ABSTRACT

The Back over Prevention application has two different implementations namely DSP version and the (Texas Instruments’ Embedded Vision Engine) DSP + EVE version. This document describes the DSP + EVE version of BoP and different objectives of demonstrating this application.

EVE is a high-performance, low-power, programmable single instruction, multiple data (SIMD) accelerator with flexible data accesses. It is composed of a scalar RISC processor, called ARP32, and a vector co-processor, called VCOP. More details are available in EVE programmer’s guide [1]

CONTENTS

1 INTRODUCTION ........................................................................................................................................................................... 1
2 OBJECTIVES OF BOP DEMO ................................................................................................................................................................... 2
3 BOP ALGORITHM .................................................................................................................................................................................. 2
  3.1 Different Stages of BoP ................................................................................................................................................................... 4
  3.2 Functional mapping of BoP onto VisionMidEVE .................................................................................................................................. 6
4 MEETING THE SET OBJECTIVES ......................................................................................................................................................... 9
5 LIMITATIONS OF THE DEMO ............................................................................................................................................................. 10
References .......................................................................................................................................................................................... 11

1 Introduction

Texas Instruments has developed EVE as high performance, low-power architecture that impacts the embedded computer vision space by providing a programmable SIMD engine with flexible data accesses. Automotive vision systems are a key embedded environment where EVE’s architecture will excel. Automotive vision systems such as lane departure warning, back over prevention, and traffic sign recognition present a great potential to save human-life, and will become more prevalent in automobiles over the next decade.

Back over Prevention (BoP) is the prevention of running over a child or an object accidentally when a car is backing over. BoP installed in cars alert drivers of a possible blind spot hazard during backing over. Input is typically through a Fish Eye camera fitted to the rear bumper of the car. Output will be an alarm to alert the driver.
However in this demo, we test the algorithm using a pre-recorded video sequence from a Fish Eye camera fitted to rear bumper of a car.

The back over prevention application has two different implementations namely

- DSP version
- DSP + EVE version

The details DSP version of Backover Prevention and its integration in AV BIOS SDK is available in [2]

## 2 Objectives of the BoP demo

- Compare the performance of DSP and EVE with three kernels of BoP as an example
- Demonstrate
  - EVE in a video chain
  - Usage of EVE library API’s in an application
  - Usage of EVE Starterware API’s in an application
  - Partitioning an algorithm across two cores (DSP and EVE)
  - Usage of SYSBIOS IPC for communication between DSP and EVE
  - Graphics and Video Overlay
  - Out of Box Booting

## 3 Back over Prevention Algorithm

The algorithm is designed to detect the objects that would cause a potential hazard when a car is backing over. It also simulates an alert that will help the driver of the car to act and prevent such a hazard. In this section, details of the algorithm are presented wherein we briefly explain different stages of the algorithm as well as different implementations (DSP and EVE) of the algorithm.
3.1 Different Stages of BoP

The different stages of the BoP are shown in Figure 1. The pre-recorded video sequence from a Fish Eye camera is played using a DVD player. The deinterlaced input resolution is 720 x 480 YUV422. Since the algorithm operates on only the luma component of the input, neglecting the chroma component, luma (Y) extraction is performed as a pre-processing step. The three stages of BoP highlighted below in Figure 1 are mapped onto the EVE core, while the remaining stages are retained on DSP.

![Figure 1: Block Diagram of BoP](image)

3.1.1 Stage 0 → Lens Distortion Correction (Geometric Warp)

The lens distortion correction is a basic pre-processing step in any rear camera automotive use case. In many rear camera automotive use cases, a fish eye camera is fitted on the rear bumper of the car. The fish eye camera provides a large field of view for processing. The images captured using these cameras are severely distorted and hence unsuitable for vision processing. Figure 2 demonstrates the need for Lens Distortion Correction.
Figure 2: Lens Distortion Correction

The Lens Distortion Correction is available as a part of Texas Instruments’ software library Rear View Module (RVM) optimized for C6x DSP core. We make use of this RVM module in our BoP application.

**Input** → 720 x 480 Y

**Output** → 480 x 320 Y

### 3.1.2 Stage 1 → Location Matrix

In this stage, the input image is first scaled down to 240 x 160 followed by gradient computation in both horizontal and vertical condition. Based on the gradients, the orientation of each pixel is computed. The orientation angles are then stored in one of the nine bins located in one of the nine different matrices (L1, L2 ... L9) called as location matrices. Each location matrix has the same dimension as the sub sampled image (240 x 160). The location matrix is a binary matrix. Hence for example, if in the location matrix L5 there is a “1” at location (x,y), then it means that the image pixel at location (x,y) has an angle bin of “5”

**Input** → 480 x 320 Y

**Output** → Nine Location Matrices of dimension 240 x 160
3.1.3 Stage 2 ➔ Computation of Integral Images

In this stage, the integral images of each of the nine Location matrices L1 to L9 are computed. These nine integral images G1, G2 … G9 are needed to quickly compute histogram bins in the next stage (Stage 3).

