Real-Time Object Tracking System on FPGAs*

Su Liu[1], Alexandros Papakonstantinou[2], Hongjun Wang[1], Deming Chen[2]
[1]School of Information Science and Engineering, Shandong University, Jinan, China
[2]Electrical & Computer Engineering Department, University of Illinois, Urbana-Champaign, IL, USA
[1]sueliu84@gmail.com, [2]{apapako2, dchen}@illinois.edu, [1]hjw@sdu.edu.cn

***Abstract*— Object tracking is an important task in computer vision applications. One of the crucial challenges is the real-time speed requirement. In this paper we implement an object tracking system in reconfigurable hardware using an efficient parallel architecture. In our implementation, we adopt a background subtraction based algorithm. The designed object tracker exploits hardware parallelism to achieve high system speed. We also propose a dual object region search technique to further boost the performance of our system under complex tracking conditions. For our hardware implementation we use the Altera Stratix III EP3SL340H1152C2 FPGA device. We compare the proposed FPGA-based implementation with the software implementation running on a 2.2 GHz processor. The observed speedup can reach more than 100X for complex video inputs.**

Keywords - object tracking; FPGA; application acceleration.

I. INTRODUCTION

Computer vision has become an important application of smart embedded systems used in a wide range of fields ranging from human computer interaction to robotics. Object tracking is a fundamental component of computer vision that can be very beneficial in applications such as unmanned vehicles, surveillance, automated traffic control, biomedical image analysis and intelligent robots, to name a few. Object tracking is used for identifying the trajectory of moving objects in video frame sequences [1]. Like most computer vision tasks, object tracking involves intensive computation in order to extract the desired information from high-volume video data. In addition, the real time processing requirements of different computer vision applications stress the need for high performance object tracking implementations. In this work we propose the implementation of an efficient object-tracking system on FPGA that could be employed in a wide range of embedded systems providing high-performance and low-power.

With shrinking process technologies enabling higher transistor capacities per silicon die, FPGAs have become attractive compute platforms for complex applications with high-performance and low-power requirements. With hundreds of thousands of configurable logic blocks along with thousands of distributed memory and hard DSP modules, they offer great flexibility for mapping applications onto spatially parallel architectures. Nevertheless, exploiting the advantages of their re-configurable and hard modules requires efficient mapping of algorithms through careful balancing of performance, area and power parameters. In this paper we describe our object tracking implementation on the Altera Stratix III FPGA. Through profiling and analysis of the software implementation we identified the performance bottlenecks and designed a hardware architecture which leverages efficiently the spatial parallelism of the reconfigurable fabric and exposes the different types of inherent parallelism in the selected object tracking algorithm. Our experimental results show that significant performance improvement (over 100X) can be achieved compared to the software execution for multi-objects video inputs.

The contributions of this work can be summarized as follows:

- We propose a highly parallel hardware implementation of an object tracking algorithm.
- We improve the object region identification performance of the object tracking algorithm by introducing a dual search technique.
- We provide experimental results that show that our hardware implementation achieves up to 100X speedup over the software execution.

In the next section we discuss related work and we provide a detailed overview of the selected object tracking algorithm in Section III. The proposed hardware implementation on the Altera FPGA is presented in Section IV followed by experimental results in Section V and conclusions in Section VI.

II. RELATED WORK

Numerous algorithms for object tracking have been proposed. It is a complex task which comprises two main subtasks: i) object detection and ii) tracking. Object detection algorithms can be classified according to Yilmaz et al [1] into point-detection schemes [7], background subtraction techniques [8], and supervised learning techniques [9-11]. Furthermore the tracking portion of object tracking can be performed either separately or jointly with object detection. Tracking aims to generate the trajectory of objects across video frames and Yilmaz et al. [1] characterized tracking algorithms across three main categories: i) point tracking [12], ii) kernel tracking [13], and iii) silhouette tracking [14].

*Su Liu completed this work during the period when she was a visiting student at University of Illinois.

In this work we leverage an efficient algorithm which is based on background subtraction detection and kernel tracking of objects. The main focus of this work is on the performance improvement achieved over a software implementation from a carefully-crafted hardware implementation on the FPGA.

