International Journal of Electronics
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713599654

FPGA-based architecture for motion sequence extraction

Online Publication Date: 01 May 2007
To cite this Article: Díaz, J., Ros, E., Mota, S. and Rodriguez-Gomez, R. , (2007) 'FPGA-based architecture for motion sequence extraction', International Journal of Electronics, 94:5, 435 - 450
To link to this article: DOI: 10.1080/00207210701294908
URL: http://dx.doi.org/10.1080/00207210701294908

International Journal of Electronics,
Vol. 94, No. 5, May 2007, 435–450

FPGA-based architecture for motion sequence extraction

J. DÍAZ*†, E. ROS†, S. MOTA‡ and R. RODRIGUEZ-GOMEZ†

†Dep. Arquitectura y Tecnología de Computadores,
Universidad de Granada, Spain
‡Dep. Informática y Análisis Numérico,
Universidad de Córdoba, Spain

(*Received 12 May 2006; in final form 4 December 2006*)

The estimation of motion from image sequences has been widely studied by the scientific community but it is rarely used in real-time applications mainly due to the high computational requirements. A large number of interesting applications (such as robotics, vigilance, sequence compression, etc.) require embedded processing systems which are not yet available. The presented approach implements a novel superpipelined and fully parallelized architecture for optical flow processing with more than 70 pipelined stages that achieve a data throughput of one pixel per clock cycle. The whole system has been implemented into reconfigurable technology to facilitate its adaptation to different application specifications. It achieves high performance computation (148 frames per second at VGA resolution). In this contribution we justify the optical flow model chosen for the implementation, we analyse the presented architecture, and measure the system resource requirements. In particular, we present a massive parallelism design methodology that makes these high performance systems possible. Finally, we evaluate the system comparing its performance with other previous approaches. To the best of our knowledge, the obtained performance is more than one magnitude higher than any previous real-time approach described in the literature.

Keywords: Pipeline architecture; Real-time; FPGAs; Image motion processing; Optical flow

1. Introduction

Optical flow is a well known research field used to recover 2-D motion from image sequences. There are different approaches based on image block-matching, gradient constraints, phase conservation or energy models (Barron *et al*. 1994). Until now, most of the comparative studies focused on the different estimation approaches and their accuracies (Barron *et al*. 1994, McCane *et al*. 2001). Nevertheless, some of them also covered the implementation feasibility (Liu *et al*. 1998). They show that best accuracy is achieved by using phase-based and differential methods but, even though these models work fine for low motion velocities, they fail when trying to estimate fast motion (their accuracies are significantly degraded due to the temporal aliasing) (Weber and Malik 1995, Lim *et al*. 2005). The temporal aliasing problem is a

*Corresponding author. Email: jdiaz@atc.ugr.es

International Journal of Electronics
ISSN 0020–7217 print/ISSN 1362–3060 online © 2007 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/00207210701294908

complex topic where we cannot isolate the temporal sampling rate and the image structure. The spatial frequency has significant importance to calculate the maximum speed which can be recovered from an image sequence according to the Nyquist–Shannon theorem (Weber and Malik 1995, Lim *et al*. 2005).

Our approach focuses on the utilization of digital cameras of high frame-rate acquisition as a valid alternative to reduce temporal aliasing. Advances in imaging sensor technology make it possible to acquire more than 1000 frames per second (fps) (see products from: http://www.coreco.com, http://www.hitachi-service.net/, http://www.ims-chips.com) prompting us to develop processing architectures running at higher frame-rates than standard video at 30 fps. Although the 1000 fps is still far away from our processing capabilities, an over-sampled factor of 4 or 5 dramatically reduces the motion aliasing presented in most common scenarios. The utilization of high frame-rate cameras reduces the motion range presented at the video sequence, allowing gradient models to achieve an outstanding accuracy.

In previous works, Lucas and Kanade (L&K) gradient based method (Lucas and Kanade 1984, Barron *et al*. 1994) is highlighted as a good candidate to be implemented on hardware with affordable hardware resources consumption (Liu *et al*. 1998, McCane *et al*. 2001, Díaz *et al*. 2004, 2006b). The comparison of L&K with other differential approaches (Bainbridge-Smith and Lane 1996, 1997), (also of feasible hardware implementation as indicated in Maya-Rueda and Arias-Estrada (2003)), concludes that the L&K least-squares fitting approach achieves the best accuracy.

