Application Report
SPRAAM6–June 2007

Video Background/Foreground Detection Implementation on TMS320C64/64x+ DSP

Cheng Peng, Ph.D.

ABSTRACT

Four existing methods of background/foreground (B/F) detection are implemented on the TMS320C64/64x+ DSP in this application report.

Contents

List of Figures

List of Tables

1 Introduction

Intelligent video content analysis (VCA) is getting more and more attention lately and many analog video surveillance systems are being replaced by digital solutions. The security industry has shown great interest in various VCA features such as classification, motion detection, and object tracking. These applications are helping to save human resources and significantly reduce false alarms.

Many VCA applications are expected to be ported from PC to DSP as real-time embedded solutions. The TI high performance TMS320C64/64x+ DSP family with VLIW architecture is very good at video processing, including VCA. B/F detection is a key signal processing module of a VCA application and requires very intensive pixel-wise computation. It is one of the important kernels that require thorough optimization for VCA implementations. A rich set of C64/C64x+ DSP devices offered by TI provides a wide variety of choice to VCA applications.

VCA was born out of the computer vision and pattern recognition (CVPR) area. This is an ongoing research area with no international standard currently available in this area. Therefore, it is a perfect application for TI DSP to focus on. Several VCA terms need to be clarified before moving on to algorithms details.

- Video object – An object captured in a sequence of video frames.
- Video scene/shot – A video sequence is composed of video scenes/shots. A video scene/shot is a group of continuous video frames with a stable background and a roughly fixed number of video objects.
- Feature of a video object – Spatial features (height-width-ratio, histogram, texture, shape) and temporal features (motion vector, speed, variation of object shape).

Usually, a surveillance camera monitors an area or several areas 24 hours a day. A VCA engine processes the captured video frames constantly and generates an alert when a security event happens. A general VCA algorithm flow is shown in Figure 1.

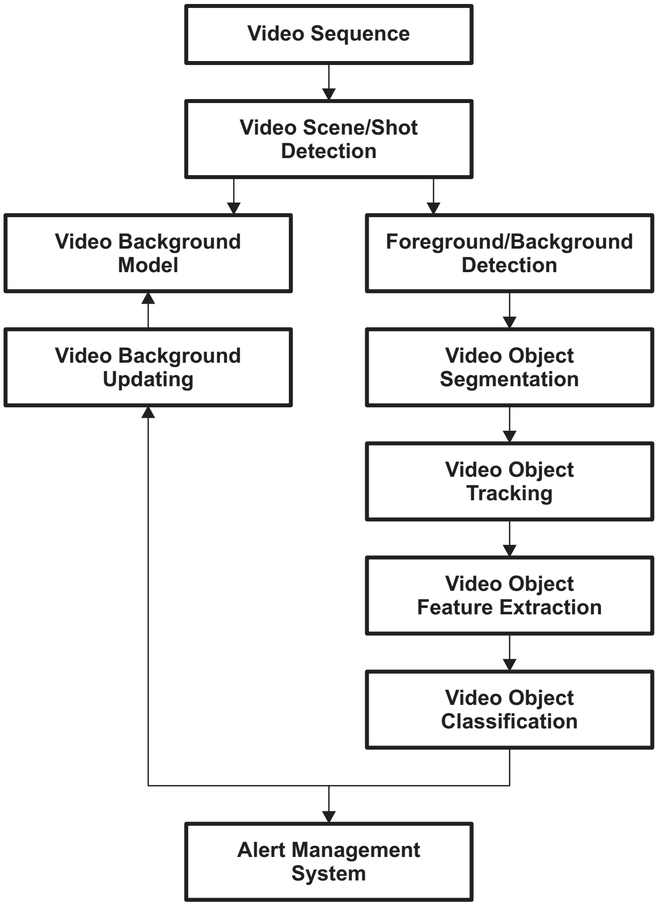

Video Sequence
Video Scene/Shot Detection
Video Background Model
Foreground/Background Detection
Video Background Updating
Video Object Segmentation
Video Object Tracking
Video Object Feature Extraction
Video Object Classification
Alert Management System

Figure 1. Video Content Analysis Block Diagram

When a new video frame is captured, it goes through a series of algorithm processing modules within a VCA engine.

- First, a video scene/shot detection algorithm decides if the current video frame belongs to an existing video scene/shot based on the frame histogram or the camera movement feedback.
- Second, a foreground/background detection module decides that every pixel in the current video frame belongs to the foreground or background based on a background model.
- Third, the B/F mask is morphologically dilated and grouped into a set of video objects with a bounding box. The location and speed of every video object is the foundation for the following object tracking. Kalman filtering is a well-known object tracking method.

A basic object classifier may only require height-width-ratio and speed as an object feature vector. A complicated object classifier usually depends on principle content analysis (PCA) or linear discriminate analysis (LDA) for a performance enhancement. All these video object features (height-weight-ratio, location and speed) are sent to the alert management module. Some of these algorithm modules are pixel-wise, the others operate at object level. Compared with the object-level algorithm modules, pixel-wise algorithm modules are a lot more intensive computationally because of high-pixel clock rate. B/F detection is one of the pixel-wise algorithm modules in VCA. It usually requires a thorough optimization in a VCA implementation. B/F detection is used as an example to show how to optimize a VCA algorithm on a C64/C64x+ DSP. This helps many VCA applications migrate to efficient DSP-based real-time implementations from expensive PC-based solutions.

