Issues of hardware implementation of image based vehicle detection

Wieslaw Pamula
Faculty of Transport
The Silesian University of Technology
Krasińskiego 8, 40-019 Katowice, Poland
wieslaw.pamula@polsl.pl

Abstract. *The paper presents the discussion of ways of implementation in hardware of a vehicle detection algorithm. Instruction set architectures and configurable processing architectures are assessed for the demanding video stream calculation tasks. The significance of decomposition of the detection algorithm for achieving set forth efficiency and speed requirements is presented.*

The vehicle detection algorithm is based on extracting moving objects on a slowly changing background. The algorithm may be divided into data driven computation blocks. It is shown that a processing pipeline, designed using reconfigurable FPGA resources, satisfies implementation criteria providing a robust solution for application in road traffic control systems.

Keywords: image processing, processing architectures, vehicle detection

1 Introduction

Image processing requires the highest processing capabilities available but at a cost, which is acceptable for the end user. This requirement is hard to comply with using ordinary video processing equipment. A way to meet this requirement is to use specialized processing hardware. Such hardware extensively utilizes parallel processing.

The problem of concurrent execution of different processing tasks can be solved using multiple processing units working in parallel on a set of data or feeding a pipeline of processors with parts of data, which make up the original data set

The first solution is highly flexible, but unnecessarily complicated as image processing can be carried out efficiently, using simple processing elements working on small pixel neighbourhoods.

The second solution based on partitioning of the data for processing can be implemented very effectively using gate arrays. Gate arrays especially FPGA, which can easily be reconfigured, allow for efficient design of multiple simple processing blocks.

The basic image processing tasks are centred on neighbourhood operations so it is necessary to map the acquired image to the gate structure in a way to streamline processing. Elementary logic blocks inside the gate arrays consist of a LUT, a few FF and multiplexers. This is not enough to implement even a simple local filter on a 4-connected neighbourhood of an image pixel. The logic blocks need to be combined into more complex structures.

Different researchers present two main groups of solutions for the problem of efficient use of FPGA resources [2]. First is based on designing specialized window processing elements and the other on concepts utilising processing pipelines.

In [1] characteristics of different moving window architectures are discussed showing that throughput is dependent on the product of number of clock cycles per pixel and the number of FF in a window processing element for completing a processing task. In order to choose the optimal architecture for a particular image processing algorithm task, it is necessary to consider the way the algorithm is partitioned into elementary steps. A window-based architecture is most suitable for implementing 2D convolvers [4, 16].

Design of a pipeline based image processor requires careful decomposition of the processing task. The decomposition must take into account the way pixel data will be exchanged between memory blocks and pipelines. A very important issue is to determine the level of complexity of operations to be performed by the pipeline stages.

Example systems with pipeline architectures prove that appropriate distribution of processing tasks efficiently use available FPGA resources [5, 11].

The paper is divided into 5 sections. Section 2 outlines the vehicle detection algorithm; in the next section discussion of implementation issues is presented. The 4th section presents a pipeline based implementation. The ending section proposes further work on tuning the solution.

2 Vehicle Detection Algorithm

It is specified that data for processing is provided by a standard CCTV camera mounted above a section of road, where vehicles are to be detected.

The task of detecting vehicles is defined as recognizing the presence of moving objects in defined detection fields. It is assumed that there are no other objects than vehicles in the analysed video sequence. Such a conjecture simplifies the problem of detecting vehicles.

Several approaches for object detection were analysed and evaluated [9, 14]. It was decided to use background subtraction to highlight moving objects. This simple method requires efficient modelling of background data. The method must be immune to noise and ambient light changes.

A modified scheme using recursive first order filter and pixel value statistics is used as the base of adapting the value of background pixels to current traffic and light situation [3, 10, 15].

An object oriented approach was chosen for specifying the scope of processing requirements and related data structures of the algorithm [6]. The basic objects are data structures associated with memory operations. Calculation tasks further characterize their properties.

Fig. 1. presents the outline of the detection algorithm. An unmarked Petri net is used to link data objects – places and processing steps – transitions of the algorithm [12]. Analysis of properties of the net and possible transformation will indicate ways for optimising the processing flow [8]. This model was chosen to emphasize processing requirements and separate them from hardware particularities thus enabling a discussion of implementation variants.

Apparent concurrency is of special interest as it contributes to high processing throughput. Desirable are regular data objects in order to simplify memory operations.

Input data object of the processing scheme is raw data from a CCTV camera. This data structure is processed - transition t_1 to obtain a video frame object and synchronization signals, marking the start of detection cycles. These objects supplemented with a background frame and detection masks are input places for transition t_2 which calculates occupancy states of detection fields.

