GMM FOREGROUND SEGMENTATION PROCESSOR BASED ON ADDRESS FREE PIXEL STREAMS

Ryo Yagi, Tomohito Kajimoto and Takao Nishitani

Tokyo Metropolitan University

ABSTRACT

A compact implementation of a foreground segmentation processor in a multi-resolution transform domain has been proposed for HDTV signals. The proposed architecture is designed to simplify system controls by the hardware streaming and to reduce required memory capacities. It enables flowing pixels through all functional units in order, including multi-resolution spatial transform and temporal segmentation. The resultant architecture does not use memories except I/O buffers. Therefore, memory modules as well as complex address manipulation over the multiple global transforms and spatial/temporal interface is not required. The FPGA prototype chip dissipates 150 mW of power. This approach can be used for tablets and smart-phone by an ASIC implementation which will reduce the operation power to about 1/6.

Index Terms— custom DSP processor, multi-resolution transform, Gaussian Mixture Model, FPGA prototype, streaming

1. INTRODUCTION

A foreground segmentation based on a Gaussian Mixture Model (GMM) of pixels [1] has a robust nature to dynamic backgrounds. However, one feature processing by GMM requires many GMM parameters: every component Gaussian requires a weighting coefficient in addition to its mean and variance. When 3 component Gaussians are employed, 9 parameters are required to find out foreground objects and to maintain the background. The conventional approach uses one set of features from every R, G, B component of a pixel [1]. Then, GMM parameters become 27 per pixel and occupy 27 frame memories for a video signal. In addition, this approach is not so stable for HDTV videos, due to the pixel independent processing. In addition, there are some implementation trials by FPGA, but these implementations employ simplified algorithms to reduce arithmetic operations [2]. Another way to implement the above approach seems to employ GPGPU which can provide multiple floating point units and a large capacity memory in the host microprocessors. High power consumption is resulted, but still usable for surveillance purposes.

The transform domain GMM approach [3], which utilizes multiple resolution transforms over the luminance components followed by GMM, produces quite stable segmentation over HDTV videos with only 10% of the conventional processing resources. This reduction comes from the fact that only low band features are used. In the same ratio of the processing reduction, the required memory capacity for GMM parameters also decreases. Therefore, this approach is attractive for realizing a compact system for mobile applications of gait recognition and action recognition which require non-rigid shape foreground. This is because recent mobile terminals are becoming popular to introduce high resolution cameras. However, GPGPU cannot be employed, due to their huge power consumption of more than 100 W. The power dissipation can be reduced to only 10% in average by a FPGA implementation instead of GPGPU [4]. Further power reduction is possible to 10% when ASIC converted from the FPGA design is employed [5].

The approach proposed here is based on the GMM algorithm in multi-resolution transform domain. However, the straight forward ASIC implementation of the algorithm is not efficient in terms of complexity. The algorithm requires iterative operations of feature extraction from every resolution transform followed by a temporal GMM processing. The multi-resolution processing is realized by a set of transforms whose source area sizes are 2x2 times larger or smaller in each other. Therefore, the multi-resolution transformation should be carried out by a fast algorithm for the operation reduction. Then, every stage of the fast algorithm becomes spectra on a resolution and they are stored into an internal memory for the next stage transforms. The internal memory plays a role of storing interim spectra for further transform and also of converting the spatial processing to the temporal processing in every multi-resolution transform domain. However, it requires storages for storing spectra in every stage for a fast algorithm.

The architecture described here is concerning about simplification of the above straight forward implementation. The proposed architecture makes pixels flow through all the processing in a single pixel-stream by replacing internal memories with a small amount of registers. Once a pixel data is fetched from the input buffer, the related processing on the pixel is carried out continuously as much as possible. This is possible because only low sequency features on Walsh transform (WT) is used for the GMM segmentation. The used nature of WT is that the WT spectra in a block can be produced from a couple of spectra sets for two half size blocks in one dimensional case. Therefore, buffer memories, which store interim spectra of all small blocks in the largest block, are not required for multi-resolution transform. The pixel flow order can be determined by specially designed register files in the multi-resolution transform processor.

In the paper, the transformed domain foreground segmentation is first reviewed. In the following sections, the proposed pixel flow, processor architectures and the resulting performance are described.