2 Background/Foreground Detection

B/F detection plays a very important role in a video content analysis system. It is a foundation for various post-processing modules such as object tracking, recognition, and counting. Many approaches are proposed on this topic based on the background module and procedure used to maintain the model. In the past, the computational power of the processor limited the complexity of B/F detection implementation. This lead to a dilemma, some complicated implementations were too slow to catch up real-time requirements; the other basic implementations required a very controlled environment. Recently, faster computers and DSPs allow researchers to design a complicated B/F detection algorithm and meet real-time requirements. A real-world B/F detection should be robust and adaptive to different light conditions in a noisy environment.

There are two types of B/F detection methods: non-adaptive and adaptive. Non-adaptive methods depend on certain numbers of video frames and do not maintain a background model in the algorithm. Adaptive methods usually maintain a background model and the parameters of the background model evolve over time. Four well known B/F detection methods are introduced and implemented in this application report. These methods are implemented in the American National Standards Institute (ANSI) C (PC-flavor).

- Non-adaptive methods
  - B/F detection based on two frames
  - B/F detection based on three frames
- Adaptive methods
  - Adaptive B/F detection
  - Statistical B/F detection based on Gaussian model

2.1 Background/Foreground Detection Based on Two Frames

Two-frame-based B/F detection is the simplest non-adaptive method. First, a pixel-wise absolute difference is calculated between the current frame and the previous frame. Second, the absolute difference is compared with a given threshold value. If the absolute difference is greater than the threshold value, the corresponding pixel belongs to the foreground. Otherwise, it belongs to the background. The threshold value is chosen based on imager noise level and the complexity of the video sequence. Usually, this basic method is used for simple motion detection, not object tracking/recognition because the foreground objects can not be effectively extracted from the video sequence (See Figure 3). Sometimes, a pixel may be misclassified as foreground because of noise contribution.

The algorithm of two-frame-based B/F detection is described below.

- f_l : A pixel in a current frame, where *l* is the frame index.
- f_{l-1}: A pixel in a previous frame (f_l and f_{l-1} are located at the same location.)
- d_l: Absolute difference of f_l and f_{l-1}.
- b_l: B/F mask - 0: background. 0xff: foreground.
- T: Threshold value.

1. $d_l = |f_l - f_{l-1}|$
2. If $d_l > T$, f_l belongs to the foreground; otherwise, it belongs to the background.

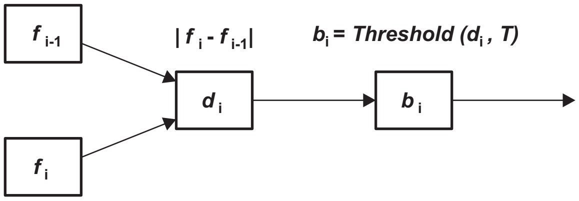
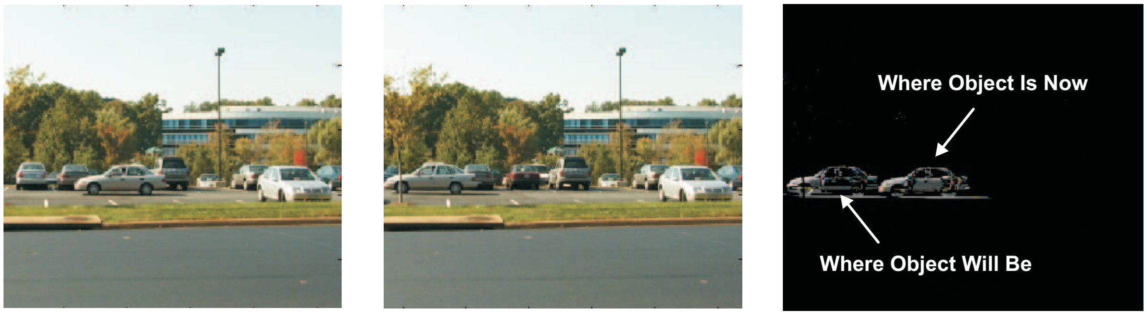


Figure 2. B/F Detection Based on Two Frames



(a) Previous Frame **(b) Current Frame** **(c) B/F Mask**

Figure 3. Examples of B/F Detection Based on Two Frames

A ghost object (where the object originally was) is shown as a foreground in Figure 3C.