There are 32 detection fields. Masks define the fields; additionally each field has an occupation threshold and sensitivity coefficient assigned during a configuration procedure. These values are used for calculating the state of detection field.

Masks are stored in local memory.

The crucial object *background* is derived using a number of auxiliary data objects and associated processing tasks as shown in fig.2.

This second net diagram, in fact, explicitly shows the complexity of data transfers and size of necessary memory objects.

Updating image pixel values statistics is done in 3 separate ranges. Only the mean values are taken into consideration when estimating background. The statistics are calculated over 256 update periods. Direct calculation of mean values requires the allocation of at least 16 bit memory cells to fit the growing value of the sums of pixels. To save memory resources the mean value is calculated on the run [14].

Pixel value statistics are calculated for every input frame pixel so there are 6 tables of the size of a video frame. Other data objects such as detection fields masks, pixel update frequencies and current video frame should also reside in memory.

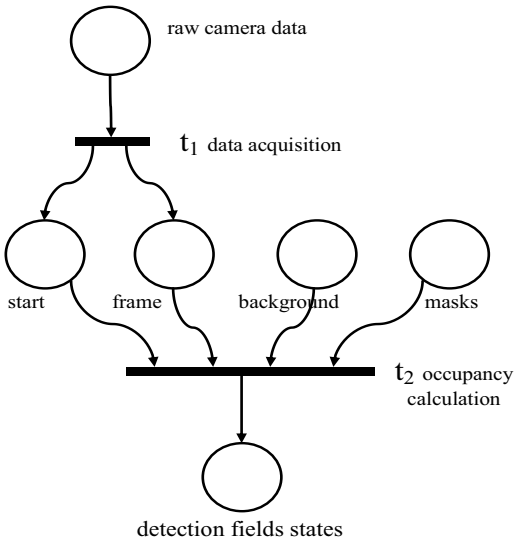

Figure 1. Detection algorithm

Background frame is updated using three different calculation procedures – t_{10}, t_{11}, t_{12} transitions, which are chosen in accordance with current video frame number – t_6, t_7, t_8 transitions.

Tables are concatenated into one data object using t_3.

BI update calculates the preliminary value of background as the most probable mean values of pixels from 3 value ranges. A simplified version of statistical modelling is used, which incorporates three fixed centroids and no dispersion calculation.

BIII update makes a "sleeping person" correction once every 6,4k frames using a max function. The max function uses the entity *tables* for determining the correction values.

The core procedure for determining background values, *bII* update, derives the new value using a linear function of previous background value and current pixel value [10].

When the absolute difference of the current frame and background, which may be regarded as a mask, is larger than a set threshold the background is calculated as:

$$B(t) = aF(t) + (1 - a)B(t - 1)$$

where:
B - background value, F - pixel value, t - update count, a $\in(0, 1)$.
otherwise it is not updated.