Due to the advantages offered by FPGAs in compute intensive applications, several object tracking algorithms have been implemented on reconfigurable devices in previous works. Nevertheless, one of the biggest challenges of custom hardware implementations is mapping complex algorithms onto reconfigurable fabric architectures that can offer good performance under rigid resource constraints. Jung et al. [3] implemented a multiple objects tracking system on hardware based on particle filters [15]. However, the tracking speed was below 57 frames per second (fps) and no comparison with software execution was provided. Usman et al. [4] adopted an FPGA software processor based design to implement mean shift [1] based object tracking. However, the biggest size of tracked objects is limited to 32X32 pixels and the performance does not exceed 25 fps. Jinbo et al. [5] implemented a hardware detection system based on the Active Shape Model (ASM) algorithm, and they reported speedup of up to 15X compared to software execution. However, their implementation does not include tracking. A multi-camera object tracking system based on multiple-Cam-shift algorithm is implemented by Sirisak [6], but the reported speed is constrained to 25 fps.

Our FPGA implementation employs a carefully designed parallel architecture which helps significantly boost tracking performance over software execution. Moreover, we propose a dual-search method to utilize resources in an efficient and compact way. For the evaluation of our implementation, we use an Altera Stratix-III EP3SL340H1152C2 FPGA device. We achieve 21X to 104X speedup over the software execution with performance that ranges between 70-690fps for frames with 0-6 objects.

III. Object Tracking Algorithm

The algorithm employed in this work is based on the background subtraction object tracking algorithm proposed by Yuri [2]. This algorithm tracks moving objects in video frames captured by fixed cameras (i.e. non moving cameras). Initially the background scene is built through averaging of several successive frames to handle time varying background scenes, such as waves on the water, moving clouds etc. This task is called background training and it helps build a reference frame which can be used to classify object regions during the actual object tracking processing. Generally, the reference frame is formed based on a weighted averaging function which takes as input N previous frames during background training. After the reference scene has been established, all subsequent frames will be classified in relation to it. The reference frame is updated when the values of all the frame pixels have changed significantly with respect to the corresponding reference frame pixels.

Our hardware implementation takes VGA resolution (640x480) video as input. The incoming video frames are processed within three main stages: i) preprocessing stage, ii) main detection and search stage and iii) tracking and display stage. In the preprocessing stage, the VGA-resolution input frames are downscaled to 80x60 resolution. Subsequently, in the main detection and search stage classification of the downscaled frame is performed with regard to the reference frame and the object tracking map is generated. In addition, further object region processing steps are implemented in this stage. Finally, in the third stage, the locations of moving objects are identified and marked on the screen. In the following sub-sections we describe the three processing stages in more detail.

A. Preprocessing Stage

During the preprocessing stage the resolution of input frames is reduced from 640x480 to 80x60. Each frame is downscaled by a factor of eight through 2D Haar transform. Each 2D Haar transformation contains two 1D Haar transformations that take place sequentially first along frame rows and then along pixel columns. The dimensions of the frame that is generated by one 2D Haar transform are half of the original frame dimensions. We use three consecutive 2D Haar transforms to get the 80x60 size frame on which the rest of the object tracking processing is performed. The main motivation for image compression in our design is related to storage resource constraints as well as computation throughput constraints. Through downscaling, a reduction of the data volume per frame is achieved (i.e. from 9000Kbits/frame to 140.6Kbits/frame). Hence, the volume of data that needs to be stored and processed is significantly cut down. Subsequently, the new background reference frame, B_{n+1}, is determined in the downscaled image resolution of 80x60, according to the following weighted averaging function

$$B_{n+1} = \alpha * F_n + (1 - \alpha) * B_n, \qquad (1)$$

where α is the background training rate (typically 0.05), Fn is the most recent input frame, B_n is the old reference frame and B_{n+1} is the newly trained reference frame. Tracking proceeds as normal until significant difference is measured in the input frame with regard to the reference frame. In that case the reference scene image is updated to the weighted average reference calculated by (1).