The algorithm kernel loop is shown below. This is what the PC-based implementation looks like in general.

```
for (I = 0; I < size; I++)
{
  difference= abs(previousFrame[i]-currentFrame[i]);
  if(difference>threshold) bfMask[i]=0xff;
  else bfMask[i]=0;
}
```

When this kernel loop is ported to the C64/C64x+ DSP, the code structure and data access pattern need to be totally re-arranged in order to benefit the most from VLIW architecture and single-instruction, multiple data (SIMD) instructions. The optimized kernel loop is shown below.

```
threshold=threshold | threshold<<8 | threshold <<16 | threshold<<24;
// pack four threshold values into a integer
#pragma UNROLL(2);
for(i=0;i<size;i+=8)
{
  p0123=_lo(_memd8_const(&currentFrame[i]));
  p4567=_hi(_memd8_const(&currentFrame[i]));
  b0123=_lo(_memd8_const(&previousFrame[i]));
  b4567=_hi(_memd8_const(&previousFrame[i]));
  dif0123=_subabs4(p0123,b0123);
  dif4567=_subabs4(p4567,b4567);
  bit0123=_cmpgtu4(dif0123,threshold);
  bit4567=_cmpgtu4(dif4567,threshold);
  bitPos0123=_xpnd4(bit0123);
  bitPos4567=_xpnd4(bit4567);
  _memd8(&bfMask[i])=_itod(bitPos4567,bitPos0123);
}
```

In the optimized kernel loop, eight pixels are loaded to the dual registers first. It means that one register contains four pixels. Absolute differences of four pairs of pixels are calculated by using the _subabs4 instruction. The results of every four pixels are compared with the threshold value by using the _cmpgtu4 instruction. The output B/F bit-masks are expended to a byte-mask by using the _xpnd4 instruction.

2.2 Background/Foreground Detection Based on Three Frames

Three-frame-based B/F detection fixes the issue of *ghost objects* without increasing too much computational cost. First, a pixel-wise absolute difference is calculated between the current frame and the previous frame. Second, another pixel-wise absolute difference is calculated between the current and the next frame. Third, both absolute differences are compared to the given threshold value. If both of them are greater than the threshold value, the corresponding pixel belongs to the foreground. Otherwise, it belongs to the background. Three-frame-based method also reduces false foreground pixels because of noise contribution. This non-adaptive method can enable a short-term video object tracking/recognition in a controlled environment.

The algorithm of three-frame-based B/F detection is described below.

- f_{i-1} : A pixel in a previous frame, where *i* is the frame index.
- f_i : A pixel in a current frame
- f_{i+1} : A pixel in the next frame (f_i , f_{i-1} , and f_{i+1} are located at the same location.)
- d_i: A pixel-wise absolute difference between f_i and f_{i-1}.
- d_{i+1}: A pixel-wise absolute difference between f_i and f_{i+1}.
- b_i: B/F mask - 0: background. 0xff: foreground.
- T: Threshold

1. $d_i = |f_i - f_{i-1}|$ and $d_{i+1} = |f_i - f_{i+1}|$
2. If $d_i > T$ & $d_{i+1} > T$, f_i belongs to the foreground; otherwise, it belongs to the background.

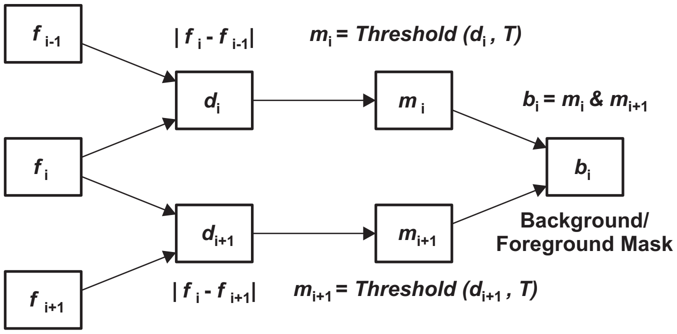
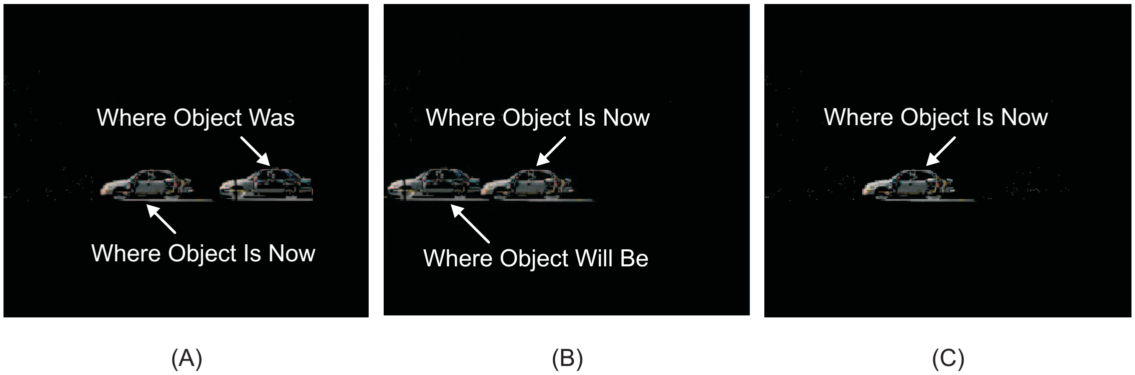


Figure 4. B/F Detection Based on Three Frames



A The B/F mask based on the current frame and the previous frame.

B The B/F mask based on the current frame and the next frame.

C The output B/F mask based on three frames.

