Low Latency real-time Deflagration Detection using a Field Programmable Gate Array

Marcel Eckert, Jakob Krooß, Bernd Klauer, Alexander Fay
Helmut-Schmidt-University/University of the Federal Armed Forces Hamburg, Hamburg, Germany

Felix Kümmerlen
Bundeswehr Research Institute for Protective Technologies and NBC Protection (WIS) Munster, Germany

Abstract

Deflagrations pose a serious threat in various scenarios. To reduce resulting damage to man and material, it needs to be detected and extinguished in its initial phase. An algorithm for soft real-time detection of deflagrations based on image processing and high-speed cameras has been presented in previous work. In this contribution, the latest iteration of the algorithm is transferred to a field programmable gate array, resulting in a hard-real-time system that reduces computation times to negligible 80 ns and enables improvements of the algorithm that are to slow on CPU- or GPU-based systems.

Keywords: deflagration detection, FPGA, image processing

Introduction

Deflagrations may occur in various scenarios when a combustible material (sprayed fuel, dust, flour, etc.) mixes with oxygen. If the mixture ignites, the resulting rapid combustion results in a significant increase of pressure and temperature that poses a serious threat to surrounding material and/or people.

Deflagration protection is an important topic in various industrial scenarios since many educts and products in chemical and food industry are flammable. In military vehicles deflagrations may occur as a secondary effect of enemy fire. Most military vehicles exhibit specialized fire suppression units based on chemical extinguishing gases and infrared detectors. To minimize the impact of a deflagration, it is necessary to detect and suppress it in its initial phase [1].

However, with the ban of the chemical extinguishing gas Halon due to its negative effect on the environment, alternative agents like water mist must be considered [2]. Water mist has a high efficiency and does not have a negative impact on environment and health, but its extinguishing effect can only be fully applied locally. Thus, additional information about a deflagration, especially its location, is needed.

Related work and structure of this paper

One-dimensional sensors, e.g. infrared sensors, cannot provide information about the location of a starting deflagration. Hence, in the recent years, the Institute of Automation Technology developed a novel algorithm for the detection, localization and volume estimation of deflagrations with a high-speed multi camera system [1,3,4]. In contrast to most video fire detection algorithms (an overview can be found e.g. in [5]), it is designed specifically to meet the very high requirements on delivering results in a very short time and, therefore, computational efficiency. In [4], a GPU based implementation of the algorithm has been introduced. The computation time for one image has been measured to be approx. 3.4 ms on a high-end PC.

However, this is not fast enough. The computation time does not only delay the detection itself, but also allows the deflagration to develop further, rendering the information about position and size outdated and thus unreliable. A significantly faster computation is needed. Another deficiency of the GPU implementation is the missing ability for hard real-time. This may cause delayed detection and/or data backlog in certain situations.

Both deficiencies can be dealt with by transferring the algorithm to a field programmable gate array (FPGA). On these, in contrast to CPUs/GPUs, the algorithm's processes are not divided into the basic operations which the CPU/GPU provides; instead, even complex operations can be built based on the generic, programmable logic circuits on the FPGA. Also, the processes do no longer need to be computed sequentially but can be computed in parallel.

These two differences result, especially in the field of image processing, in an often greatly improved processing speed. In fact, often, the processing is fast enough to apply it directly on the incoming pixel stream. Depending on the specific algorithm, this allows to provide the results only a few clock cycles after the last pixels of an image have arrived. Also, FPGAs form a perfect hard real-time system; the processing time can be predicted to a certain clock count.

However, the transfer of a CPU-based algorithm to an FPGA architecture is not trivial and often linked to a higher effort compared to traditional implementations.

This paper starts with a short recap of the algorithm itself and then presents our novel and experimental FPGA-based implementation and its benefits. Its focus is on the calculation of the segmentations. The algorithms for advanced detection, localization and volume estimation are, in comparison to the calculation of the segmentations, fast to compute on any hardware system. They are thus out of the scope of the first iteration of the FPGA implementation and this paper and can instead be found in [3].

