009, LNCS 5856, pp. 595–602, 2009.

the detected hot spots are then transmitted by the UAV to a firemen team fighting a forest fire [2]. Our solution is general enough to be integrated in other systems, and with different cameras. The rest of this paper is organized as follows: section 2 explains the hot spot detection algorithm, section 3 explains the FPGA implementation, section 4 shows the experiments and results and section 5 explains some conclusions.

2 Hot Spot Detection

The problem consists on processing video as it is captured by an IR camera on an Unmanned Aerial Vehicle. The UAV has a network centric architecture, in which all sensors and different processing units are connected to an Ethernet network [2]. Our proposed FPGA solution is inserted between the IR camera and the network. It takes the analogical output of the IR camera, processes the frames in real time and returns the location and spatial configuration of the found hot spots (if any) in UDP packets. As many infrared cameras have analogical outputs, as composed video, the proposed solution can be used for different IR cameras and is not tied up to one specific camera.

The most important constraint for the solution is that the image needs to be processed in real time, with the minimum possible delay between the acquisition of the last pixel and the transmission of the results. It is also desirable that the whole application works at the slowest possible clock frequency, to minimize the FPGA's power consumption.

In order to fulfill these requirements, the proposed algorithm and hardware implementation exploit the intrinsic parallelism of the process, obtaining the results of a complete frame at the moment that its acquisition is finished, with a total processing delay equal to the acquisition time of one pixel (i.e, the camera´s pixel video frequency). Moreover, as the camera delivers continuous images, the results of the previous frame are transmitted in parallel with the processing of the current frame. Finally, the application runs with the smallest possible clock frequency that allows to fulfill the previous requirements: the IR camera´s pixel clock frequency. This also simplifies the integration between the camera and the proposed solution.

2.1 Segmentation Algorithm

The proposed algorithm segments the image in hot and cold regions, storing the location and spatial configuration of the found hot regions (i.e, hot spots). Its complexity lies in grouping the pixels in hot spots and updating the stored hot spot's data as the image is being captured. The IR camera's output video is first digitalized, the temperature pixel is extracted and then classified as a hot o cold pixel (i.e, if the pixel belongs to a hot spot or not). The segmentation algorithm then checks if the adjacent pixels belong to hot spots and decide if the current pixel is the beginning of a new hot spot, if it belongs to an already existing hot spot, and whether this pixel unifies two previously discovered hot spots. It also updates the stored hot spot´s data accordingly.

In this manner, the algorithm performs the segmentation of the image using only the current pixel and a list L that stores to which hot spot (if any) the previous line of pixels belong to. Therefore, there is no need for extra memory to store parts or the complete image, and the total processing delay is independent on the image size. The algorithm´s pseudo code is shown in *Listing1*.

Listing 1. Segmentation Algorithm

```
inputs:
- pixel(m,n)
- line L of previous pixels indicating which hot spot the belong to, if any.

receive pixel(m,n)
    if pixel(m,n) does not belong to a hot spot
            mark in the line L that pixel(m,n) does not belong to a hot spot.
    if pixel(m,n) belongs to a hot spot
        if pixel(m-1,n) and pixel(m,n-1) do not belong to hot spots
                create a new hot spot for pixel(m,n)
                mark pixel(i,j) in the line L as belonging to the new hot spot
        if (pixel(m-1,n) belongs to hot_spot_x and pixel(m,n-1) does not belong to any hot spot)
        or (pixel(m-1,n) does not belong to any hot spot and pixel(m,n-1) belongs to hot_spot_x)
        or (pixel(m-1,n) and pixel(m,n-1) belong to hot_spot_x))
                add pixel(m,n) to hot_spot_x in the memory
                mark pixel(m,n) in line L as belonging to hot_spot_x
        if (pixel(m-1,n) belongs to hot_spot_x and pixel(m, n-1) belongs to hot_spot_y
        and id(hot_spot_x) < id(hot_spot_y))
                add hot_spot_y data to hot_spot_x in the memory
                add pixel(m,n) to hot_spot_x in the memory
                mark hot_spot_y as invalid in the memory
                mark pixel(m,n) in line L as belonging to hot_spot_x
                for each pixel in line L
                        if (pixel belongs to hot_spot_y)
                                mark pixel as belonging to hot_spot_x
```