Figure 5. Examples of B/F Detection Based on Three Frames

The Algorithm kernel loop is shown below. This is what the PC-based implementation looks like in general.

```
for (i = 0; i < size; i++)
{
  difference= abs(previousFrame[i]-currentFrame[i]);
  if(difference>threshold) bfFlag1=0xff;
  else bfFlag1=0;
  difference= abs(nextFrame[i]-currentFrame[i]);
  if(difference>threshold) bfFlag2=0xff;
  else bfFlag2=0;
  bfMask[i]=bfFlag1 & bfFlag2;
}
```

The optimized kernel loop on C64/C64x+ DSP is shown below.

```
threshold=threshold | threshold<<8 | threshold <<16 | threshold<<24;
for (i = 0; i < size; i+=8)
  {
    p0123=_lo(_memd8_const(&previousFrame[i]));
    p4567=_hi(_memd8_const(&previousFrame[i]));
    c0123=_lo(_memd8_const(&currentFrame[i]));
    c4567=_hi(_memd8_const(&currentFrame[i]));
    n0123=_lo(_memd8_const(&nextFrame[i]));
    n4567=_hi(_memd8_const(&nextFrame[i]));
    pcDif0123=_subabs4(p0123,c0123);
    pcDif4567=_subabs4(p4567,c4567);
    cnDif0123=_subabs4(c0123,n0123);
    cnDif4567=_subabs4(c4567,n4567);
```

```
   pcBit0123=_cmpgtu4(pcDif0123,threshold);
   pcBit4567=_cmpgtu4(pcDif4567,threshold);
   cnBit0123=_cmpgtu4(cnDif0123,threshold);
   cnBit4567=_cmpgtu4(cnDif4567,threshold);
   bitPos0123=pcBit0123 & cnBit0123;
   bitPos4567=pcBit4567 & cnBit4567;
   bitPos0123=_xpnd4(bitPos0123);
   bitPos4567=_xpnd4(bitPos4567);
   _memd8(&bfMask[i])=_itod(bitPos4567,bitPos0123);
}
```

2.3 Adaptive Background/Foreground Detection

Usually, the non-adaptive methods are useful only in high-supervised, short-term tracking applications without significant changes in the video scene. When errors happen, it requires manual re-initialization. Without re-initialization, errors in the background accumulate over time. Adaptive B/F detection is chosen for more and more VCA applications because of the limitation of non-adaptive methods. A standard adaptive B/F detection method maintains a background model within the system. Every new video frame is slowly blended into the background at a given learning rate. For every pixel, the absolute difference between the current frame and the background is calculated. If the result is greater than the given threshold value, the corresponding pixel belongs to the foreground. Otherwise, the corresponding pixel belongs to the background. This method is effective for many video surveillance scenarios where objects move continuously and the background is visible a significant portion of the time.

The algorithm of adaptive B/F detection is described below.

- f_i : A pixel in a current frame, where *i* is the frame index.
- μ : A pixel of the background model (f_i and μ are located at the same location).
- d_i: Absolute difference between f_i and μ.
- b_i: B/F mask - 0: background. 0xff: foreground.
- T: Threshold
- α: Learning rate of the background.

1. $d_i = |f_i - \mu|$
2. If $d_i > T$, f_i belongs to the foreground; otherwise, it belongs to the background.

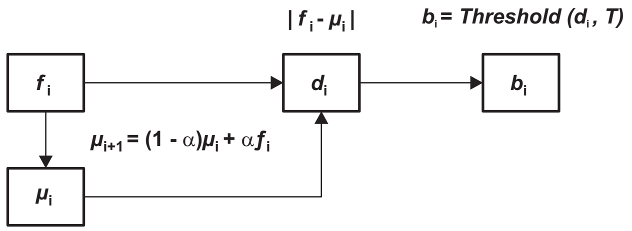

$|f_i - \mu_i|$
b_i = Threshold (d_i, T)
f_i
d_i
b_i
$\mu_{i+1} = (1 - \alpha)\mu_i + \alpha f_i$
μ_i

Figure 6. Adaptive B/F Detection

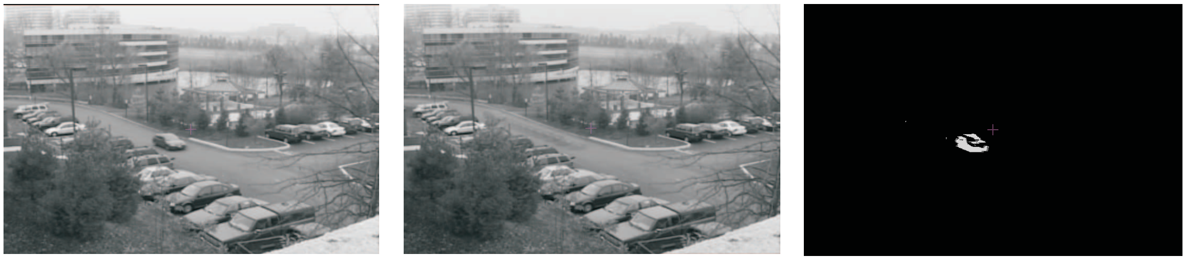
The algorithm kernel loop is shown below. This is what the PC-based implementation looks like in general.

```
 coefficient=255-learningRate;
 for (i = 0; i < size; i++)
{
  difference= abs(currentFrame[i]-background[i]);
  if(difference>threshold)
  bfMask[i]=0xff;
  else
  bfMask[i]=0;
  weightSum = (background[i]*coefficient+currentFrame[i]*learningRate)>>8;
  background[i]=weightSum;
}
```