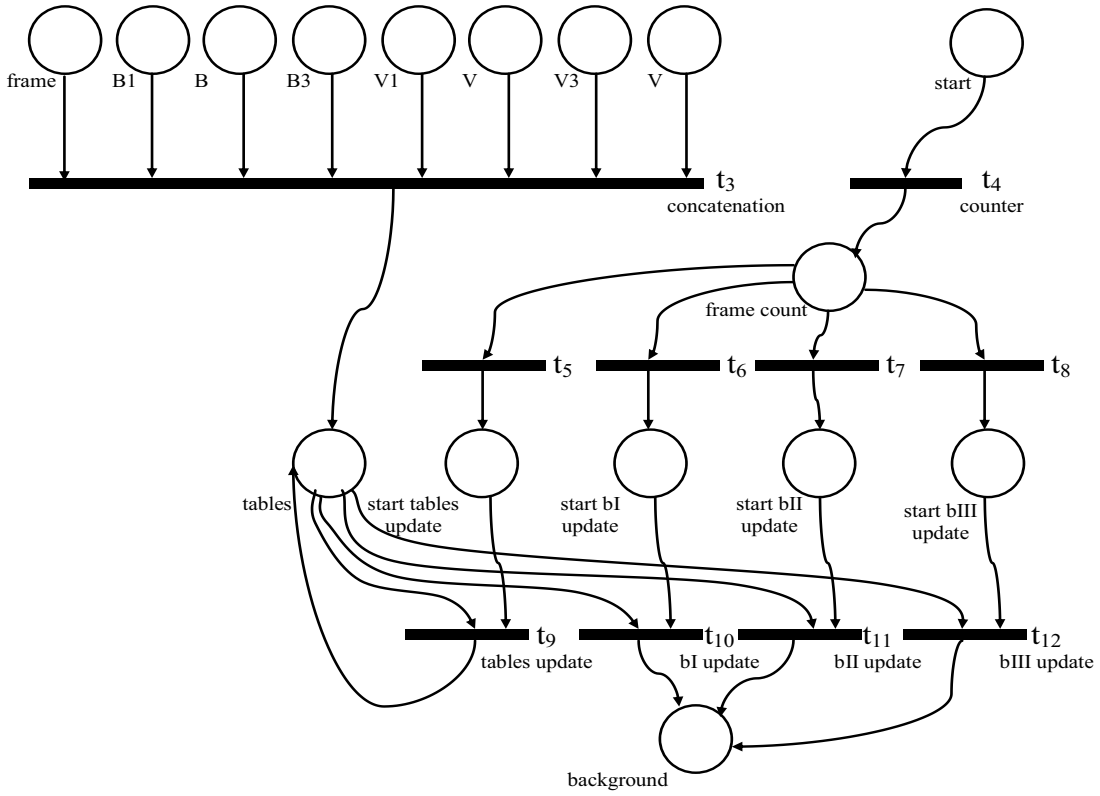

Figure 2. Background calculation

Prior to hardware implementation a software version of the algorithm was tested on a set of video streams from road traffic cameras. The software run on a PC (3,4GHz P4) equipped with a frame grabber and a massive hard disk. More than 50 hours of different traffic situations were processed [14]. The video streams were analysed at 1 frame per second of processor time.

More than 95% of vehicles arriving at detection fields were accounted for.

3 Implementation Discussion

In order to design efficient hardware architecture, data structures and processing requirements were reviewed. Besides these hardware specification requires additionally a time scale for performing operations and a concept of register level transfer operations.

3.1 Memory Resources

To ensure the freedom of choice, of memory solutions, diagrams do not include memory operations. Storage technology defines the time budget of the implementation.

The basic data object has the size of a video frame that is roughly 0,4M bytes. This entity may be saved or retrieved several times during a detection cycle.

Transition t_9 uses the largest amount of memory resources, in all 8 objects, which are read and written back to a memory structure. The way of accessing memory and its execution time define the resultant performance of processing organization.

The obvious choice for memory is static RAMs. To achieve real time operation that is a detection cycle equal to video frame acquisition, memory access time cannot exceed $t_{acc}=t_{frame}/(8*0,4\cdot10^6*2)$.

Assuming a word length of 1byte, t_{frame} equal to 40 ms, t_{acc} must be less than 6,3 ns. Such a small access time requirement may be ensured only with costly, very high speed devices.

The other choice is dynamic RAMs. In this case to achieve comparable throughput, block exchange of data is necessary. Block exchange although fast requires cache memory to facilitate uninterrupted data processing. Addressing consecutive blocks forces breaks in data flow which should be filled by data from auxiliary memory such as cache.

To ease the dictate of memory speed long memory words may be envisaged. Consistent with the data objects in net diagram of fig.2. the entity *tables* is a suitable candidate for deriving the word's length. It is a concatenate of 8 data objects. Objects use byte long words so the resultant word length may be 64 bits.

3.2 Processing Architectures

The presented net diagrams show data flows and processing tasks, imposing no constraints on hardware implementation. One can devise a processor based unit or a configurable logic array for implementing these.

Devising a processor solution requires modification of transitions to account for processor specifics. Execution efficiency will highly depend on the size of used processor word and features of available instruction set.

Especially t_9 transition requires attention as it involves the largest data object.

Embedded processors which are preferred for use in control applications usually work with 8 to 32 bit long words. These components lack elaborate memory management units and associated control instructions. This implies that transition t_9 has to be converted to a series of tasks working on parts of the entity *tables*. In consequence execution time may be 2 to 4 times longer in comparison to whole entity processing.

Besides memory accesses several operations are applied to data entities which require a number of processor machine cycles to complete.

A complete processor based solution comprises a processor block, memory management unit, program and data memory. Usually embedded processors contain on chip programme memory which is sufficient to store instruction codes for necessary data operations.

Analysis of net diagrams indicates that many transitions may be performed concurrently. This feature can be effectively utilized in elaborating the configuration of logic arrays.

Architectures using arrays of logic gates can be tailored to frameworks of concurrently acting components.

Distinctive feature of logic based implementation is the ability to merge operations and derive results without multiple data relocations. Although it is possible to organize asynchronous operation, in this case, when data needs to be fetched from memory, clocking is more desirable. Synchronizing operations additionally eliminate data skew which may arise when manipulating large data entities.