3 FPGA Implementation

We propose the use of FPGA technology to achieve the real time processing of the image, with a total processing delay equal to the acquisition time of one pixel (i.e video frequency). Some hardware architectures have been presented in literature aimed at accelerating image processing methods [3][4][5][6] but none intended for the segmentation of an IR image for hot spot detection.

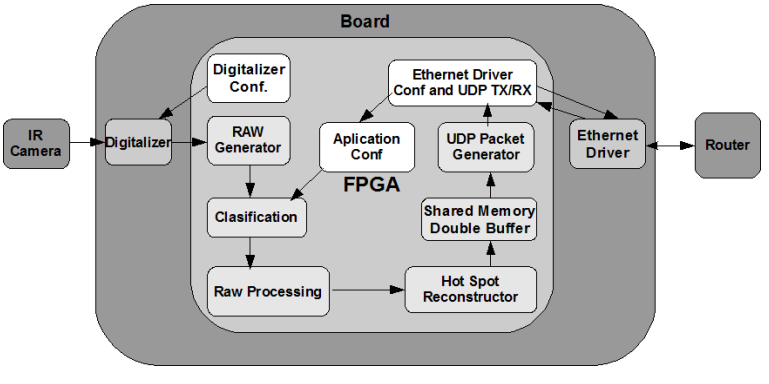

Fig. 1. Complete board including the video digitalizer, the FPGA and Ethernet physical driver

In this section we describe the FPGA implementation, focusing in the Raw Processing, Hot Spot Reconstructor and Double Buffer Shared Memory modules that are the core of the solution. These modules implement the proposed algorithm (*Listing 1*) in a way that the real time constraint is achieved. Fig. 1 shows the complete hardware solution.

3.1 Raw Processing, Hot Spot Reconstructor and Memory Modules

The Raw Processing module determines if the current pixel is the beginning of a new hot spot, if it belongs to an already existing hot spot, and whether this pixel unifies two previously discovered hot spots. To do this. it keeps the line *L* of previous pixels showing to which hot spot they belong to, and update this line as shown in *Listing1*.

The core of the Raw Processing module is the implementation of *L* such as to calculate which hot spot the current pixel belongs to and update all the list in only one pixel clock. For this purpose, *L* is implemented as a stack: the top of the stack stores the *id* of the hot spot that *pixel(m,n-1)* belongs to, and the bottom of the stack stores the *id* of the hot spot *pixel(m-1,n)* belongs to. This two special records can be accessed to obtain the hot spot *ids* needed in the algorithm. The remaining records in *L* hold the information of the line of pixels between *pixel(m-1,n)* and *pixel(m,n-1)*, i.e, the previous line of pixels. In each clock, the hot spot *id* corresponding to the new pixel is pushed onto the stack, all the middle records are updated if necessary and moved to the next stack position, and the bottom record (i.e, the hot spot *id* of *pixel(m-1,n)*) is discarded.

When a pixel unifies two previously discovered hot spots, say *hot_spot_y* and *hot_spot_x*, *hot_spot_y* is marked as invalid and all the pixels belonging to that hot spot are added to *hot_spot_x*. In that case, all records in the stack have to be accessed, compared with the *id* of *hot_spot_y* and changed to the *id* of *hot_spot_x* (if needed) in one clock cycle. To accomplish this, each record of the stack has a comparator and a multiplexer. The result of comparing the record's *id*, say *idA*, with the *id* of *hot_spot_y* enables the multiplexer that either propagates *idA* or the *id* of *hot_spot_x* to the next record as needed. The implementation of one record of the list *L* in this module is shown in Fig. 2.