The optimized kernel loop on C64/C64x+ DSP is shown below.

```
 threshold=threshold | threshold<<8 | threshold <<16 | threshold<<24;
 lr=learningRate | learningRate <<8 | learningRate<<16 | learningRate<<24;
 clr=_sub4(constantFF,lr);
 #pragma UNROLL(2);
for(i=0;i<size;i+=8)
{
 p0123=_lo(_memd8_const(&frame[i]));
 p4567=_hi(_memd8_const(&frame[i]));
 b0123=_lo(_memd8_const(&background[i]));
 b4567=_hi(_memd8_const(&background[i]));
 dif0123=_subabs4(p0123,b0123);
 dif4567=_subabs4(p4567,b4567);
 bit0123=_cmpgtu4(dif0123,threshold);
 bit4567=_cmpgtu4(dif4567,threshold);
 bitPos0123=_xpnd4(bit0123);
 bitPos4567=_xpnd4(bit4567);
 part1a = _mpyu4(b0123,clr);
 part1b = _mpyu4(p0123,lr);
 part2a = _mpyu4(b4567,clr);
 part2b = _mpyu4(p4567,lr);
 term1=_packh4(_hi(part1a), _lo(part1a));
 term2=_packh4(_hi(part1b), _lo(part1b));
 term3=_packh4(_hi(part2a), _lo(part2a));
 term4=_packh4(_hi(part2b), _lo(part2b));
 backgroundNext0123=_saddu4(term1,term2);
 backgroundNext4567=_saddu4(term3,term4);
 _memd8(&background[i])=_itod(backgroundNext4567, backgroundNext0123);
 _memd8(&bfMask[i])=_itod(bitPos4567,bitPos0123);
}
```

When the background is updated, every four pixels are processed together by using a set of SIMD instructions. First, αf_i and $(1 - \alpha)\mu_i$ are calculated for every four pixels using the _mpyu4 instruction. Second, the output terms are packed by using instruction _packh4. Third, the sum of αf_i and $(1 - \alpha)\mu_i$ is derived by using the _sadu4 instruction for every four pixels. Finally, the background model stored in the memory is updated. A captured video frame is shown in Figure 6A. The background is shown in Figure 6B. The output B/F mask is shown in Figure 6C.



(A) A Captured Frame

(B) A Background Model

(C) An Output Background/ Foreground Mask

Figure 7. Examples of Adaptive B/F Detection

2.4 Statistical Background/Foreground Detection Based on Gaussian Model

The most complicated B/F detection is based on a statistical background model. A background pixel in a given video frame is modeled as a random variable that follows the Gaussian distribution [1].

$$P_{i,k} \sim N\left(\mu_{i,k},\sigma^2_{i,k}\right) \quad (1)$$

$P_{i,k}$ is a pixel-wise random variable and follows the Gaussian distribution. It is located at the k^th^ position in the i^th^ video frame. $\mu_{i,k}$ and $\sigma_{i,k}$ are corresponding mean and standard deviation parameters of the Gaussian distribution.

Over time, a pixel is modeled as a time series called *pixel process*.

$$\left\{ P_{i,k} : P_{i,k} \sim N\left(\mu_{i,k}, \sigma^2_{i.k}\right) 0 \leq i < M, k = 0,1,2\ldots \right\} \tag{2}$$

M is the size of the image.

A pixel-wise Gaussian distribution is totally determined by its mean and standard deviation. The statistical property (mean and standard deviation) of a pixel process evolves over time based on live video data. For example, the standard deviation of a pixel located at the lake is usually bigger than the pixel located at the road (See Figure 7). The B/F detection based on this statistical background model can adapt to dynamic light changing and environmental noise. A pixel in a new video frame, with a pixel value within x (=2.5 or a given number) standard deviation of the corresponding Gaussian distribution, belongs to the background. Otherwise, it belongs to the foreground. If a pixel belongs to the background, the corresponding mean and standard deviation are updated at a given learning rate linearly.

Basically, a background can be seen as a collection of pixel-wise means per frame at a given time. Every background pixel has its own threshold value derived from the corresponding standard deviation. This is indeed a per-pixel/per-distribution thresholding method. It is very effective when different regions have different lighting conditions or different noise levels. A uniform threshold may result in objects disappearing when they enter a low-noise region.

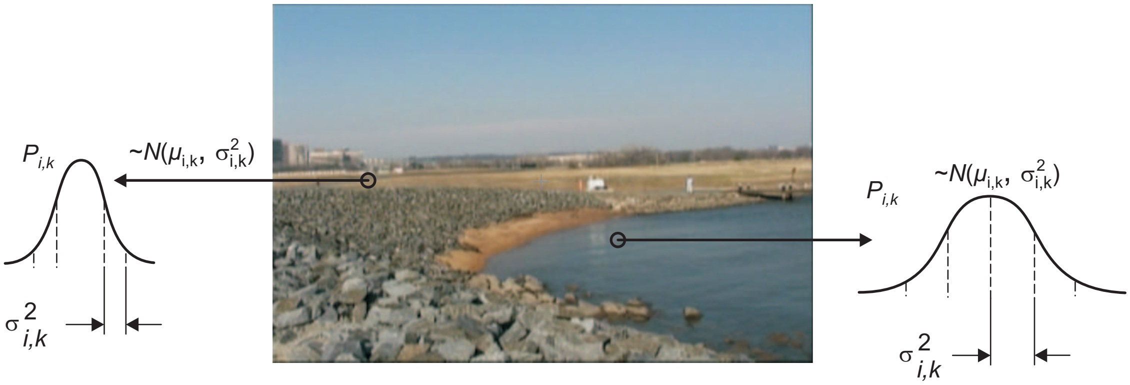

$P_{i,k}$
$\sim N(\mu_{i,k}, \sigma^2_{i,k})$
$\sigma^2_{i,k}$
$P_{i,k}$
$\sim N(\mu_{i,k}, \sigma^2_{i,k})$
$\sigma^2_{i,k}$