Algorithm

The algorithm consists of two phases. In a first phase, each pixel is classified individually, resulting in a segmentation of the deflagration. In a second phase, the whole image is classified, amongst others using the amount and its increase of segmented pixels. Also, localization and volume estimation take place in the second phase, using information like number and moment of the segmented pixels.

The segmentation in the first step is based on four sub-steps:

1) The tupel of intensity as well as its increase in comparison to the last image is evaluated (deflagrations are bright and become brighter). The exact decision making is done by a heuristic, two-dimensional look-up-table.

2) The color of each pixel is evaluated in RGB-space by the ruleset

$$r > b \cap r > g \cap r > \bar{r}, \quad \text{(Eq. 1)}$$

where r, g and b are the red, green and blue color channels and $\bar{x}$ is the mean of the corresponding color channel x over the whole image.

3) The color of each pixel is evaluated in YCbCr-space by the ruleset

$$\bar{Y} > 17 \cap Y > \bar{Y} \cap Cb < \overline{Cb} \cap Cr > \overline{Cr}, \quad \text{(Eq. 2)}$$

where Y, Cb and Cr are the three components of the YCbCr-color-space that have to be calculated from the RGB-values by multiplication with a constant 3x3-matrix.

4) To exclude non-changing pixels from segmentation, a background-model based on an exponential moving average is applied. The background B is initialized with the current red channel r. Each following background is then calculated by

$$B_i = 0.4\, B_{i-1} + 0.6\, r_i \quad \text{(Eq. 3)}$$

and each pixel classified by

$$|r_i - B_i| > \Theta \; with \; \Theta = \begin{cases} 12 & r_i > 130 \\ 28 & r_i > 85. \\ 56 & else \end{cases} \quad \text{(Eq. 4)}$$

Only if all for sub-steps result in a positive classification, a pixel is set in the segmentation map^{det}.

The resulting segmentation has a high sensitivity, which is highly beneficial for detection. However, it does not accurately segment the deflagration but also areas like illuminated dust / droplets surrounding the deflagration or reflections, which significantly decreases the quality of position and size estimates. Instead, for the latter, a lower sensitivity and thus a more precise segmentation is desirable. Finding a compromise regarding sensitivity is difficult and will reduce capability of both detection and localization.

These contradicting requirements are dealt with by determining another, less sensitive and more precise segmentation map^{loc} for localization etc. by applying an additional constraint on the detection segmentation: The red channel needs to reach its maximum. Also, this second segmentation step "remembers" pixels segmented a few images earlier to prevent non-segmented pixels in the center of the deflagration (the intensity values don't increase any longer since they reached the maximum intensity already; the pixels are not segmented in the current image despite belonging to the deflagration):

$$map_i^{loc} = \left(map_i^{det} \cup map_{i-1}^{det} \cup \ldots \cup map_{i-n}^{det}\right) \cap r = 255, \qquad \text{(Eq. 5)}$$

where i describes the current image and $i-1$ the last image etc.

I.e., pixels stay in the segmentation until r drops below the maximum value or the last occurrence in a detection segmentation is more than n images ago. In a CPU- or GPU-based setup, n is limited to a very low value since every additional segmentation considered adds to the overall computation time. There are work arounds for this [3], however, these prove instable in certain situations.

VHDL model and demonstrator implementation

Our FPGA design has been written in VHDL, a hardware description language for simulation and synthesis. The overall architecture is shown in Fig. 1. The idea is to insert an FPGA into the transmission line (the cable) between a camera and a computer which further processes the results of the FPGA. This setup allows to precisely predict the latency of the calculations. Initially, the five parts of the described algorithm (four sub-steps for the first segmentation step, one for the second segmentation step) were ported into a simulation model. This simulation model can be used to verify the functional correctness of the VHDL-implementation in comparison to the software implementations.

The simulation model was then used as a reference to verify a synthesizable VHDL-model for each of the five parts of the segmentation and the final overall implementation. In Fig. 2, a processing step of an example video used in simulation is shown.

The VHDL-implementations allow to calculate each part of the segmentation algorithm in exactly one clock cycle – the pixel-clock of the transmission line is used as clock. Hence, the entire two step segmentation algorithm is calculated with a delay of two pixel-clock cycles. This is achieved by exploiting the massive parallelization possibilities of an FPGA. To allow for an easy adaption to different resolutions (e.g. for different cameras) in the future, the VHDL models are written in a generic way.