In this work, we describe a novel superpipelined processing architecture capable of computing one pixel per clock cycle. This architecture significantly improves our previous works (Díaz *et al*. 2004, 2006b) thanks to the new fine-grain pipeline, a novel memory management unit which enables the utilization of FIR temporal filters and an improved image differentiation technique. It allows real-time processing of oversampled frame-rates, which opens the door to use the advanced image sensors to achieve high accuracy of optical flow. In the next sections, we describe in detail the computing architecture, we evaluate its performance and system resources utilization, and we compare our results with other previous approaches.

2. High accuracy optical flow model description

On the previous discussion, we have presented the L&K model as a good candidate for real-time optical flow computation. The L&K algorithm belongs to gradient-based techniques, which means that the estimation of pixel velocities is based on image derivatives, and the assumption of constant luminance over a temporal window is required. The velocities V_x and V_y are computed using a convolution with separable kernels operating as discrete derivatives. We note I_t to the temporal derivative of the sequence and I_x and I_y to the spatial image derivatives along columns and rows. L&K method constructs a flow estimation based on these first-order derivatives of the image. Using least-square fitting, the model extracts an estimation of pixel motion based on the hypothesis of velocity similarity on spatial neighbourhoods Ω of each pixel, typically of 5×5 pixels, for details see Lucas and Kanade (1984) and Barron *et al*. (1994). Using this notation, the final

velocity estimation is computed from equation (1), where W stands for the weights used to properly integrate estimations over the neighbourhood Ω (which typically represent a five taps Gaussian kernel)

$$\left.\begin{aligned} V_x &= A_{xy} \cdot A_{yt} - A_{yy} \cdot A_{xt} \\ V_y &= A_{xy} \cdot A_{xt} - A_{xx} \cdot A_{yt} \\ A_{kj} &= \sum_{\Omega} W^2 I_k I_j \end{aligned}\right\} \quad (1)$$

Most of works in the literature use the derivative kernels and model parameters presented in Barron *et al.* (1994) but, as described in Brandt (1997), they are susceptible of significant improvement. For instance, Brandt proposes derivative kernels based on complementary derivation-smoothing operations that significantly improve the derivation process accuracy. Furthermore, this leads to reducing the temporal pre-smoothing filter length from 15 to 3 taps, which is of significant importance for embedded systems and also play an important role in terms of accuracy. This allows computing higher spectral frequencies because they are not filtered. In that work, he also analysed different neighbourhood Ω windows. These modifications lead to an improvement of 31.3% for a similar density of estimations with respect to the version given in Barron *et al.* (1994) (see § 3.2 for more details).

To summarize, the processing stages including the modifications proposed in Brandt (1997), are the following.

1. Pre-filtering with a separable kernel of $3 \times 3 \times 3$, $P = [1, 2, 1]/4$. The utilization of this small smoothing kernel allows high optical flow estimation density, as it does not reject the high frequency terms and at the same time contributes as anti-aliasing filter too.
2. Complementary derivative kernels (2-D smoothing and 1-D derivation for each axis derivative) as designed by Simoncelli (1994). These kernels increase the architecture complexity compared with previous approaches (Díaz *et al.* 2004, 2006b), but significantly improve the accuracy of the system (Brandt 1997). In terms of performance, they represent a computation load increment of a factor of 3 but this is not a problem when designing customized hardware, due to the fact that it can be implemented in the pipeline structure without throughput degradation.
3. The image derivatives I_x, I_y and I_t (subscript stands for axis direction derivative) are cross-multiplied to get the five products $I_x \cdot I_x$, $I_y \cdot I_y$, $I_x \cdot I_y$, $I_x \cdot I_t$, and $I_y \cdot I_t$ and then, they are locally weighted on a neighborhood area Ω. The weighting operation is implemented as separable convolution operations over the derivatives products using the 2-D spatial central-weighting separable kernel [0.0625, 0.25, 0.375, 0.25, 0.0625].
4. Finally, the weighted image derivatives products are combined to get each pixel velocity estimation (Barron *et al.* 1994).

The overall support of the system is $11 \times 11 \times 7$ pixels using the parameters described above. Thus, just a 7 images storage is required, which is feasible on systems embedded on a single chip. In a previous implementation of the L&K model

(Díaz *et al*. 2004, 2006b), we used the Fleet and Langley (1995) IIR temporal filter, which requires just 3 images storage. The drawback of such an approach is that IIR filters produce lower accuracy in the estimated optical flow and need higher fixed-point bit-width to compute filter values (see § 3.2 for more details).