B. Main Detection Stage

An initial classification of each pixel in the current frame is performed in this stage. Background subtraction based object detection relies on the property that the color values of pixels within the frame regions of moving objects, generally differ greatly from those of the corresponding pixels in the reference frame of the background scene. Thus, the absolute difference of the RGB values between the corresponding pixels in the current frame and the reference frame is used to identify potential moving object locations. One difference value is calculated for each color component in the RGB representation of the frame. If the value for any of the R, G, or B absolute differences exceeds a predetermined threshold, the corresponding pixel is marked as *foreground* (i.e.

potential moving object region). Otherwise, it is marked as *background* (i.e. reference scene region).

The frame classification generates an 80x60 binary matrix, called *object map*. Each binary element in the object map corresponds to one pixel in the 80x60 downscaled frame, which is set to '1' or '0' depending on whether the corresponding pixel is classified as foreground or background, respectively. In particular the value of each element in the object map is determined as follows:

$$\Delta I_{t,R(x,y)} = |I_{t,R(x,y)} - I_{bg,R(x,y)}| \tag{2}$$

$$\Delta I_{t,G(x,y)} = |I_{t,G(x,y)} - I_{bg,G(x,y)}| \tag{3}$$

$$\Delta I_{t,B(x,y)} = |I_{t,B(x,y)} - I_{bg,B(x,y)}| \tag{4}$$

$$L(x,y) = \begin{cases} 1, & if\ \Delta I \geq th \\ 0, & if\ \Delta I < th \end{cases} \tag{5}$$

where $I_{t,R(x,y)}$ represents the Red color component of the $(x,y)^{th}$ pixel in the current frame and $I_{bg,R(x,y)}$ represents the Red color component of the (x,y)th pixel in the reference image. $L(x,y)$ is the classification value of the $(x,y)^{th}$ element in the object map.

Subsequently, the object map is processed through a filtering phase which aims to smoothen the edges of the identified object regions. The filtering phase comprises three transformation steps which are applied to reduce the noise and scratch-like artifacts in the object map. In particular, dilation and erosion filters with structure sizes of 3x3 and 5x5 are used in a three-step dilation-erosion-dilation sequence which generates a new object map.

The newly generated object map is used in the final object region identification phase. In particular the binary matrix of the object map is scanned to identify the exact location of each individual object in the image. We have improved the original algorithm of object region identification to include two object identification modes: i) single mode and ii) block mode. Moreover, we parallelize the object region identification by splitting the binary matrix of the object map into 12 sub-matrices. All the 12 sub-matrices are processed concurrently. Details for both the dual mode identification technique and the object map processing parallelization are discussed in Section IV.B.

C. *Tracking and Display Stage*

In this stage, the object region identification results generated in the previous stage for each sub-matrix are combined to calculate object region results for the entire frame. A *boundary joiner* module is used to build the final object position information from the partial information calculated for each sub-matrix of the object map. The object regions at the frame level are built through a 5-stage filtering process. Subsequently a bounding rectangle for each object is superimposed over each video frame and sent to the video output for display.

IV. Hardware Implementation

A. *System Architecture Overview*

In our implementation we have used the Altera DE3 Development Board to take advantage of the different input and output interfaces to implement and verify the object tracking system. A daughter board with a video camera is connected to the GPIO interface of the DE3 development board to provide real-time video data, while a VGA display is connected to the corresponding DVI output of the DE3 board to display the processed video with highlighted object tracking results. Apart from the FPGA device the DE3 board contains abundant DRAM memory (2GB of DDR2). The DDR2 memory is used to either temporally store the streaming video data input or pre-load video frame sets that need to be processed. To evaluate the performance of our object tracking system, we explored in our experimental study the second case. Thus, we were able to measure processing throughputs beyond the real-time restrictions imposed by the system camera. Such a scenario is useful to process pre-recorded videos to identify and track certain objects of interests.

A control button on the DE3 board allows the user to select between training and tracking modes in the system operation. In both modes the pre-processing stage converts the raw video frames to downscaled RGB frames. This is achieved with the use of a frame grabber module and a 2D Haar module. The frame grabber module converts raw data captured with a CCD camera into standard RGB image values for a 640x480 frame. Then the RGB data are stored in the DDR2 memory of the DE3 board. Due to the high compute density of the 2D Haar transform, the system employs a pipelined architecture for processing the Haar transform and the subsequent tracking computations without impacting throughput. That is, 2D Haar transform operates on the $(N+1)^{th}$ frame while the rest of the tracking hardware is processing the $(N)^{th}$ frame. Furthermore, the rich memory resource on the DE3 board is leveraged to build a quadruple buffering scheme to enhance performance (Fig. 2).