2. FOREGROUND SEGMENTATION ALGORITHM IN TRANSFORM DOMAIN

The overall processing of the foreground segmentation in WT domain [3] can be depicted in Fig. 1 in the following page, where a multi-resolution WT parameter processor (WPP) and the GMM

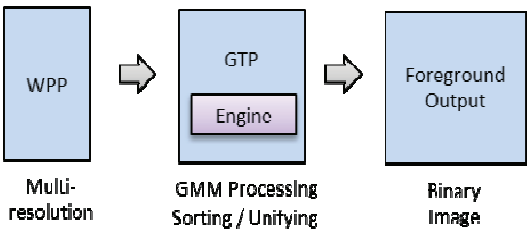

Fig.1. Functional block diagram of FPGA processor.

thread processor (GTP) are depicted. In WPP, a HDTV picture is first divided into (4 x 4) pixel blocks and every block is transformed by WT. After that, the original blocks are merged to a larger blocks by combining (2 x 2) adjacent blocks and the block is transformed again. This process is iteratively carried out until the transform of a (64 x 64) pixel block is completed. In every transform, three WT parameters are calculated in the following way.

$$X_{DC} = |W_{00}|, \qquad (1)$$
$$X_{ACH} = \sum_{i=1}^{3} i^2 |W(0,i)|, \quad X_{ACV} = \sum_{j=1}^{3} j^2 |W(j,0)|,$$

where $|W_{ij}|$ shows the absolute value of Walsh spectrum at the (i, j) coordinate. These features are, then, sent to GTP.

In GTP, every WT feature is processed by the GMM of every background block, having the probability density function of

$$\sum_{i=1}^{3} \omega_i N_i(\mu_i, \sigma_i^2), \qquad (2)$$

where $N_i(\mu_i, \sigma^2_i)$ is the i-th Gaussian with the mean μ_i and variance σ^2_i, and ω_i is the i-th Gaussian weighting coefficient. Every WT feature value is checked whether it is included in a range of a component Gaussian or not. When a Gaussian covers the WT feature value in (1), the Gaussian parameters of the mean and variance are updated by using the input WT feature values. The weighting coefficient value also increases. Otherwise, Gaussian parameters of the mean and variance are kept unchanged, but the weighting parameter value decreases. When no component Gaussian is found, the least weight component Gaussian is replaced with a new component Gaussian having the WT parameter value as μ_i and a relatively large σ^2_i as initial values.

After these updates, the weighting coefficients from three Gaussians in every WT feature are sent to a Sort-Engine in GTP, where the sorting process is carried out in accordance with the weighting coefficients. This process segregates the component Gaussians into background or foreground classes. Gaussians having bigger weights are classified to the backgrounds. When the WT feature value is covered with a foreground Gaussian, the block becomes foreground. This information is sent to the output buffer where a foreground binary image is generated by combining the results of different resolution decisions.

3. PIXEL FLOW

The design effort has been paid to eliminate internal memory units, including interface between WPP and GTP. This is because the employment of internal memory units requires complex address control as well as memory hardware. Therefore, once a pixel is fetched from the input buffer, all processing from the input to the output is carried out just like a stream of pixels. There are two restrictions on the internal memory elimination between WPP and the GTP. The first one is to realize continuous processing between the spatial transform and the temporal GMM, and the second one is to regulate the data transfer rate from the transform to the GMM so as not to overrun the GMM processing.

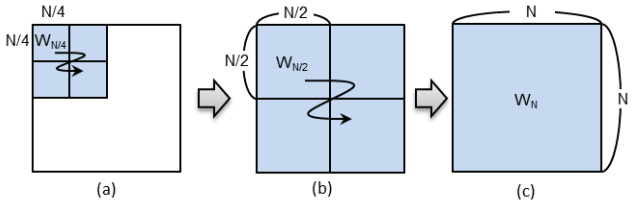

Fig.2. Walsh Transform Expansion.

The continuous processing of the transform and the GMM can be realized by using the WT spectra property. A set of four ($N/2$ x $N/2$) spectrum blocks can be efficiently merged into the (N x N) spectrum block by only additions and subtractions. This approach is known as the Fast Walsh Transform (FWT) [6], as shown in Figs. 2 (a), (b), (c). The spectra in enlargement block are calculated by using a 2D butterfly operator, similar to that of FFT. Note that the enlargement process in spectra among adjacent blocks is not realized in FFT, due to the phase term existence in the basis functions. FFT can realize wider source area spectra from a set of small source area spectra, but the small source areas have to be selected to have interlaced relationship in the original source area.

Another Walsh spectrum property on the WT features is that the spectrum components used in (1) requires only three components of W_{00}, W_{01}, W_{10} in every ($N/2$ x $N/2$) block. This implies that every enlargement process from 4 transform blocks to a 4 times bigger block transform can be simplified and have the same processing steps.