Figure 8. Pixel Process Following Gaussian Distribution

The algorithm details are described below.

- f_i : A pixel in a current frame, (the i^{th} video frame).
- μ_i :Mean of a pixel-wise background Gaussian distribution. (f_i and μ_i are located at the same location.)
- σ_i: Standard deviation of a pixel-wise background Gaussian distribution.
- d_i: Absolute difference between f_i and μ_i.
- T_i: A pixel-wise threshold.
- α: Learning rate of the background.
- η: Threshold gain.

1. $d_i = |f_i - \mu_i|$
2. $T_i = \eta\sigma_i$
3. If $d_i > T$, f_i belongs to the foreground; otherwise, it belongs to the background.
4. If f_i belongs the background, update the pixel-wise mean and standard deviation of the corresponding pixel distribution at a given learning rate: $\mu_i = (1 - \alpha)\ \mu_i + \alpha f_i$ and $\sigma_i = (1 - \alpha)\sigma_i + \alpha d_i$.

The linear blending/interpolation method used to update pixel-wise background mean and standard deviation is effective and affordable.

The block diagram of the algorithm is shown in Figure 9.

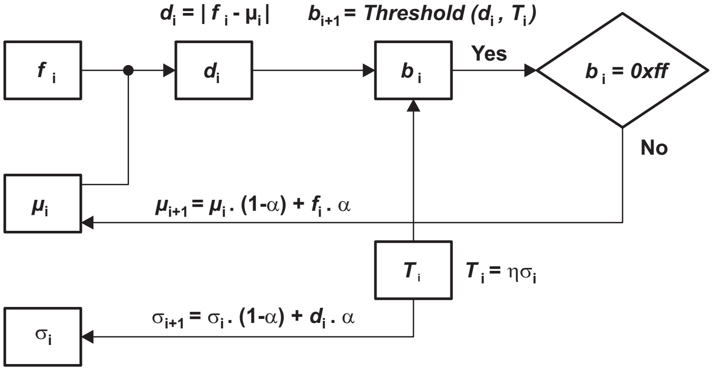

$d_i = | f_i - \mu_i |$
$b_{i+1} = Threshold\ (d_i, T_i)$
f_i
d_i
b_i
Yes
$b_i = 0xff$
No
μ_i
$\mu_{i+1} = \mu_i . (1-\alpha) + f_i . \alpha$
T_i
$T_i = \eta\sigma_i$
σ_i
$\sigma_{i+1} = \sigma_i . (1-\alpha) + d_i . \alpha$

Figure 9. A Statistical B/F Detection Based on the Gaussian Model

A simplified statistical B/F detection algorithm is shown below.

```
coefficientMean=255-learningRateMean;
coefficientVariance=255-learningRateVariance;
for (i = 0; i < size; i++)
{
  difference= abs(currentFrame[i]-backgroundMean[i]);
  if(backgroundVariance[i]<varianceThreshold)
     threshold=varianceThreshold;
  else
     threshold= backgroundVariance[i]*thresholdGain;
  if(difference>threshold)
  bfMask[i]=0xff;
  else
  {
    bfMask[i]=0;
    weightSum =
    (backgroundMean[i]*coefficientMean+currentFrame[i]*learningRateMean)>>8;
    backgroundMean[i]=weightSum;
    weightSum =
    (backgroundVariance[i]*coefficientVariance+difference*learningRateVariance)>>8;
    backgroundVariance[i]=weightSum;
  }
}
```