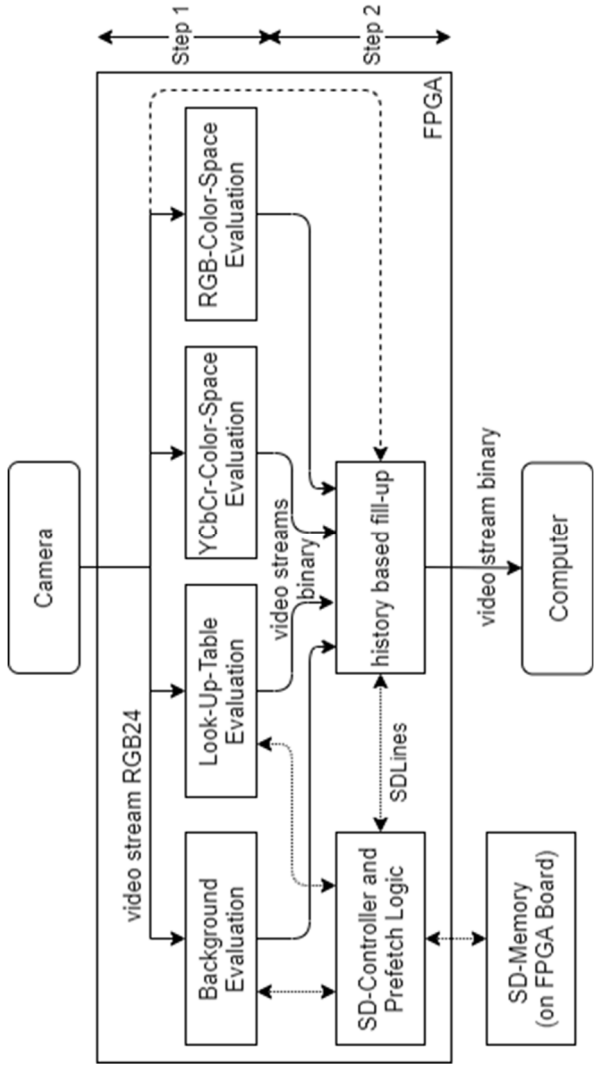

Fig. 1. Overall architecture for low latency FPGA implementation.