The processing order of WT blocks in WPP is arranged so that the first (4 x 4) WT introduces three adjacent (4 x 4) WTs in the location of the right hand side, that of the down side and that in the diagonally down side, depicted in Fig. 2(a). After these four (4 x 4) WTs, the processing of the (8 x 8) block transform, shown in Fig. 2(b), is started to execute before other (4 x 4) WT calculations. Similarly, the 4 times larger block size WT is started immediately when 4 adjacent quarter size blocks are completed. Therefore, the feature parameter processing order can be formulated by using operators in the following way.

$$P_4 = WT_2 \bullet WT_2 \bullet WT_2 \bullet WT_2 \bullet (BF \bullet FP_4),$$
$$P_8 = P_4 \bullet P_4 \bullet P_4 \bullet P_4 \bullet (BF \bullet FP_8), \qquad (3)$$
$$P_{16} = P_8 \bullet P_8 \bullet P_8 \bullet P_8 \bullet (BF \bullet FP_{16}),$$
$$P_{32} = P_{16} \bullet P_{16} \bullet P_{16} \bullet P_{16} \bullet (BF \bullet FP_{32}),$$
$$P_{64} = P_{32} \bullet P_{32} \bullet P_{32} \bullet P_{32} \bullet (BF \bullet FP_{64}),$$

where P_4 shows the process of generating a set of (4 x 4) WT features, WT_2 is the (2 x 2) WT operator, $\bullet$ is a concatenation operator. Operators BF and FP_4 in the parenthesis at the end of the line show that the WT feature calculator FP_4 is required after the butterfly operator of BF. Other capital letters in the following lines show the same operators in P_4, but the different suffixes show different block sizes.

Equation 3 also shows quite interesting property for the second restriction which asks not to overwrite an interface register between WPP and GTP. A (4 x 4) WT feature set is sent to GTP after the period of 4 WT_2 operations plus the period of the feature

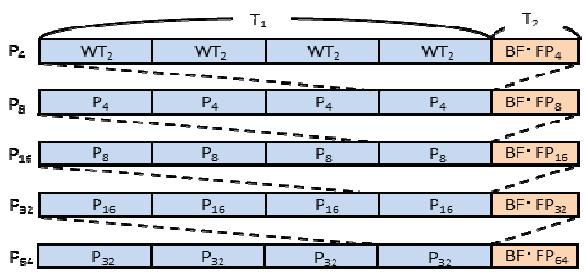

Fig. 3. Structure of Equation 3.

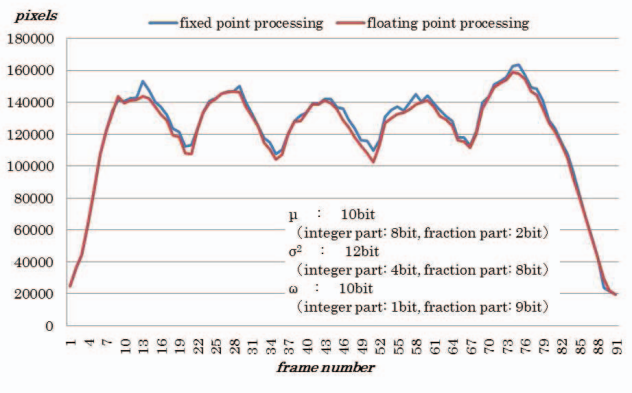

Fig.4. Foreground pixel difference between floating-point and 16/32 bit fixed processing on a video scene.

calculator (FP) with a block enlargement processing (BF). This P_4 processing periodically appears in the second equations in (3). The additional *BF* and *FP* operations, terminating every P_i (i=8, 16, 32, 64), disturbs the periodical data transfer. The following consideration is to realize a single interface register connection.

Let T_1 be the time period of 4 WT_2 in P_4 and let T_2 be that of BF and FP processing at the last term in P_4, shown in Fig. 3. Also, let T_3 be a time period of GTP processing. Then, T_3 should be equal or less than (T_1+T_2) for real time processing. However, at the last procedure in every equation of (3), GTP has to be activated every T_2 time period after T_1. The busiest period of GTP appears at the end of P_{64}, where the consecutive five GTP activations occur. Therefore, the condition of a single register interface without overwriting can be summarized in the following 2 cases: (1) $nT_2 < (n-1)\,T_3$ where n is less than 5 and the left term shows the timing of register input from WPP and the right term shows the end of $(n-1)$ GTP processing or the n-th data fetch timing point from the register during n consecutive activations, (2) 5 GTP processing can be executed during $4T_2+(T_1+T_2)$, where (T_1+T_2) is the vacant time period of the first *P4* processing in the next (64x64) block. Precise consideration will be described later.