The optimized kernel on the C64/C64x+ DSP is shown below.

```
  gain=gain | gain<<8 | gain <<16 | gain<<24;
  threshold1=varianceThreshold | varianceThreshold<<8 | varianceThreshold <<16 |
  varianceThreshold<<24;
  lrMean=learningRateMean | learningRateMean <<8 | learningRateMean<<16 |
  learningRateMean<<24;
  clrMean=_sub4(constantFF,lrMean);
 // #pragma UNROLL(2);
 for(i=0;i<size;i+=8)
{
 p0123=_lo(_memd8_const(&currentFrame[i]));
 p4567=_hi(_memd8_const(&currentFrame[i]));
 b0123=_lo(_memd8_const(&backgroundMean[i]));
 b4567=_hi(_memd8_const(&backgroundMean[i]));
 v0123=_lo(_memd8_const(&backgroundVariance[i]));
 v4567=_hi(_memd8_const(&backgroundVariance[i]));

 temp0123=_mpyu4(v0123,gain);
 threshold2=_packl4(_hi(temp0123), _lo(temp0123));
 bit0123=_cmpgtu4(threshold2,threshold1);
 bitPos0123=_xpnd4(bit0123);
 cBitPos0123=~bitPos0123;
 term1=bitPos0123 & threshold2;
```

```
   term2=cBitPos0123 & threshold1;
   threshold0123=_saddu4(term1,term2);


   temp4567=_mpyu4(v4567,gain);
   threshold2a=_packh4(_hi(temp4567), _lo(temp4567));
   bit4567=_cmpgtu4(threshold2a,threshold1);
   bitPos4567=_xpnd4(bit4567);
   cBitPos4567=~bitPos4567;
   term3=bitPos4567 & threshold2a;
   term4=cBitPos4567 & threshold1;
   threshold4567=_saddu4(term3,term4);

   dif0123=_subabs4(p0123,b0123);
   dif4567=_subabs4(p4567,b4567);
   bit0123=_cmpgtu4(dif0123,threshold0123);
   bit4567=_cmpgtu4(dif4567,threshold4567);
   bitPos0123=_xpnd4(bit0123);
   bitPos4567=_xpnd4(bit4567);
   cBitPos0123=~bitPos0123;
   cBitPos4567=~bitPos4567;
   part1a = _mpyu4(b0123,clrMean);
   part1b = _mpyu4(p0123,lrMean);
   part2a = _mpyu4(b4567,clrMean);
   part2b = _mpyu4(p4567,lrMean);

   term1=_packh4(_hi(part1a), _lo(part1a));
   term2=_packh4(_hi(part1b), _lo(part1b));
   term3=_packh4(_hi(part2a), _lo(part2a));
   term4=_packh4(_hi(part2b), _lo(part2b));
   backgroundNext0123=_saddu4(term1,term2);
   backgroundNext4567=_saddu4(term3,term4);

   part1a = _mpyu4(v0123,clrVariance);
   part1b = _mpyu4(dif0123,lrVariance);
   part2a = _mpyu4(v4567,clrVariance);
   part2b = _mpyu4(dif4567,lrVariance);
   term1=_packh4(_hi(part1a), _lo(part1a));
   term2=_packh4(_hi(part1b), _lo(part1b));
   term3=_packh4(_hi(part2a), _lo(part2a));
   term4=_packh4(_hi(part2b), _lo(part2b));
   varianceNext0123=_saddu4(term1,term2);
   varianceNext4567=_saddu4(term3,term4);

   term1=bitPos0123 & b0123;
   term2=cBitPos0123 & backgroundNext0123;
   backgroundNext0123=_saddu4(term1,term2);

   term3=bitPos4567 & b4567;
   term4=cBitPos4567 & backgroundNext4567;
   backgroundNext4567=_saddu4(term3,term4);

   term1=bitPos0123 & v0123;
   term2=cBitPos0123 & varianceNext0123;
   varianceNext0123=_saddu4(term1,term2);

   term3=bitPos4567 & v4567;
   term4=cBitPos4567 & varianceNext4567;
   varianceNext4567=_saddu4(term3,term4);

   _memd8(&backgroundMean[i])=_itod(backgroundNext4567, backgroundNext0123);
   _memd8(&backgroundVariance[i])=_itod(varianceNext4567, varianceNext4567);
   _memd8(&bfMask[i])=_itod(bitPos4567,bitPos0123);
  }
```

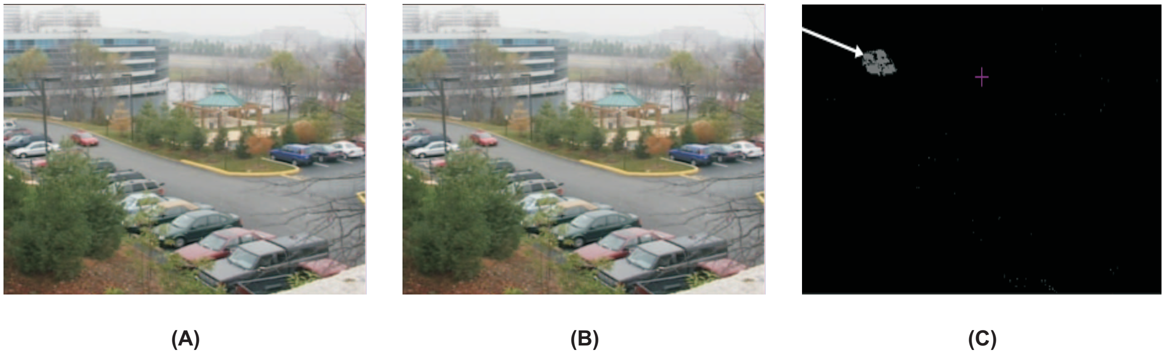
In order to use the SIMD instruction efficiently, delete the conditional structure of step 4 in the following equation.

$$
\begin{aligned}
\mu^{+}_{i+1} &= (1-\alpha)\mu_i + \alpha f_i \\
\sigma^{+}_{i+1} &= (1-\alpha)\sigma_i + \alpha d_i \\
\mu_{i+1} &= (1-b_i)\ \&\ \mu^{+}_{i+1} + b_i\ \&\ \mu_i \\
\sigma_{i+1} &= (1-b_i)\ \&\ \sigma_i + b_i\ \&\ \sigma_i
\end{aligned}
\tag{3}
$$

B/F mask, b_i , is 0xff for the background and 0 for the foreground. For every four pixels, the B/F masks are gotten first and then packed into a 32-bit integer by using the _cmptu4 and _xpnd4 instructions. Second, $(1\text{-}b_i)$ is calculated. The _saddu4 instruction is used to generate the new $\{\mu_{i+1}, \sigma_{i+1}\}$ for every four pixels together.