Merging background updates bI, bII, bIII creates a component purely acting on data from tables with no memory write back path. The data does not have to be read from memory it can be routed from other components which perform memory accesses in the same time slots. The inherent flexibility of routing data which is available in logic arrays is of use in this case.

T_9 transition remains the only one task requiring bi-directional memory access. With no limits on data length a whole data entity update can be carried out. Additionally a RAM controller may be incorporated to organize data exchange with a dynamic RAM.

3.3 Frame vs. Pixel Processing

The conjecture that processing done with video frames as basic data objects facilitates real time implementation is hard to prove.

"Off the shelf" components have significantly lower performance than anticipated. Ordinary memory access times are in the range of tens of ns which leads to even 10 times longer data exchange times. Adding to this processor machine cycles of a similar duration, required for data processing, vehicle detection times of several hundred ms may be attained.

A solution based on logic arrays is much faster but still falls behind real time timing target. The main source of time loss is inefficient frame based memory access. Streamlining data exchange is the key to successful algorithm implementation.

Pixel by pixel processing is an alternative solution. At first it may appear infeasible.

A modification of net diagrams by redefining data entities is adequate to model this alternative solution. New entities refer to single pixel data and associated auxiliary table entries. Revision of fig.2 indicates that only the scope of transitions changes. In result a new approach for organizing data exchanges is required.

Video data is usually acquired using a standard video decoder IC which provides data synchronously with a 27 MHz clock. Data is coded in compliance with ITU-R BT 656 standard. Luminance is outputted every second clock cycle and this is the time interval in which all transitions must fit in.

The pixel processing cycle is therefore 74ns long. Crucial t_9 transition must be optimised to fit in this time slot.

Processor based solution especially using embedded devices is in no way suitable for such a demanding task.

Utilization of caching memory operations and logic circuits for conducting data operations should meet this timing constraint.

4 Hardware Design

As pointed out in part 3 specialized hardware must be considered as the basis for implementing the detection algorithm. Such hardware, based on configurable arrays of logic gates, has the very serious handicap of scarce memory resources. Devices usually posses a few hundred thousand bits of memory, that is far too little to contain an image frame and associated statistic tables.

As remarked previously hardware specification requires a time scale for operations and a register skeleton to provide the means for data exchange in the processing framework.

The time scale for operations may be determined by the speed of acquiring picture elements from the video camera or by the update rate of the detection fields' occupation.

In the first instance all operations must be performed between two consecutive pixel acquisitions, while the second time limit is much more lenient and may amount even to a number of video frame periods. The length is dependent on update needs of a traffic controller which will incorporate the vehicle detector.

To cover update rates of many controllers the concept of pixel by pixel processing was accepted as the design base.

Several architecture frameworks were analysed and their characteristics were evaluated [14]. In all cases the problem of storing large amounts of auxiliary data such as tables had to be addressed. To diminish hardware costs a DDRAM was incorporated.

The detection algorithm as presented on fig.1 and 2 comprises data objects and processing tasks. The processing tasks were assigned to components with clearly defined data interfaces.

The hardware framework consists of components for calculating backgrounds, detection fields' occupancy and data exchange with camera and DDRAM.

To sustain the required data flow a specialized dynamic RAM controller was designed. This component incorporates dual port block memories acting as cache memories for data transfers. One of the ports is handled by logic organizing dynamic RAM block transfer operations while the other is used for pixel access.

Using a 32 bit data path for RAM access throughput of over 0,5GB/sek may be achieved without excessively straining RAM components.

Fig. 3. presents the pixel based architecture. It is a processing pipeline fed by data from the DDRAM memory. The pipeline setup reproduces the algorithm diagrams with the exception that entities are acted upon and pushed through components corresponding to transitions.

Transitions t_5, t_6, t_7, t_8 are merged into one component - the processing sequence controller.