4. FIXED-POINT ARCHITECTURE

For mobile application purpose, the fixed point processing as well as small amount of internal memory is mandatory for reducing power consumption. Computer simulation shows that 32/16 bit processing is reasonable selection, as shown in Fig. 4, where foreground pixels are calculate at every frame by using floating-point calculation and by using fixed point calculation. In addition, the set of three Gaussian feature parameters are unified into a 32 bit word for the purpose of I/O bandwidth reduction from an external

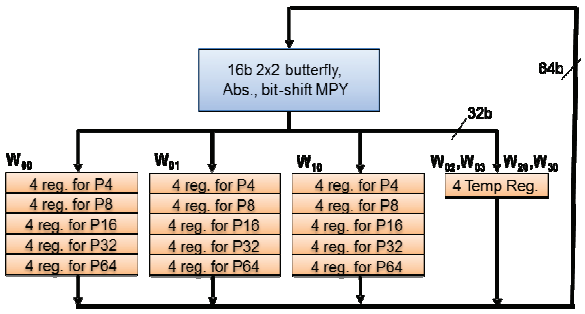

Fig.5. WPP structure.

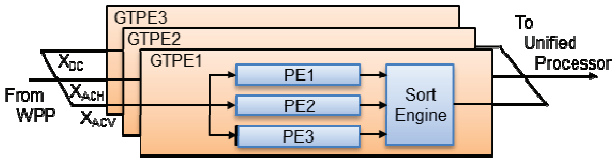

Fig.6. GTP architecture.

parameter memory having one frame, 32 bit word capacity. The fixed point calculation generates only slightly higher false foreground pixels.

The employed WPP and GTP processing is carried out every (64 x 64) input pixel block transferred from the external frame memory. The (64 x 64) block is the maximum block size for the multi-resolution transform. In order to process HDTV (1920 x 1024 pixels in 30 frames per second) in real time, the processing time for a (64 x 64) input block should be within 69.44 μsec.

Figure 5 shows the architecture of WPP, having a form of a tandem connection of two sets of two 16 bit ALU's so as to realize a WT butterfly in a pipelined form. In order to realize efficient processing in the tightly connected ALUs, four operands can be fed into the arithmetic units through a 64 bit buss. Specially designed 16 bit register files and the input buffer outputs are connected to this bus. Two pipeline cycles produce one pair of spectra on a (2 x 2) WT. In addition, the tightly connected ALUs can calculate two absolute values at a time and integer multiplications by using bit-shift operations for calculating WT features, shown in (1). The arithmetic unit performance can process all transforms and WT feature generations in real time, when WPP works at 100 MHz.

The function for realizing processing without internal memory exists in specially designed controller and register files in WPP. The order of fetching pixels from the input buffer is accomplished by using the zigzag scan for the WT block enlargement order, shown in Fig. 2. It can be easily realized by setting the address register for the input buffer by a binary counter where the even bit positions are shifted to the upper half positions for the column address and the odd bit positions are shifted to the lower half positions for the row address.

In order to realize a larger block layer processing, the calculated spectrum results are stored in the register files, having an autonomous control in a register selection in the file. Three files are employed and named as W_{00}, W_{01} and W_{10}, for holding the corresponding spectrum. Every register file consists of 20 registers, having a structure of 5 layers of 4 registers for the operation of the right hand side terms in (3), except the operation of the last term. The autonomous register selection control is also simple. When 4

registers in a same block layer become full, a register in the next layer registers is selected to store. The contents of smaller layer register contents are abandoned.

The temporal registers hold W_{02}, W_{20}, W_{03} and W_{30} for calculating WT parameters in (1). Whenever the spectra in an enlarged block are calculated, These coefficients are used only once for WT feature calculations in WPP.

On the other hand, GTP consists of 3 sets of Gaussian Tread Processing Elements (GTPE), as shown in Fig. 5. The 3 sets of GTPE come from the employed 3 WT features. In every GTPE, 3 homogeneous Processing Elements (PEs) are used for 3 component Gaussians, where every PE checks the inclusion of the WT feature value in the component Gaussian and updates its weighting coefficient as well as Gaussian Parameters of their means and variances, depending on the checked results. The final decision of foreground or background and the adjustment of weighting coefficients have to be carried out in the sort engine, where the checked results of 3 Gaussians from PEs are combined to give the final block result.

Every PE is composed of ALU, a multiplier, another ALU and a register in a pipeline form. The process to check the WT feature is carried out in the variance domain by using the first ALU followed by the multiplier. The third ALU has a function of storing flags. This is used in the parameter update processing and it enables replacing conditional branch operations with non-conditional ones for some modifications of update order. Therefore, the processing of GMM inclusion and parameter update functions can be achieved by keeping the same processing steps. This enables using a SIMD control for these three PEs. The sort engine is composed of multipliers for normalizing weight coefficients and adder trees for sorting these results.