One of the most important compute throughput boosters in our hardware implementation is the architecture of the object region identification subsystem. This hardware subsystem integrates 12 object map processing modules (marked as P1–P12 in Fig. 2). Each of these modules processes one sub-matrix of the object map binary matrix. By processing the object map matrix with 12 parallel processing modules the FPGA implementation achieves significant speedup compared to the software implementation, which processes the object map elements in a sequential fashion.

As mentioned in the previous section, the proposed FPGA implementation incorporates a new object region identification technique which helps enhance the algorithm efficiency. In the proposed technique, there are two region identification modes: i) single mode and ii) block mode. The software version uses only the single mode which expands the identified object region by four neighboring points in the horizontal and vertical dimension, during each step. By introducing the block mode for object region identification, 16 points can be explored concurrently. By combining both modes, the proposed implementation achieves high processing throughput during the object region identification stage, especially for video inputs with large objects. The dual mode region identification technique is further detailed in Section IV.C.

The 12 object map processing modules are complemented with a five-stage cascaded boundary joiner in the tracking and display stage. The boundary joiner module is responsible for combining the results of the parallel object map processing modules to get the accurate position information of all moving objects. In particular, the boundary joiner decides which objects are sub-parts of larger objects in order to correctly identify the frame-level objects and highlight them in the output video stream. More details on the boundary joiner are provided in Section IV.D.

B. *Parallel Object Map Processing*

Based on execution latency profiling we have observed that the object region identification computation constitutes a big part of the total execution latency. The object region identification initially scans for foreground elements in the object map. Then a set of new elements to be examined is selected based on a 2D wavefront originating from the foreground elements which are stored in a BRAM. Therefore, the object region identification process entails an linearly growing set of elements with slope equals to 1 that have to be examined in each step. The computation density of each exploration step increases accordingly. In the original software version the exploration process is done in a sequential fashion, which results in high execution latencies. For example, a thousand comparisons need to be performed for a single exploration step in the worst case for the selected resolution. To improve execution latency we distribute the compute load to 12 parallel object exploration engines. We split the object map into 12 20x20 sub-matrices (in a 3x4 layout), which constrains the critical path latency to the time needed to examine 400 points in every sub-matrix. The advantage of this parallelism exposure is particularly useful in the case of large foreground regions.

As shown in Fig. 1, each object map processing module keeps track of foreground point by storing their coordinates into a BRAM of size 400x13 bits. There are 12 such BRAMs in our design.

C. *Dual-Mode Object Identification*

In our implementation, we combine the single mode and block modes to search the object map in a more efficient way. While in the single mode, the object region is built by starting from the first identified object element and then progressing to the four neighboring elements. Conversely, in the block mode, a four-element block is used as the object exploration unit, with neighboring blocks scanned in each step. Rather than 4 new pixels as in the single mode, the block mode examines up to 16 new pixels during each step; therefore, it is particularly suitable for video inputs with high ratios of object-to-frame area (i.e. objects cover a big percentage of the video frame area). By combining the single and block modes, object region identification gains considerable efficiency improvement. We evaluated the latency improvement with dual-mode object region identification over single mode for video input with dense distribution of objects. With single mode execution latency reached 13.4 ms while with dual-mode execution latency dropped to 9.8ms (i.e. a reduction of 26.9%).