The Hot Spot Reconstructor module is in charge of updating the information of hot spots found to the moment with the information from the current pixel, as shown in the algorithm in *Listing 1*. The Raw Processing module tells the Hot Spot Reconstructor module whether it has to create a new hot spot with the current pixel, unify two hot spots or simply add the pixel to an existing hot spot. This module has to access the hot spot Memory, retrieve the corresponding information, recalculate the data and write the results back, all in one pixel clock. The implementation of this module is shown in Fig. 3

The Memory module is designed as a shared double buffer. Each buffer is organized as a vector of records, with one record for each hot spot. Each record stores the location and spatial configuration of the hot spot. The partial results of the current frame are stored in one buffer, while the final results of the previous frame are stored in the other one. In this manner, the results of the previous frame can be transmitted in parallel with the segmentation of the current frame.

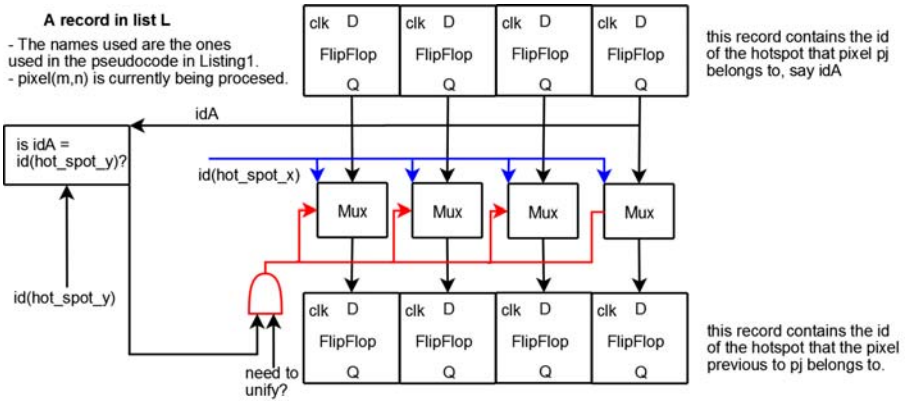

Fig. 2. Implementation of a record of list *L* in the Raw Processing module

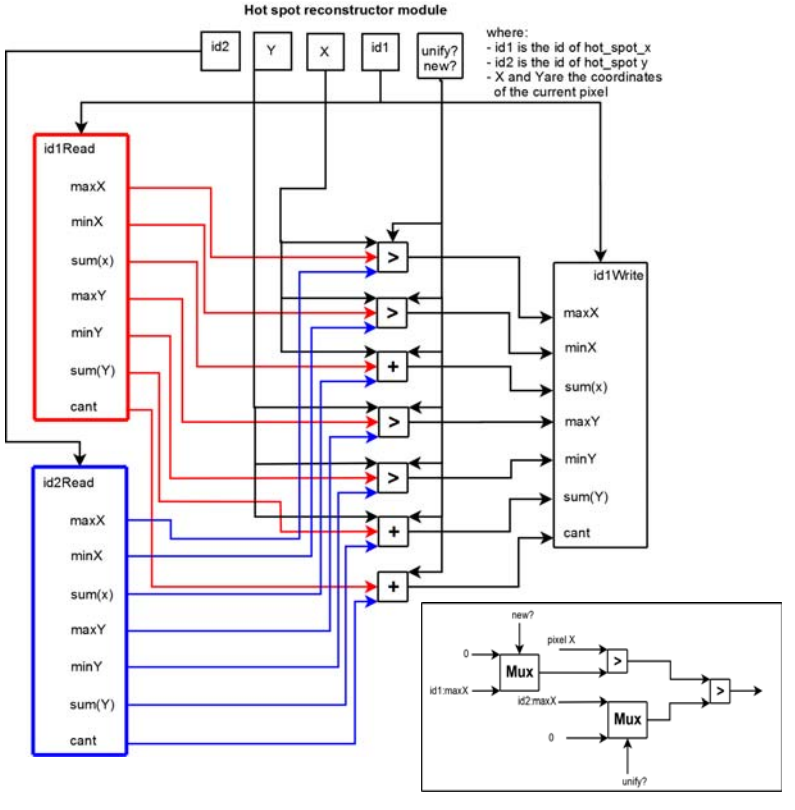

Fig. 3. Hot Spot Reconstructor module and detail of the logic for the maximum calculation

3.2 Auxiliary Modules

The Raw Generator module generates the RAW stream by extracting the temperature pixels from the digitalized video. The Classification module classifies each pixel as belonging to a hot spot or not, depending on its IR radiation determined by the pixel´s value.