**Input** ➔ Nine Location Matrices of dimension 240 x 160

**Output** ➔ Nine Integral Images of dimension 240 x 160

3.1.4 Stage 3 ➔ Histogram of Oriented Gradients (HoG)

The input frame is accessed along a grid of point’s spaced 20 pixels apart in the horizontal and vertical direction. A total of 44 grid point locations are performed in this algorithm. A sub-block at each grid point in the associated integral images G1 to G9 is accessed where we compute the Histogram of Oriented Gradients to build the feature list for that grid point. Non-contiguous accesses at grid-point locations from the lookup table accesses the integral image data because the grid point accesses are non-contiguous.

**Input** ➔ Nine Integral Images of dimension 240 x 160

**Output** ➔ Feature descriptor of size 44 x 2124

3.1.5 Stage 4 ➔ AdaBoost Classifier

The AdaBoost classifier is used in the current implementation of the algorithm as a binary classifier. The classifier outputs a “1” if an object of interest is detected or a “0” if the object of interest is NOT detected. The number of AdaBoost classifier used in the current classifier implementation is 51. The classifier accesses the feature list for each grid point to form a decision on it collectively whether the object of interest is present in the given image.

**Input** ➔ Feature Descriptors at 44 grid points

**Output** ➔ Decision “0” or “1”
3.2 Functional mapping of Back Over Prevention in AV BIOS SDK into processing cores of VisionMidEVE

The different stages of BoP algorithm was described in Section 3. Stages 1 through 4 comprises of the heart of the BoP algorithm, with Stage 0 being the important pre-processing step.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td></td>
</tr>
<tr>
<td>i.</td>
<td>DVD based capture (video sequence from a fish eye camera)</td>
</tr>
<tr>
<td><strong>Processing</strong></td>
<td></td>
</tr>
<tr>
<td>a.</td>
<td>De-interlace</td>
</tr>
<tr>
<td>b.</td>
<td>Y extraction</td>
</tr>
<tr>
<td>i.</td>
<td>DSP version using Cache</td>
</tr>
<tr>
<td>c.</td>
<td>Lens Distortion Correction (LDC) – <strong>Stage 0</strong></td>
</tr>
<tr>
<td>i.</td>
<td>DSP based LDC routine</td>
</tr>
<tr>
<td>d.</td>
<td><strong>EVE processing for Stages 1, 2 &amp; 3 of BoP</strong></td>
</tr>
<tr>
<td>e.</td>
<td>AdaBoost Classifier – Stage 4</td>
</tr>
<tr>
<td>f.</td>
<td>Graphics overlay</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td></td>
</tr>
<tr>
<td>a.</td>
<td>HDMI output</td>
</tr>
</tbody>
</table>

**Figure 3: Demonstration of mapping of different stages of BoP in AV BIOS SDK on to different processing cores of VisionMidEVE**
The partitioning of the BoP algorithm across DSP and EVE is demonstrated in Figure 3. The difference between DSP version and the DSP + EVE version of BoP is that the BoP algorithm stages namely the computation of Location matrix, Integral Images and HoG are ported to EVE. However the remaining stages are retained on the DSP. The reason for retaining the classifier stage on DSP is because; classification is a high level vision operation and is not EVE friendly. The lens distortion correction operation is also not EVE friendly and is not expected to give a significant speedup; hence this stage is also retained on DSP.

### 3.2.1 Process Flow

In the DSP + EVE version of BoP, the DSP is considered to be the master and EVE is acting as the slave. The DSP process Stage 0 of BoP for the current image frame and hands off the pointers for data arrays to EVE when EVE is ready to process. EVE then executes Stages 1, 2 and 3 and signals completion. While the EVE is processing these three stages, the DSP is in idle state. The DSP finally executes Stage 4 of BoP for the current frame. It has to be noted here that while DSP is processing Stage 0 or Stage 4, the EVE will be in idle state as well.

**Process Flow – DSP + EVE Version**

![Process Flow Diagram](image)

*Figure 4: Process Flow for the DSP + EVE Version of BoP*
3.2.2 BoP Application Performance on VisionMidEVE

The DSP is clocked at 500 MHz while the ARP32 is clocked at 200 MHz. The DSP loading for the DSP + EVE version of BoP is between 45 to 50%. Hence there is a reduction of 45 to 50% DSP CPU loading compared to the DSP version of BoP. This is the result of offloading BoP Stages 1, 2 and 3 to the EVE core. The EVE CPU loading is measured to be between 45 to 55%. This is a preliminary version of the demo. There is lot of scope for further improvement. Some important points to note here are:

1. Use EDMA for the Y extraction stage and Lens Distortion Correction stage is ABSENT
2. SYSBIOS IPC notification requires the BIOS OS running on ARP32
3. The EVE starterware mailbox communication does NOT require the BIOS on ARP32. However BIOS is running on DSP.