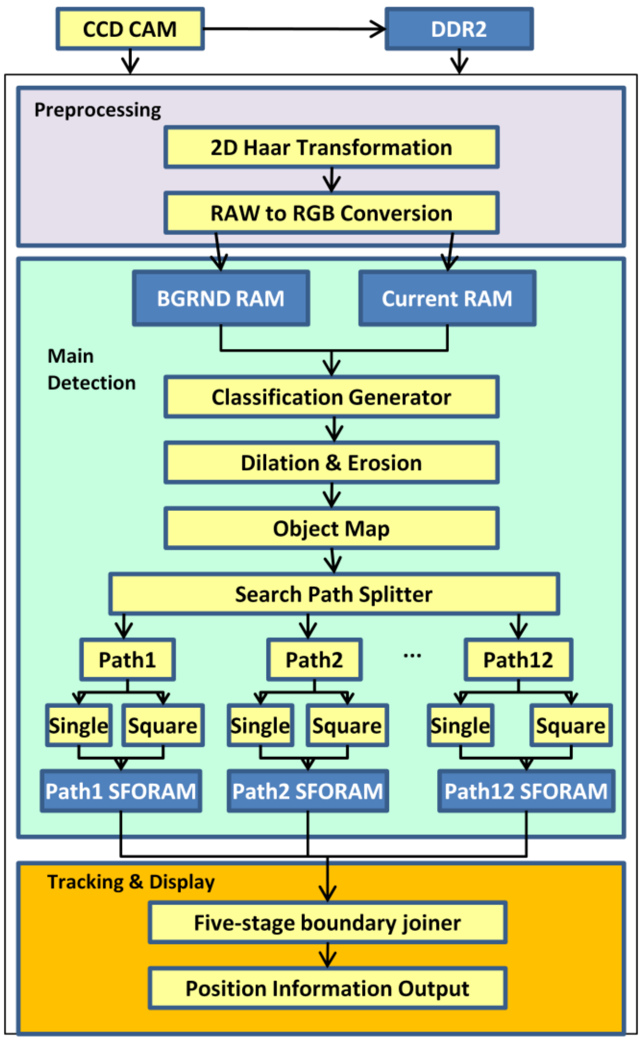


Figure 1. FPGA-based object tracking system – Block diagram

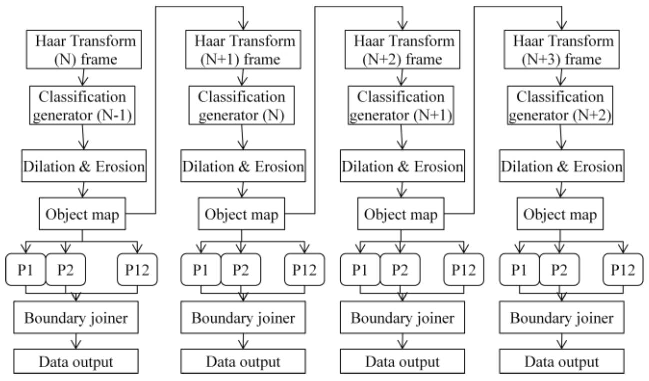
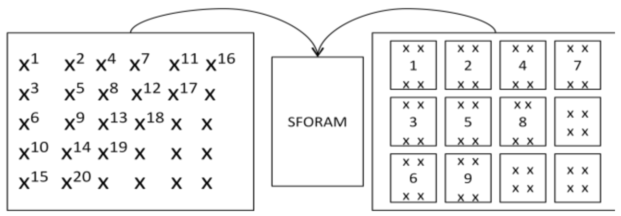

Haar Transform (N) frame
Classification generator (N-1)
Haar Transform (N+1) frame
Classification generator (N)
Haar Transform (N+2) frame
Classification generator (N+1)
Haar Transform (N+3) frame
Classification generator (N+2)
Dilation & Erosion
Object map
P1
P2
P12
Boundary joiner
Data output

Figure 2. System task-level parallelism



(a) Single mode (b) Block mode

Figure 3. Dual-mode search overview

Fig. 3(a) shows the single mode exploration whereas Fig. 3(b) shows the block mode exploration. As depicted in Fig. 4, we can see that the single mode is based on exploring four neighboring elements from the current object map element across the vertical and horizontal dimensions; the block mode is based on a similar exploration strategy but with the difference that exploration is performed on neighboring block regions instead of single elements. The numbers in both figures represent the exploration order: for example, if the top-left corner element x1 is the first foreground element detected in the object map, we examine its right neighbor element which is marked as x2 and the lower neighbor x3 (there are no left and upper neighbors for x1). After x1, we start examining neighbors of x2. Fig. 4 depicts the pseudo code for the single mode and block mode search algorithms which were implemented in Verilog in our hardware implementation. The pseudo code also describes how the system switched between the two modes.