All of the estimated GMM parameters inside a (64 x 64) block are fetched from an external memory, synchronized with the (64 x 64) input pixel data, as described in section 3. Due to the pixel flow approach, precise address information to control the external memory is not required. This is because such parameters should appear exactly after every frame rate. Therefore, the external memory for GMM parameters should have a form of a ring buffer. The starting point for the ring buffer at the beginning of processing does not affect the processing results.

5. FPGA IMPLEMENTATION RESULTS

The proposed foreground segmentation core has been implemented by FPGA. The WPP processing efficiency reaches more than 95 % when the clock frequency is set to 100 MHz. The hardware elements are summarized in Table 1, including the input/output buffers of pixels and GMM parameters. The BRAM in the table 1 are used for these buffers. The power dissipation is evaluated by X-power for Xilinx Vertex4-XC4VLX200, and summarized in Table 2. The power dissipation of the IP core part, excluding the leakage power and I/O buffer power which are consumed by FPGA itself, dissipates only around 152 mW. As the Verilog net list of has been obtained, further power reductionto around 20% is possible by Full ASIC design.

The straight forward approach described in section 3, which employs the iterative processing of a single resolution transform completion followed by GMM processing is considered here. This configuration needs a 2D buffer between the transform and GMM. The first (2 x 2) WT layer processing can be executed by a single ALU processor, when the processor runs at the 110 MHz clock

Table 1. Hardware Resources.

	Flip-Flop	LUTs	BRAM	DSP48
Device	2730	5452	17	18

Table 2. Power Dissipation.

	IP-Core	IOs	Leakage	Total
Power (Watt)	0.152	0.034	1.343	1.529

frequency. In order to process the following (4 x 4) WT in a pipeline manner in terms of the 2x2 WT layer, a double buffer structure has to be introduced. Also, the buffer has to store W_{00}, W_{01} and W_{10} for preparation of the (4 x 4) WT with extended dynamic range. About four times bigger buffer capacity than that of the input buffer is required. From the (4 x 4) WT processing, the output buffer has to store 7 spectra used in (1) for every block. As a result, from the (4 x 4) block layer to the last (64 x 64) block layer, a single processor can execute all the transform in real time. However CPU efficiency drops to 70%, if the address calculation is carried out by other processing resources. In this pipeline stage, WT features should be calculated from the (4 x 4) block layer to the (64 x 64) block layer by another processor. As a result, the straight forward approach needs a lot of memory module inside the processing stage with complex address computations.

The following consideration is on the connection between WPP and GTP, in terms of processing efficiency of GTP. The implementation result shows the time period ratio of T_1 to T_2, described in section 2, becomes 10 to 12 in WPP. Based on this figure, the evaluation of the lowest GTP processing efficiency of maximizing GTP processing time is calculated. It becomes 61% for the single register employment between WPP and GTP. When an additional register is inserted for the buffer register purpose, it becomes 72%. Our system is designed to have around 45% activity in average and to sleeps until WPP wakes up. Therefore, the single retiming register employment is enough for our purpose.

6. CONCLUSION

Foreground segmentation in multi-resolution transform domains for HDTV pictures have been proposed and implemented by FPGA. The architecture is based on a newly introduced pixel flow structure. The power consumption of 153 mW excluding leakage power and I/O buffer seems to be applicable for mobile applications when it is implemented by a custom ASIC approach.

REFERENCES

[1] C. Stauffer, et al., “Adaptive background mixture models for real-time tracking,” Proc. CVPR’99, pp.246-252, 1999.

[2] K. Appiah, and A. Hunter, “A single-chip FPGA implementation of real-time adaptive background model,” ICFPT, pp. 95-102, December 2005.

[3] H. Tezuka , et al., "Multiresolutional Gaussian Mixture Model for Precise and Stable Foreground Segmentation in Transform Domain", IEICE Trans. Fund., Vol. E92-A, pp. 772-778 , 2009.

[4] A. Papakonstantinou, et al., “ Multilevel Granularity Parallelism Synthesis on FPGAs”, FCCM’11, pp.178-185, 2011.

[5] I. Kuon, et al., “Measuring the gap between FPGAs and ASICs,” FPFA’06, pp.21-30, 2006.

[6] J. W. Manz, “A sequency-ordered fast walsh transform,” IEEE Trans. Audio and Electroacoustics, vol. AU-20, pp.204-205, 1972.