3 Experiments and Results

Two 720×480 security video sequences are used to benchmark the algorithms. The performance data is collected in the Code Composer Studio™ software C64x+ cycle accuracy simulator. First, the results of all methods are shown. Second, special application scenarios are listed for all methods. Finally, the algorithm cycle consumptions on C64x+ DSP are presented.

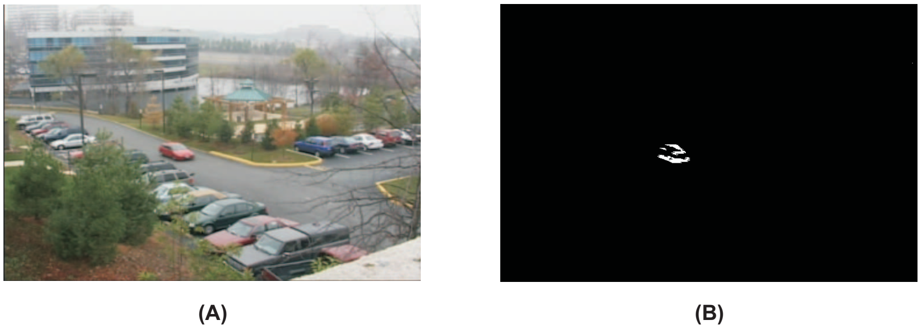


A A captured video frame.

B The output B/F mask based on two-frame method.

C The output B/F mask based on three-frame-method.

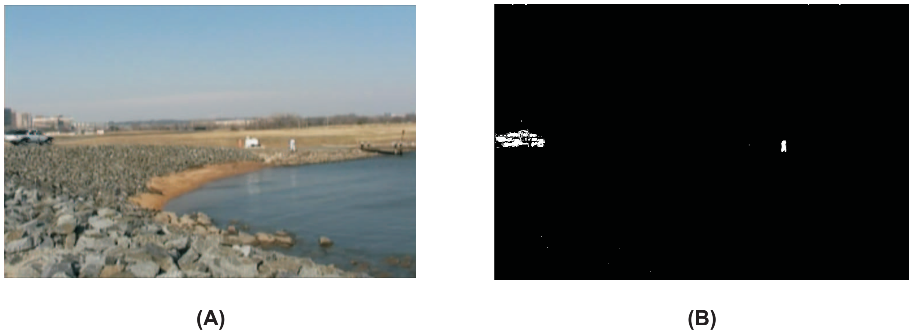
Figure 10. The Results of Non-Adaptive B/F Detections



A A captured video frame.

B The output B/F mask.

Figure 11. The Results of Adaptive B/F Detections



A A captured video frame.

B The output B/F mask.

Figure 12. The Result of Statistical B/F Detection Based on Gaussian Model

The different methods of B/F detection are described in Table 1.

Table 1. Application Scenarios of Different B/F Detection Methods

B/F Detection	Application Scenarios
B/F detection based on two frames	Motion detection
B/F detection based on three frames	A short-term object tracking/recognition in a controlled environment.
Adaptive B/F detection	A long-term surveillance in a noiseless environment.
Statistical B/F detection based on Gaussian model	A long-term surveillance in a dynamic lighting condition and noisy environment.

The cycle consumption for various algorithms is shown in Table 2.

Table 2. C64/C64x+ DSP Cycle Consumption for Various Algorithms[1]

Cycles/Pixel	B/F Detection 2f	B/F Detection 3f	Adaptive B/F Detection	Statistical B/F Detection
Natural C implementation	3	4	5	6
Optimized implementation on C64/C64x+ DSP	0.38	0.5	0.5	1.13
QVGA 10 fps	291840	384000	384000	867840
D1 10 fps	1313280	1728000	1728000	3905300

[1] Flat memory model

We observed that the MHz consumption of all methods are reduced significantly after optimization in Table 2. These achieve very efficient B/F detection on the C64/C64x+ DSP.

4 Conclusion

Four existing methods of B/F detection are implemented on the C64/C64x+ DSP in this application report and achieve very efficient MHz consumption. Non-adaptive methods are useful for short-term object tracking/recognition in a controlled environment. Adaptive B/F detection is useful for long-term surveillance in a noiseless environment. Adaptive statistical method can handle long-term surveillance in a dynamic lighting condition and noisy environment. B/F detection is an important module of VCA. The optimized kernels are key to reducing the overall VCA application MHz consumption. The optimization technique used for B/F detection is helpful for the remaining algorithm modules in VCA.

5 References

1. C. Stauffer and W Grimson. *Adaptive Background Mixture Models for Real-Time Tracking*, Proc IEEE Conference Computer Vision and Pattern Recognition, 1999.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
Low Power Wireless	www.ti.com/lpw	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265