D. *Cascade Boundary Joiner*

Due to the parallelization of the object map processing across 12 object exploration engines, the object region information for the entire frame is separated into 12 parts. Each part corresponds to one object map sub-matrix and it is stored in a separate on-chip buffer called *sub-frame object RAM* (SFORAM). Initially, the actual frame-wise physical information of the foreground elements contained in each SFORAM is recovered and stored into a *frame object RAM* (FORAM). Subsequently, a five-stage boundary joiner processes the physical position information in the FORAM (Fig. 5). Every partial object is represented as a rectangle window in the frame. The joiner algorithm first consolidates the object boundary information across row-wise neighboring sub-frames. As shown in Fig. 6(a), during the first three filtering steps, all the sub-frames across the horizontal axis are merged into frame-wide sub-frames. Subsequently, during the last two filtering steps, column-wise merging is performed to obtain the frame-wide object region boundaries. Fig. 6(b) lists the sub-frames being merged in each stage of the boundary joiner. Merging is performed for each neighboring pair of sub-frames, hence, generating larger and potentially overlapping sub-frames. Redundant and overlapping object boundaries are identified and removed. At the output of the joiner, the complete object tracking information is collected and used to highlight the identified objects in the output video stream. Details of filters are depicted in Fig. 6(b).

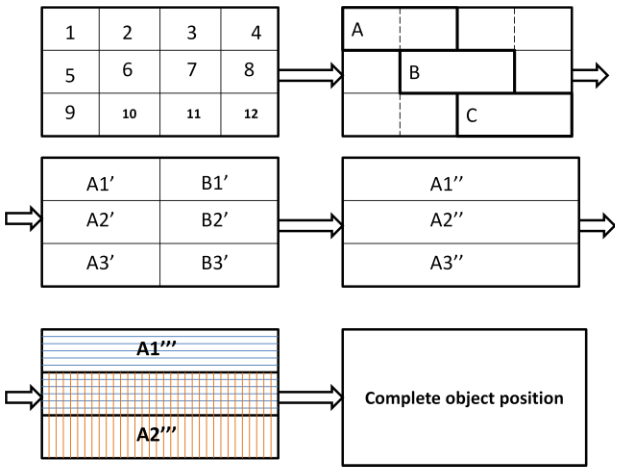
```
Single mode:
if(right_neighbor&down_neighbor&diag_neighbor)
   block_mode <=1;
   single_mode<=0;
else begin
   add_left_neighbor;
   add_right_neighbor;
   add_up_neighbor;
   add_down_neighbor;
end
 search_order <= search_order+1;

Block_mode:
  if(all_left_unit)
   add_left_unit;
 else if(~ all_left_unit)
   add_ones_in_left_unit;
   suspend_left <=1;
 if(all_right_unit)
   add_right_unit;
else if(~ all_right_unit)
   add_ones_in_right_unit;
   suspend_right<=1;
 if(all_up_unit)
    add_up_unit;
 else if(~ all_up_unit)
    add_ones_in_up_unit;
    suspend_up <=1;
  if(all_down_unit)
     add_down_unit;
 else if(~all_down_unit)
     add_ones_in_down_unit;
     suspend_down<=1;
 if(suspend_left || suspend_right || suspend_up ||suspend_down)
     single_mode<=1;
     block_mode<=0;
if(~suspend_left&&suspend_right&&suspend_up&&suspend_down)
    search_order<= search_order+1;
```

Figure 4. Single and Block mode search algorithms

V. EXPERIMENTAL RESULTS

The employed object tracking algorithm is based on Chesnokov Yuriy's [2] framework which achieves tracking rates between 0.5-35 fps on a 2.2 GHz processor. In this section, we compare the performance between the hardware implementation and the original software version. The algorithm executed in both implementations is equivalent in terms of functionality. The software implementation is



(a) Visual overview of five-stage boundary joiner

Filter1:
Merge (1,2); Merge (2,3); Merge (3,4);
Merge (5,6); Merge (6,7); Merge (7,8);
Merge (9,10); Merge(10,11); Merge(11,12)

Filter2 [1]*:*
Merge (A, B) in Row 1; Merge (B, C) in Row 1;
Merge (A, B) in Row 2; Merge (B, C) in Row 2;
Merge (A, B) in Row 3; Merge (B, C) in Row 3;

Filter3:
Merge (A1', B1');
Merge (A2', B2');
Merge (A3', B3');

Filter4:
Merge (A1", A2");
Merge (A2", A3");

Filter5:
Merge (A1"', A2"');

Note [1]: Even though A, B and C sub-frames are marked in different rows for clarity purposes, they exist in every row.