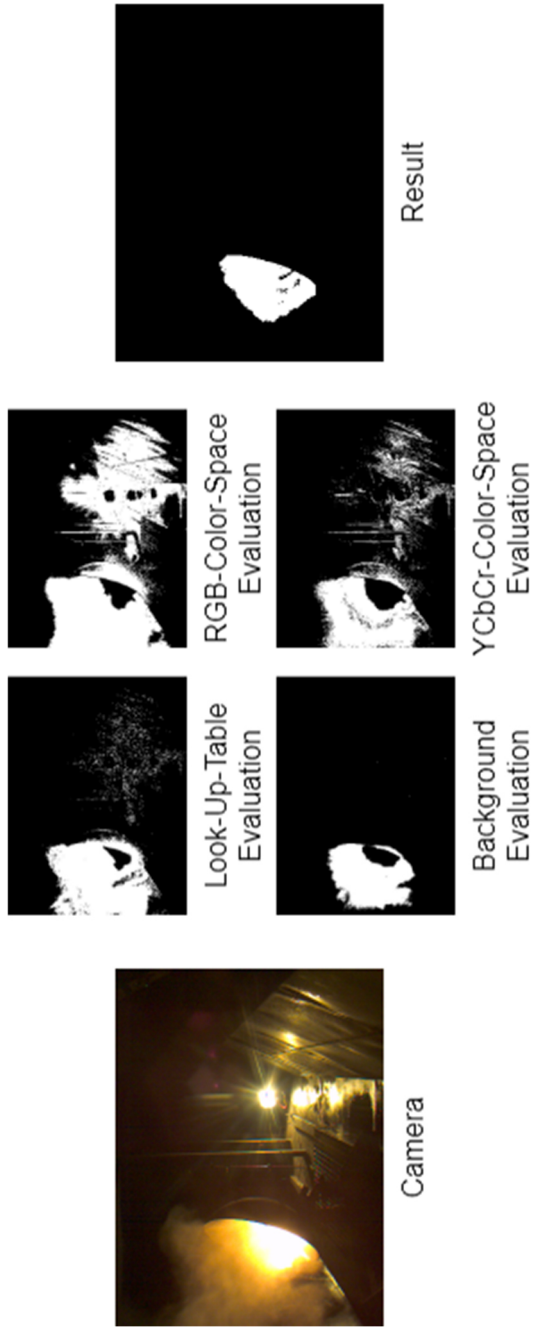

Fig. 2. Example image and its processing results generated by VHDL simulation model.

To use the full potential of an FPGA, two changes have been applied to the original software implementation:

1) The color space evaluation parts of the algorithm are based on average calculations (Eq. 1 and Eq. 2). To compute the average of color channels would require to handle one image twice before the final comparison decisions could be made. Therefore, the FPGA-implementation of the color space evaluation parts uses the average values of the previous image. Software simulation experiments showed that this does not result in considerable differences for the overall result.

2) As already described along with Eq. 5, in step 2 of the ported algorithm (history based fill up), the number n is limited to a very low value in the CPU/GPU software implementation for performance reasons. For the FPGA implementation it is possible to use a higher value for n (16 for the first demonstrator, see subsequent section) without suffering from a higher processing latency. Furthermore, this enhances the overall robustness of the algorithm.

However, the most challenging part of getting a synthesizable VHDL-model is not in transforming the deflagration detection algorithm. The challenging problem to be solved is the need to store information of the previous image. Unfortunately, the memory capacity, provided by the used FPGA itself, is not sufficient to store all of the required data. To just use a bigger FPGA is not a suitable solution, as this will not scale for higher resolutions (one will always find a resolution for which the biggest FPGA is not big enough). Hence a SD-Memory controller is needed, to allow for the usage of external SD-Memory. This usage of external SD-Memory introduces a problem to the real-time constraints: Answer latency of SD-Memory is unpredictable, due to its constantly recurring refresh needs. However, this problem can be solved by introducing a prefetch mechanism (requesting data early enough and buffer it in a queue) of the old image data since the required order of the old image data is known in advance

The first demonstrator implementation connects a cheap OV7670 camera with a Nexys4DDR-FPGA-Board. Thus, the FPGA computes the raw video signal. The OV7670 camera is used in 640x480 Pixel at 60 Hz refresh rate mode, resulting in a pixel-clock frequency of 25 MHz.

Results and Discussion

A comparison of the initial software models and the VHDL-models (simulation and synthesizable models) show minimal, but negligible, differences, which are caused by the already mentioned change in the average value usage and the waiving of opening/closing operations in the current implementation (see “Further Research” below).

The demonstrator shows that the segmentation algorithm works as intended on the FPGA. A pixel-clock of 25 MHz results in a latency of 80 ns (two clock cycles, 40 ns each). Hence, the demonstrator is able to prove that the VHDL models are synthesizable to an FPGA.

However, the strong noise and the low refresh rate of the underlying cheap CMOS-sensor results in noticeable artifacts in comparison to simulations with higher quality videos. Also, the current refresh rate would be too low for deflagration detection.

Thus, future demonstrators should use CMOS sensors with less noise and a higher refresh rate. This might introduce a problem for the color-space based evaluation parts, especially the YCbCr Color-Space-Evaluation: As this evaluation requires multiplications, the pixel clock frequency might be too high (200 MHz or higher) to yield the multiplication results within one clock cycle. However, well known parallelization techniques like pipelining or superscalar processing can be used. This would result in an increased amount of clock cycles needed, however, also due to the higher clock frequency, overall latency would still be around 100 ns.

The amount of FPGA resources needed by the first demonstrator is shown in Table 1. A detailed explanation of the FPGA resources is beyond the scope of this paper. However, the overview demonstrates that the deflagration detection algorithm itself only consumes little resources compared to the OV7670-camera management hardware and the previous image and SD-Memory controller hardware – as would be expected by experienced hardware developers.

Table 1. FPGA resource utilization for demonstrator.