We have numerically evaluated the accuracy of the different approaches using the synthetic sequence of the through-flow across the Yosemite Valley (sequence available at: ftp://ftp.vislist.com/SHAREWARE/CODE/OPTICAL-FLOW/). Using a flow density of 36.4% and the error measurement of Fleet (1992), the angular error is 4.6° using the implementation described on Barron *et al*. (1994), 6.4° using the approach of Fleet and Langley (1995), and 3.4° using the modifications of Brandt (1997). We conclude from these measures that the modifications of L&K model adopted by Brandt (1997) improve the accuracy and therefore, this is the option whose implementation in specific hardware is presented in this paper.

3. Hardware architecture considerations

Nowadays, standard PC processors have significant computing performance thanks to the high system clock frequency and to the MMX and SSE instruction extensions, which give them DSP capabilities. Nevertheless, although there are some optical flow approaches running on software in near real-time (Bruhn *et al*. 2005), the intensive computation required to process optical flow makes it non-viable to process oversampled sequences in real-time and the large incoming data needs that specific hardware be processed on real-time. DSP and multimedia processors are specifically suitable for embedded high-load processing (Dumontier *et al*. 1999), but their computing performances are still far from allowing real-time computing of optical flow at more than 30 fps. The high performance required makes DSP unviable. A feasible solution is the implementation of a custom ASIC, but this alternative is quite expensive for prototyping purposes and limits its adaptation to different application specifications. Because of that, we focus on a FPGA-based approach. This is a technology that is being used in very different research fields, such as mobile robotics (Bonato *et al*. 2006), network intrusion detection (Clark *et al*. 2006), cryptography (Kerins *et al*. 2006), etc. They are able to achieve significant computing power. Furthermore, their reconfiguration capability allows correcting the design "if the specifications change". Furthermore, since FPGAs are based on general purpose digital technology (following Moore's law), systems developed for FPGAs benefit from the continuous technology advances, avoiding the possibility of becoming obsolete within short periods of time. Therefore, we have chosen reconfigurable hardware as our target technology.

3.1 *Coarse grain pipeline structure*

We use a pipelined architecture (figure 1). The basic computational stages represent the different steps of L&K algorithm briefly described on § 2. The system has been designed as an embedded processing platform which can be used on mobile applications and thus, user interface, hardware controller for memory, VGA visualization, and input camera interface have been embedded on the

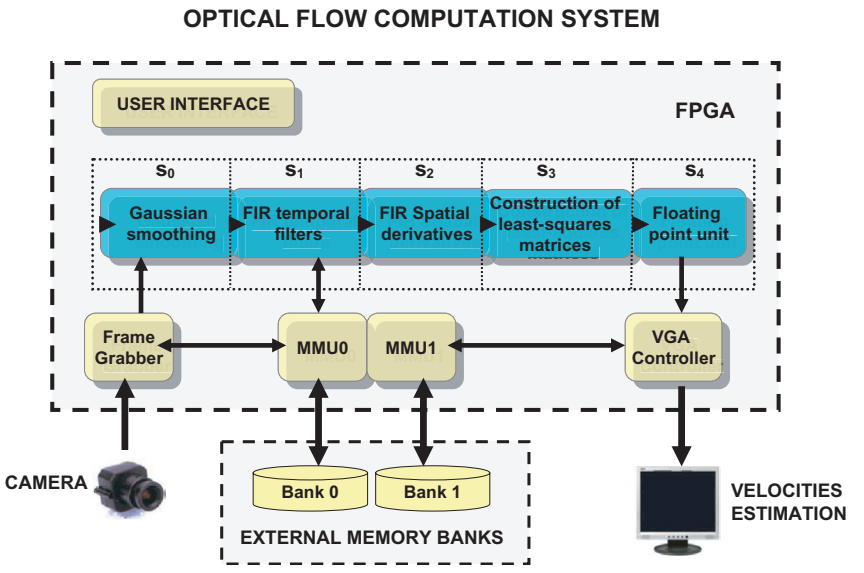

Figure 1. Optical flow system structure. Thin dotted line marks the processing core. Light-color blocks indicate hardware controllers inside the FPGA and external memories. The user interface consists of an LCD display plus mode-selection buttons. All the computation has been accomplished inside the FPGA device.

same FPGA device, a preliminary version has been presented in Díaz *et al.* (2006a). This strategy enables the utilization of the system combined with a high frame-rate camera as a smart sensor on diverse potential applications. The different elements that form the system are represented on figure 1. Note that the thin dotted line marks the optical flow, processing core the stages which are summarized as follows.