(b) Sub-frame merging in each stage of the boundary joiner

Figure 6. Boundary Joiner

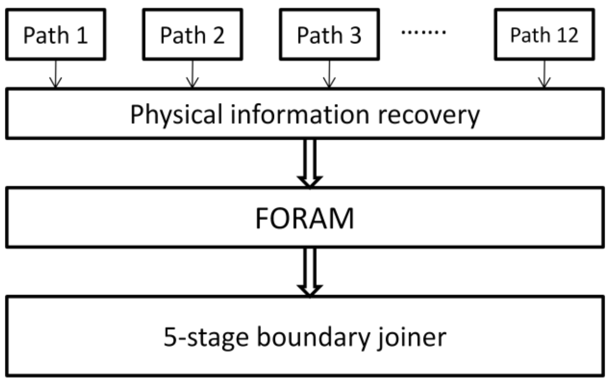


Figure 5. Frame-level foreground information composition

TABLE I. FPGA RESOURCE UTILIZATION SUMMARY

Resource	Used Resources	Utilization
Registers	32880	12%
LUTs	73794	22%
Block Memory	847 Kbits	5%

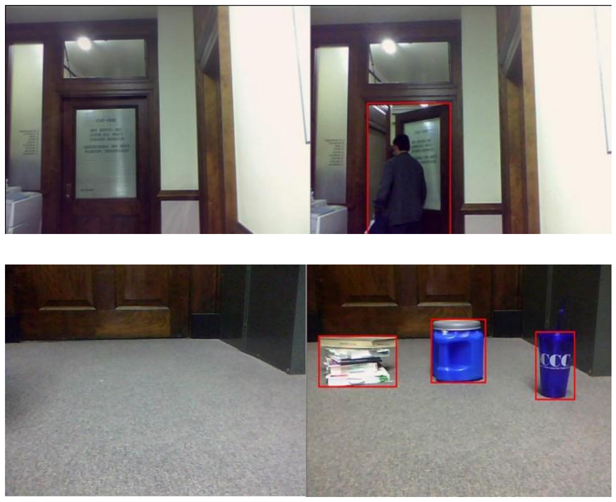


Figure 7. Object tracking testing

written in Visual C++ with SSE optimizations. Its performance is measured on an AMD Turion processor with 2.2 GHz frequency. The proposed hardware implementation is designed for the Altera DE3 board, which features the Stratix III (EP3SL340H1152C2) FPGA device, and it can run at a clock rate of 182.88MHz. Table I shows the hardware resource utilization on the FPGA device.

To evaluate the performance of the hardware implementation we compared different video inputs that contain frames with different number of objects. For each experiment the same video input was tested in both implementations. The second and third columns in Table II show the average processing time per frame and the corresponding throughput (in fps) for the software implementation. The corresponding hardware

TABLE II. PERFORMANCE COMPARISON: SW VERSUS HW

Object #	SW Exec. Time	SW fps	HW Exec. Time	HW. fps	Speedup
0	30ms	33.3	1.45ms	689.6	20.69X
1	79 ms	12.66	3.53 ms	283.3	22.38X
2	252 ms	3.97	3.87ms	258.4	65.12X
3	392ms	2.55	5.17ms	193.4	75.82X
4	546ms	1.83	5.27ms	189.7	103.6X
5	857ms	1.17	8.73ms	114.5	98.16X
6	1489 ms	0.67	14.4 ms	69.4	103.4X

implementation performance results are listed in the fourth and fifth columns of Table II. Finally the sixth column of Table II lists the speedup of the proposed FPGA implementation over the software version. We can see that the speedup of the FPGA-based tracker is higher for video inputs with higher number of objects. This further demonstrates the advantages of the proposed architecture in scenarios with video input that depicts dense traffic environments.