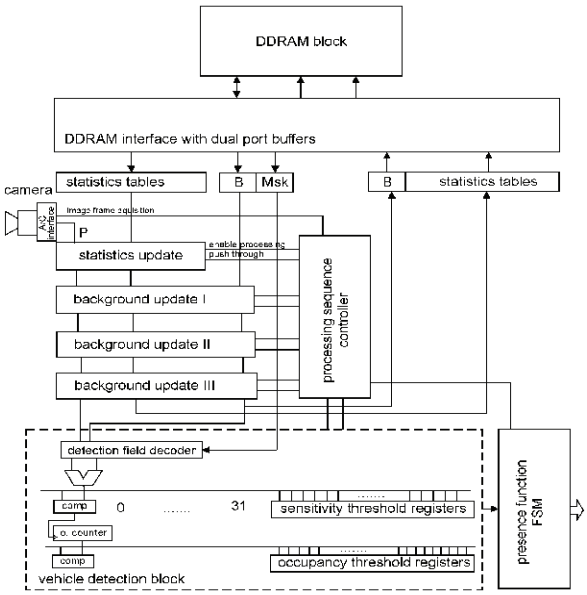

Figure 3. Processing pipeline for detecting vehicles

The processing sequence controller pushes data through processing stages. When data is not needed for calculation it is just pushed through. Results are written back to memory.

The stages of this pipeline are implemented in a low cost FPGA. Video stream comes from a digitized signal of a CCTV PAL camera. A Xilinx Spartan 3E device is the logic array component used [7].

The hardware implementing the vehicle detection algorithm must be augmented with controllers interface for passing detection results and suitable configuration protocols.

5 Conclusions and Future Work

Object oriented approach to hardware design proved its usefulness. Goals set for the design were met. Currently first prototypes are field tested and detection results are compared against traditional loop detectors.

Further reduction of the designed hardware structure is possible by changing the way pixels are ordered in memory. The presented solution utilizes two-dimensional arrays and elaborates data row by row. This requires counters for controlling row numbers and picking pixels within rows. The size of rows is not a power of two, which further complicates control circuits.

By using a vector representation of image data it will be possible to eliminate row counters and simplify control circuitry [13].

7 Acknowledgments

This research is a part of a project financed by the European Union and the Polish Ministry of Science and Higher Education.

References

[1] Battle J, Marti J, Ridao P, Amar J: New FPGA/DSP-Based Parallel Architecture for Real-Time Image Processing, Real Time Imaging 2002, vol. 8:345-356

[2] Benitez D: Performance of reconfigurable architectures for image-processing applications, Journal of Systems Architecture 2002, vol. 49:193-210

[3] Bhandarkar S, Luo X: An Efficient Background Updating Scheme for Real-time Traffic Monitoring. IEEE ITS Conference Washington USA Oct 2004, pp.859-864

[4] Cardells-Tormo F, Moline P: Area-Efficient 2-D Shift-Variant Convolvers for FPGA-Based Digital Image Processing, IEEE Transactions On Circuits And Systems 2006, vol. 53, No. 2

[5] Chang C-C, Huang Z-Y, Li H-Y, Hu K-T, Tseng W: Pipelined Operation of Image Capturing and Processing Proceedings of 2005 5th IEEE Conference on Nanotechnology Nagoya, Japan

[6] Damasevicius R, Stuikys V: Application of the object-oriented principles for hardware and embedded system design. Integration the VLSI Journal 2004, vol.38:309-339

[7] Ehrig H, Hoffmann K, Padberg J: Transformations of Petri Nets, Electronic Notes in Theoretical Computer Science 2006, vol.148:151-172

[8] FPGA Features and Design. Xilinx Inc. San Jose, CA USA 2007

[9] Heikkila J, Silven O: A real-time system for monitoring of cyclists and pedestrians. Image Vision Computing 2000, vol.22:563-570

[10] Koller D, Weber J W, Malik J: Robust Multiple Car Tracking with Occlusion Reasoning, Proc. European ConJon Computer Vision, Stockholm, Sweden, May 1994, pp.189-196

[11] Li J, An X, Ye L, He H: A Reconfigurable Parallel Architecture for Image Computing, IEEE Proceedings of the 6th World Congress on Intelligent Control and Automation, June 21-23, 2006, Dalian, China

[12] Murata T: Petri Nets: Properties, Analysis and Applications, Proceedings of the IEEE, 1989, vol.77:541-580

[13] Pamula W: Advantages of Using Space Filling Curve for Computing Wavelet Transforms of Road Traffic Images. International Conference on Image Analysis and Processing, Mantova, Italy, IEEE Society Press pp. 614-620

[14] Project Report: Modules of Video Traffic Incidents Detectors ZIR-WD for Road Traffic Control and Surveillance. WKP-1/1.4.1/1/2005/14/14/231/2005, Katowice, Poland 2007

[15] Stauer C, Grimson W E L: Adaptive background mixture models for real-time tracking in: Computer Vision and Pattern Recognition, Fort Collins, Colorado, June 1999, pp. 246-252

[16] Torres-Huitzil C, Arias-Estrada M: Configurable Hardware Architecture for Real-Time Window-Based Image Processing, LNCS 2003, vol.2778:1008-1011