- S_0. Gaussian-filter smoothing stage.
- S_1. The FIR temporal filter computes the temporal derivative and space-time smoothed images.
- S_2. Spatial derivatives and complementary Gaussian filtering operations.
- S_3. Construction of least-square matrices for integration of neighbourhood velocity estimations.
- S_4. Custom floating-point unit. Final velocity estimation requires the computation of a matrix inversion, which includes a division operation. At this stage, the resolution of the incoming data bits is significant and expensive arithmetic operations are required. Thus, fixed-point arithmetic becomes unaffordable, prompting us to design a customized floating-point unit.

3.2 *IIR vs. FIR temporal filters in a hardware implementation*

The approach described by Fleet and Langley (1995) requires the storage of 3 processed intermediate images. In this way, the temporal storage requirement is reduced to 3 frames. The temporal filter can be computed as follows.

- Let us consider a separable space-time smoothing filter. After the spatial filtering operation, we can use a causal temporal filter based on a truncated exponential.

$$E(t) = \begin{cases} \exp(-t/\tau)/\tau & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{2}$$

  where τ is the time constant of the filter. The temporal derivative can be calculated using this filter.
- The digital filter equations described in Fleet and Langley (1995) are the following:

$$\left.\begin{aligned} w(t) &= I(t) - 2rw(t-1) - r^2 w(t-2) \\ R_2(t) &= q^2 w(t) + 2q^2 w(t-1) + qw(t-2) \\ y(t) &= R_2(t) - ry(t-1) \end{aligned}\right\} \tag{3}$$

  In equation (3), $I(t)$ stands for the *image* at instant t, q and r are constant parameters that depend on τ according to equation (4). Finally, $w(t)$, $R_2(t)$ and $y(t)$ stand for intermediate processed images at instant t used in the final computation (equation (5)). In our system, we store and update: $w(t)$, $w(t-1)$, $y(t)$.
- The parameters q and r are calculated from τ according to equation (4):

$$q = \frac{1}{1+2\tau} \quad r = \frac{1-2\tau}{1+2\tau} \tag{4}$$

- Finally, the smoothed temporal image and its derivative are computed with equation (5), where I_{smooth} and I_t stands for the temporally smoothed image and its temporal derivative

$$\left.\begin{aligned} I_{\text{smooth}} &= qy(t) + qy(t-1) \\ I_t(t) &= \frac{R_2(t) - I_{\text{smooth}}(t)}{\tau} \end{aligned}\right\} \tag{5}$$

We have implemented this approach in Diaz *et al.* (2006a) with typical τ values of 0.5, 1, 2 or 3. Intermediate images were computed using fixed-point representation of 12 bits and 18 bits for q and r parameters This allows packing three pixels in each memory address because we use 36 bit words to match the memory bit-width (IDT 2006). As shown in the last part of § 2, the error is increased almost by a factor 2 (from 3.4° to 6.4°) with respect to the FIR filters approach (using $\tau = 1$). Using the fixed point version of this recursive filters, the degradation due to quantization error is negligible (less than 0.1°), which justifies this bit width choice. The FIR filter based approach requires storing 4 images to compute filters of 5 taps. We also evaluate the hardware resources utilization of both alternatives as shown on table 1. Note that these results generated with the DK are pre-place and route and therefore, they should be considered just to evaluate the complexity of different approaches. Final results are slightly different in terms of resources consumption and maximum clock timing due the FPGA place and route process.

Table 1. Resources consumption for 2 temporal filter approaches implementation of stage S_1 on a Virtex II XC2V6000-4 (results taken from the DK synthesizer (Celoxica 2006)). The IIR filters require about 22% more resources. Although they achieve a faster clock rate, it is not the clock frequency limiting stage, and therefore, it is not a significant improvement in the framework of the optical flow system.