The UDP Packet Generator is the module in charge of creating the correct UDP packets with the hot spot data of the previous frame. As this module works with the clock frequency of the MAC Ethernet Module and the rest of the application works with a different clock frequency (i.e, the IR camera´s pixel clock frequency), we use a FIFO to solve the associated problems with the exchange of information between two different clock domains.

Finally, the configuration modules allows to configure the integrated circuits on the hardware board outside the FPGA: the SAA7113 digitalizer and the Ethernet physical driver. There is also an Application Configuration module that allows to configure through the Ethernet connection variables such as the threshold for the classification module or the ip address and port of the UAV's CPU.

3.3 Solution Sizing

In order to implement the high parallelism needed to achieve the proposed real time processing of the image, much space and hardware resources of the FPGA are used. The area needed for this implementation depends only on the size of the image and the maximum amount of hot spots that can be found in each image. In order to make the application suitable for different IR cameras, those parameters can be easily configured.

There are two modules in the implementation that are resource consuming: the Raw Processing module and the Memory module. The Raw Processing module implements the list *L* that stores the hot spot *id* for each pixel in the previous line, as explained in section 3.1. In terms of FPGA area, the list has *im_width* records. Each record is wide enough to store a hot spot *id*, that is *log(max_hotspot_amount)* bits, and has extra logic needed for the hot spot unifying process. The area equations is as follows:

$$im\_width * [\log(max\_hotspot\_amount) * (1 \text{ flip flop} + 1 \text{ mux}) + 1 \text{ comparator of } \log(max\_hotspot\_amount) \text{ bits}]$$

The Memory module stores the information of each hot spot found in the image, that is, the memory has *max_hotspot_amount* records. Each record stores the information needed to calculate the location and spatial configuration of one hot spot, that is: *{maxX, minX, sumX, maxY, minY, sumY, count of pixels}*. Finally, there are two memory buffers, one with the final results of the previous frame and one with the partial results of the current frame. The hot spot memory is implemented using the FPGA block rams. The needed block ram equation is as follows:

$$2 * max\_hotspot\_amount * [\log(im\_width) + \log(im\_width) + (2 * \log(im\_width) - 1) + \log(im\_height) + \log(im\_height) + (2 * \log(im\_height) - 1) + \log(im\_width * im\_height) - 1]$$

From these equations, we can see that the amount of logic cells needed depend linearly on the image width, while the amount or memory (block rams) depends linearly on the maximum amount of hot spots to be detected per frame. The image height is of little importance for area calculations. With these equations, it is straightforward to calculate the size of the needed FPGA given the size of the image and a maximum for the amount of hot spots expected in each frame.

4 Experiments and Results

Our testing environment consists of a PAL-N composed video camera, a development kit with a Xilinx Virtex 4 FX12-10C-ES FPGA, a SAA7113 video digitalizer and an Ethernet physical driver. The development environment used is the Xilinx ISE Webpack 10.1, with default settings for all the involved processes. The UDP packets with the resulting hot spots are routed to a PC in order to check the processing results. From the video camera we process and analyze 50 frames per second, corresponding to half video images even and odd, with 512 pixels by 256 lines per frame. The pixel clock generated by the SAA7113 digitalizer is the standard 27 Mhz clock for video coding ITU-R BT 656 YUV 4:2:2. The solution was configured for a 256 maximum amount of hot spots per frame. In Fig. 4 some results are shown.