Instance/Part	Slices	BlockRAM	Multipliers (DSP)
Clock management	10	0	0
OV7670 management	265	3	0
previous image prefetch	291	5	0
SD-Memory controller	1432	6	0
Background Evaluation	9	0	0
Look-Up-Table Evaluation	8	0	0
RGB-Color-Space Evaluation	35	0	0
YCbCr-Color-Space Evaluation	72	0	3
History based Fill-Up	5	0	0

The minimalistic amount of used hardware for the steps *Background* and *Look-Up-Table Evaluation* as well as for the *history-based Fill-Up* (less than 10 slices) can be explained easily: these evaluation steps are simple threshold comparison problems, which can be implemented with small amounts of hardware. The slightly higher resource demand for the Color-Space Evaluations can be explained by the need to calculate mean values over the entire image, which requires the accumulation and a division by a constant (the number of pixels per frame). Most interestingly, the YCbCr-Color-Space evaluation (which further includes a color-space transformation from RGB) uses 3 multipliers. However, due to the RGB to YCbCr transformation, one would expect 9 multipliers. This is because the synthesis tool replaces some of the multiplications with constants by a faster shift and add method.

Hence the missing 6 multiplications are hidden in the slices, used by the YCbCr-Color-Space Evaluation part.

Conclusion

In this contribution, it has been shown that the deflagration detection algorithm can be transferred to an FPGA, resulting in a system capable of hard real-time and reducing the computation time from approx. 3.4 ms to negligible 80 ns; i.e. results are available nearly instantly when the complete pixel stream has reached the FPGA. The detection time of the camera-based system can thus now be compared to the infrared sensors currently in use, while additionally providing additional information like the position and the approximate volume of the deflagration.

Furthermore, the FPGA hardware architecture also allows to consider a high number of previous images while “saving” pixels in the localization segmentation without additional calculation time. This is a further advantage compared to the CPU/GPU implementation. At the same time, the hardware used in the demonstrator is very affordable.

Future Research

So far, only phase one of the overall deflagration detection algorithm has been ported to an FPGA. To complete the implementation of phase one, opening and closing operations should be ported to the FPGA, i.e. to fill the small black holes in the segmentation detection results (see Fig. 2). These operations are already part of the original algorithm and methods to port them to an FPGA are well known [6]. Additionally, the calculation of the segmentations should be ported. When the moment of a deflagration is calculated within the FPGA, there is no further need to send any images (videos) to the next processing steps.

Currently, phase two of the algorithm, i.e. the further processing of the moments of different cameras to calculate the 3D-coordinates of a deflagration, as well as volume estimation and advanced detection, are computed on a PC. However, this can be computed also within the FPGA or on an Embedded System.

Beside porting further parts of the deflagration detection onto an FPGA, also the demonstrator needs enhancements, as already discussed above: A CMOS-Sensor with less noise and a higher sample/refresh rate should be used.

References

[1] T. Schröder, K. Krüger, F. Kümmerlen, *"Fast Detection of Deflagrations Using Image Processing",* Proc. Suppression, Detection and Signaling Research and Applications Symposium (SUPDET) 2013.

[2] A. Kim, *"Overview of Recent Progress in Fire Suppression Technology",* Invited Keynote Lecture at the 2nd NRIFD Symposium, Proceedings, Tokyo, Japan, July 17-19, 2002, pp. 1-13.

[3] J. Krooß, F. Kümmerlen, A. Fay, *"Detection, Localization and Volume Estimation of Deflagrations"*, IFAC 2020 World Congress, Berlin, Germany, July 12-17 2020.

[4] T. Ernst, F. Kümmerlen, A. Fay, *"Real-time Image Processing Based Deflagration Detection and Localisation"*, Security Research Conference 11, Berlin, Germany, September 13-14 2016, pp. 339-346.

[5] A. Gaur et. Al., *"Video Flame and Smoke Based Fire Detection Algorithms: A Literature Review"*, Fire Technology 56(5), 2020, pp. 1943-1980.

[6] D. G. Bailey, *"Design for Embedded Image Processing on FPGAs",* John Wiley & Sons (Asia) Pte Ltd, ISBN: 978-0-470-82849-6, 2011.