Pipelined stages	Equivalent gates	FFs	Memory bits	Max clock frequency (MHz)
S_1 (FIR temporal filters)	156 266	529	73 728	68
S_1 (IIR temporal filters)	193 984	642	92 160	73

The design strategy (IIR vs. FIR temporal filters) relies on the number of taps required for some specific environments and tasks, as difference in hardware resources is not very significant. If low frame rate cameras are used, we need to filter temporal high frequency information to avoid aliasing and therefore a larger number of taps is required, making the IIR approach a better option (no extra external memory resources are required). Nevertheless, this means that the motion and light conditions should be continuous and this is not always a realistic situation. Therefore, we consider that a high frame-rate camera with FIR filters is a better choice, because it provides higher accuracy and can be used in more generic environments.

4. Implementation of a highly parallelized architecture

We have described the coarse pipelined system of 5 stages. Now, we describe the number of parallel units used at each stage and the number of fine-grain pipeline stages per unit. In fact, previous approaches used a coarse pipeline processing architecture able to process up to 41 Kpps (Kpps $\equiv$ kilo pixels per second) (Díaz *et al.* 2004, 2006b). The previous scheme, with a pipelined structure divided on 5 basic stages, leads to high performance, but is still far from high frame-rate processing requirements. The main reason is that the coarse architecture in figure 1 utilizes a structure similar to that of DSP processor. There is a trade-off between pipeline length and system performance based on the dependence problems (in DSPs, branch conditions often stop the pipeline, which represents a significant time loss). Therefore, long pipelines are not presented on standard DSPs and microprocessors. On the other hand, we hereby describe a specific purpose processing architecture that highly benefits from a fine grain pipeline datapath. According to Forsell (2002), the best architecture is a superscalar and superpipelined structure. This design strategy has been adopted in our approach and is one of our novel contributions compared with Díaz *et al.* (2004, 2006b). Furthermore, this processing strategy leads to an outstanding processing performance. In figure 1, we present the global scheme. Each coarse stage has been finely pipelined leading to a processing datapath of more than 70 stages just for the optical flow computing core. The number of scalar units grows at stages in which L&K model requires to maintain the system throughput. This parallelism expansion represents the following.

(a) Stage S_0 uses one scalar unit for spatial smoothing with 12 pipeline stages.

(b) Stage S_1 uses two scalar units, one for temporal smoothing and another one for temporal differentiation. Each one requires 9 pipeline stages.
(c) Stage S_2 uses 3 parallel scalar units of 12 pipeline stages, corresponding to the 3 dimensions (I_x, I_y and I_t) in which the image derivatives are computed.
(d) Provided that 5 cross-products ($I_x \cdot I_x$, $I_y \cdot I_y$, $I_x \cdot I_y$, $I_x \cdot I_t$ and $I_y \cdot I_t$) are computed at stage S_3, the system uses 5 parallel units of 12 pipeline stages to comply with this requirement. These scalar units compute the weighted sum of these cross-products needed at the least-squares fitting.
(e) Finally, stage S_4 uses one scalar unit of 25 pipeline stages to compute the final motion for each pixel, but internally, several parallel pathways drive the data process.

In the next subsections, we describe the architecture details that allow implementing the motion processing circuit and all these parallel datapaths.

4.1 *Memory management unit*

In § 3.2, we conclude that the FIR temporal filter is the best solution for our system, but its specifications have a special effect on the designing process considerations. The limited number of memory banks accessible on board constraints the available system parallelism (which translates in performance degradation) and increments the design complexity. Therefore, an efficient memory management unit (MMU) becomes of great interest to abstract the sequential access inherent to this kind of devices. For this purpose, we create virtual memory ports (VMP), whose behavior emulates parallel independent real memory ports. The main idea for this implementation is to combine the following concepts/properties.

1. Nowadays, long memory words (36 bits) make it feasible to store up to four 9-bit-width data at each memory address with more than 512 Kaddress (IDT 2006) (up to 5 images of 720×576 pixels per memory chip).
2. A throughput of one pixel per cycle is possible by using pipelined packing and unpacking circuits, which only requires to access memory once in 4 clock cycles.

We have designed an MMU which benefits from the previous architectural descriptions. Depending on the number of VMPs required and packing/unpacking possibilities (provided by the memory word bit-width), a state machine is used to feed the VMP registers sequentially, achieving a final performance of one data per cycle. Furthermore, this architecture is scalable because an increment of N in the number of VMPs available on one memory only modifies the required access cycles on a factor of N. This can be further optimized by incrementing the MMU clock frequency by this factor with respect to the global system clock frequency. There is only one limitation, due to the packing/unpacking circuits, random access is limited to a multiple of 4. Besides, for an efficient data management, they should be stored in memory in a consecutive packed way.