As all the memories were mapped into block rams, including the UDP FIFO and the configuration memory for the SAA7113, and the hardware MAC Ethernet included in the Virtex 4 was used, the total area of the application was 84% of the FPGA slices (1 LUT + 1 flip flop) and 32% of the block rams.

In particular, the occupied area of the main modules (Raw Processing, Hot Spot Reconstructor and Memory) was 79% of the FPGA slices and 25% of the block-rams. This area corresponds to the size equations presented in section 4, and it means that the entire application fits in the smallest Virtex 4 FPGA available in the market. The maximum operation clock obtained was of little over 100 Mhz, which is enough to work with the 27 Mhz clock output of the SAA7113. In the UAV application, the mounted IR camera is the FLIR A320, that delivers 320 pixels by 240 lines images at a rate of 9 fps. Therefore, the results of the tests in the laboratory experiments indicate that the solution is well suited for the UAV application.

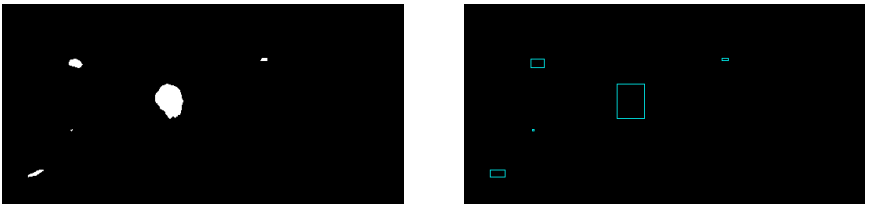

Fig. 4. *left:* hot spot image after classification in hot or cold pixels. *right:* visual representation of the results showing the location of the detected hot spot (the center of mass is not shown).

5 Conclusions

In this paper we proposed the use of FPGA technology to achieve real time processing of an IR image for hot spot detection. The proposed method successfully segments the image with a total processing delay equal to the acquisition time of one pixel (that is, at the video rate). This processing delay time is independent of the image size. There is also no need for extra memory to store parts or the complete image. The proposed solution is not tied to one specific IR camera, and may be used with several IR camera with minor adjustments. FPGA area equations were presented in order to calculate the needed FPGA size for a particular application.

The experiments show that the maximum operation clock is of little over 100 Mhz, which is enough to work with the 27 Mhz clock output of the SAA7113. Moreover, the entire application fits into the smallest Virtex 4 FPGA available in the market. The results also show that the proposed method is well suited to work with the FLIR A320 camera on the UAV application.

References

1. Dally, W., Balfour, J., Black-Shaffer, D., Chen, J., Chen, H.C., Parikh, V., Park, J., Sheffield, D.: Efficient Embedded Computing. Computer, 27–32 (2008), IEEE 0018-9162
2. Salami, E., Pedre, S., Borenzstejn, P., Barrado, C., Stoliar, A., Pastor, E.: Decision Support System for Hot Spot Detection. In: 5th International Conference on Intelligent Environments (IE 2009), Universitat Politecnica de Catalunya, Barcelona, Spain (2009)
3. Draper, B.A., Beveridge, R., Willem, B.A.P., Ross, C., Chawathe, M.: Accelerated image processing on FPGAs. IEEE Transactions on Image Processing 12(12), 1543–1551 (2003)
4. Torres, C., Arias, M.: FPGA-based Configurable Systolic Architecture for Windows-based Image Processing. EURASIP Journal on Applied Signal Processing, Special Issue on Machine Perception in a Chip 2005(7), 1024–1034 (2005)
5. Chang, C., Hsiao, P., Huang, Z.: Integrated Operation of Image Capturing and Processing in FPGA. IJCSNS International Journal of Computer Science and Network Security 6(1), 173–180 (2006)
6. Chang, L., Rodés, I., Méndez, H., del Toro, E.: Best-Shot Selection for Video Face Recognition Using FPGA. In: Ruiz-Shulcloper, J., Kropatsch, W.G. (eds.) CIARP 2008. LNCS, vol. 5197, pp. 543–550. Springer, Heidelberg (2008)