Fig. 7 depicts frames of the video output for two different examples. The left frame, in both examples, shows the background scene during the initial background training and the right frame shows the objects tracking result with highlighted object regions. The algorithm is able to identify both moving and static objects.

VI. CONCLUSIONS

In this work, we implemented an FPGA-based object tracking system which employs a background subtraction algorithm. The design was carried out using Verilog HDL and the implementation was based on the Altera DE3 development board. We studied and profiled the object tracking algorithm implemented in the software version and designed a highly-parallel architecture to achieve high throughput. We measured the hardware system performance through different experiments and observed more than 100X speedup compared to the software version for complex video inputs. As future work we plan to improve the sensitivity of the tracking algorithm to the luminance of the scene. This can be achieved by more accurate background training as well as using techniques based on hidden Markov models.

ACKNOWLEDGMENT

This work is partly supported by the Chinese Scholarship Council and GSRC (Gigascale Systems Research Center). The authors would like to thank Altera Inc. for the FPGA board donation. Meanwhile, the authors thank Sai Ma of University of Illinois for helpful discussions.

REFERENCES

[1] A. Yilmaz, O. Javed and M. Shah, “Object tracking: A survey,” *ACM Computing Surveys* (CSUR), Vol. 38, No. 4, 2006.

[2] C. Yuriy, “Real-time object tracker in C++,” http://www.codeproject.com/KB/audio-video/object_tracker.aspx, 2007.

[3] J. U. Cho, S. H. Jin, X. D. Pham, and J. W. Jeon, “Multiple Objects Tracking Circuit using Particle Filters with Multiple Features,” *Proc. IEEE Int’l Conference on Robotics and Automation*, 2007.

[4] U. Ali, M. B. Malik and K. Munawar, “FPGA/Soft-processor based real-time object tracking system”, *Proc. IEEE Southern Conference on Programmable Logic* (SPL’09), 2009.

[5] J. Xu , Y. Dou , J. Li, X. Zhou and Q. Dou, “FPGA Accelerating Algorithms of Active Shape Model in People Tracking Applications,” *Proc. 10th IEEE Euromicro Conference on Digital System Design Architectures, Methods and Tools* (DSD’07), 2007.

[6] S. Leephokhanon and T. Wiangtong, “Object Tracking and Motion Capturing in Hardware-Accelerated Multi-camera System,” *Proc. ACM Int’l Workshop on Reconfigurable computing: Architectures, Tools and Applications* (ARC’09), 2009.

[7] K. Mikolajczyk and C. Schmid, “A performance evaluation of local descriptors,” *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (CVPR’03), 2003.

[8] C. Wren, A. Azarbayejani, and A. Pentland, “Pfinder: Real-time tracking of the human body,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol 19, No 7, pp 780–785, 1997.

[9] H. Rowley, S. Baluja, and T. Andkanade, “Neural network-based face detection,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol 20, No 1, pp. 23–38, 1998.

[10] P. Viola, M. Jones, and D. Snow, “Detecting pedestrians using patterns of motion and appearance”, *Proc IEEE Int’l Conference on Computer Vision* (ICCV’03), pp 734–741, 2003.

[11] C. Papageorgiou, M. Oren, and T. Poggio, “A general framework for object detection”, *Proc. IEEE Int’l Conference on Computer Vision* (ICCV’98), pp 555–562, 1998.

[12] C. Veenman, M. Reinders, and E. Backer, “Resolving motion correspondence for densely moving points,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol 23, No 1, pp 54–72, 2001.

[13] P. Fieguth, and D. Terzopoulos, “Color-based tracking of heads and other mobile objects at video frame rates,” *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (CVPR’97), pp. 21–27, 1997.

[14] I. Haritaoglu, D. Harwood, and L. Davis, “W4: real-time surveillance of people and their activities,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 22, No 8, pp. 809–830, 2000.

[15] H. Tanizaki, “Non-gaussian state-space modeling of nonstationary time series,” *Journal of the American Statistical Association*, Vol. 82, pp. 1032–1063, 1987.