The demo is currently executing at 27 frames per second using the SYSBIOS IPC notification between DSP and EVE. The total roundtrip time of IPC notify including the EVE processing for stages 1, 2 and 3 is 17.31 msec. However using the EVE Starterware Mailbox communication between DSP and EVE, we are seeing a performance of 28 FPS. The total roundtrip time in this case is 19.19 msec.

3.2.3 Performance Comparison between DSP and DSP + EVE versions

It is evident from Figure 6 that there is a significant speedup in performance for Stages 1, 2 and 3 on EVE compared to execution of the same on C674x DSP.

<table>
<thead>
<tr>
<th>BoP Stage</th>
<th>DSP (Time in msec)</th>
<th>EVE (Time in msec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y Extraction</td>
<td>5.004</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Lens Distortion Correction</td>
<td>7.0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>BoP Stage 1</td>
<td>9.135</td>
<td>3.91</td>
<td>2.33</td>
</tr>
<tr>
<td>BoP Stage 2</td>
<td>19.0</td>
<td>2.25</td>
<td>8.44</td>
</tr>
<tr>
<td>BoP Stage 3</td>
<td>12.0</td>
<td>9.05</td>
<td>1.32</td>
</tr>
<tr>
<td>Classification</td>
<td>1.0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Figure 6: Performance comparison of BoP between DSP and EVE
Important Note: The three stages of BoP running on C674x DSP in the DSP version of BoP utilize cache for processing data present in external memory. Usage of EDMA is expected to provide improved performance for the DSP version of BoP.

4 Meeting the Set Objectives

4.1 Compare the performance of DSP and EVE with BoP as an example application

This was described in Section 3.2.3

4.2 Demonstrate

a. EVE in a video chain

The DSP + EVE version of Back over Prevention demo is integrated into the AV BIOS SDK framework where in the image frames obtained from the DVD player are processed one after the other. The demo is verified to run uninterrupted for more than 20 hours, thereby proving the stable video processing capabilities of EVE.

b. Usage of EVE library API’s in an application

Several API’s in the DSP + EVE version of BoP contain kernels either directly chosen from EVE library or variants of EVE library kernels. One such kernel that is reused without modification is the Integral Image kernel named “eve_integral_image”.

The kernel C code present in “evelib_00_03_02_00\IMGSIGLIB_kernels\Integral_image_VarBlk\src_kernelC\eve_integral_image_kernel.k” is linked in the CCS project to demonstrate the usage of EVE library API in an application development.

c. Usage of EVE Starterware API’s in an application

It is evident from Figure 4 that in the DSP + EVE version of BoP, the DSP and EVE communicate with each other very frequently. To achieve this inter processor communication, we make use of mailbox implementation between DSP and EVE demonstrated in the EVE Starterware package present at the location “evestarterware_00_02_00_01\examples\mailbox_eve1_to_gem1\”. The BIOS operating system is present on the DSP core, but absent on ARP32 core.
d. Partitioning an algorithm across two cores (DSP and EVE)

This is clearly evident in the DSP + EVE version of BoP where Stage 0 and Stage 4 is executed on the DSP while Stage 1, 2 and 3 are executed on EVE. This partitioning is demonstrated in Figures 4 and 5.

e. Usage of SYSBIOS IPC for communication between DSP and EVE

The interprocessor communication between DSP and EVE is also possible using the SYSBIOS IPC Notify. This is also demonstrated in the current release of BoP present in AV BIOS SDK 00.07.00. The BIOS operating system is present on both DSP as well as ARP32 cores.

f. Graphics and Video Overlay

The Graphics module present in HDVPSS has been utilized to simulate the alert to the driver. The Graphics module is a region based graphics processor that blends one or more graphics regions.

g. Out of Box Booting

The steps to perform Out of box booting are described in [3]. Loading the EVE executable into the EVE core is supported in this release.

5 Limitations of the Demo

i. This demo is NOT production quality, it is only for benchmarking purposes
ii. There is lot of scope of further improvement
iii. Tested on only one pre recorded video sequence
iv. Does NOT work on a live capture video
v. Y extraction is using cache, hence not completely optimized
vi. Lens Distortion Correction is also using cache, hence not completely optimized
vii. Processor level pipelining absent. The DSP is in idle state while EVE is processing and vice versa
viii. The first two stages of BoP running on EVE can be further optimized by merging these stages within a single DMA in and a single DMA out operation.
ix. The inter processor communication methods between DSP and EVE (both IPC SYSBIOS notify and EVE starterware mailbox) are NOT completely optimized
x. The Adaboost classifier present in the BoP algorithm is trained on very limited data set, preventing its usage on a generic video sequence for object detection
References

4. TI Automotive Rear View camera system
5. B.Kisacanin, “TI Automotive Backover Prevention”
   [http://www.youtube.com/watch?v=sgprCR1iE6Y&feature=youtu.be](http://www.youtube.com/watch?v=sgprCR1iE6Y&feature=youtu.be)