The MMU architecture is illustrated in figure 2 for a four VMP case. Note that a VMP is composed of a 4 addresses register (read or write type) plus a data-write

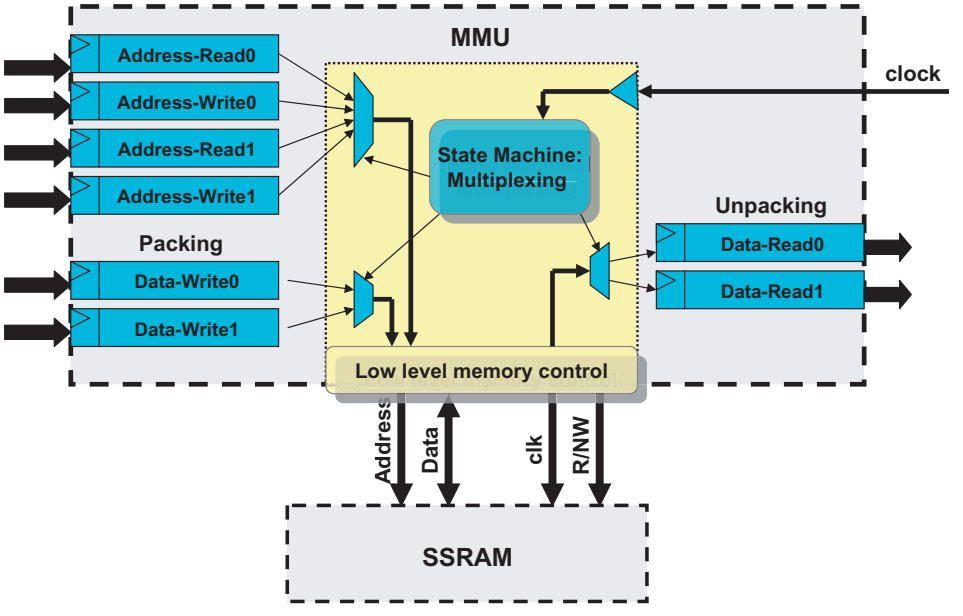

Figure 2. MMU schematic for a 4 VMPs expansion-case. VMPs are represented by one address register (type Read or Write) and a Data-Write or Data-Read register. Low level memory control manages the data and address signals as well as the SSRAM clock, read-no-write signal (R/NW), etc. The state machine feeds four VMPs sequentially and manages the low level memory access. Packing/Unpacking circuits achieve a total throughput of one pixel per clock cycle. This architecture allows us to multiply the equivalent memory parallel access by 4 .

register with packing circuits or by a data-read register with unpacking circuitry. Previous implementation of L&K used IIR filters to reduce the memory access, but the drawback was the accuracy degradation (Fleet and Langley 1995, Díaz *et al.* 2004, 2006b). The presented architecture allows the easy management of a large number of read-write processes necessary for FIR temporal filters with a minimum FPGA logic, which clearly justifies the design of the presented MMU architecture.

4.2 *Stage S_3 architecture*

Section 5 evaluates the resources consumption of the different circuit stages and highlights this stage as the one which requires more hardware resources. As shown in figure 3, this stage is expanded to compute the five cross-products utilized for the least squares fitting process. This is implemented as five parallel segmented scalar units. Furthermore, incoming data for this stage require 18 bits, which makes the arithmetic circuits consume a considerable circuit area. As illustrated in figure 3, the main computing circuit of this stage is the separable convolution unit which implements the weighted average calculation.

4.3 *Stage S_4 architecture*

Stage S_4 is critical in terms of system frequency, resources, and accuracy. The incoming data use fixed point representation of 18 bits, and this stage requires the operations of multiplication, addition/subtraction and division without losing accuracy. From our previous analysis (Díaz *et al*. 2004, 2006b), we have decided to use floating point data which allows obtaining the required precision with reasonable resources consumption (as is shown in table 2). Figure 4 presents the architecture of this stage, based again on a high pipelined and parallel datapaths to achieve a high system throughput. The whole stage requires 25 steps. Data conversion, multiplication, addition, and subtraction are computed in just one cycle, but division requires 15 steps. This is the stage limiting the system clock frequency and it could be even further pipelined to increase the clock frequency if necessary.

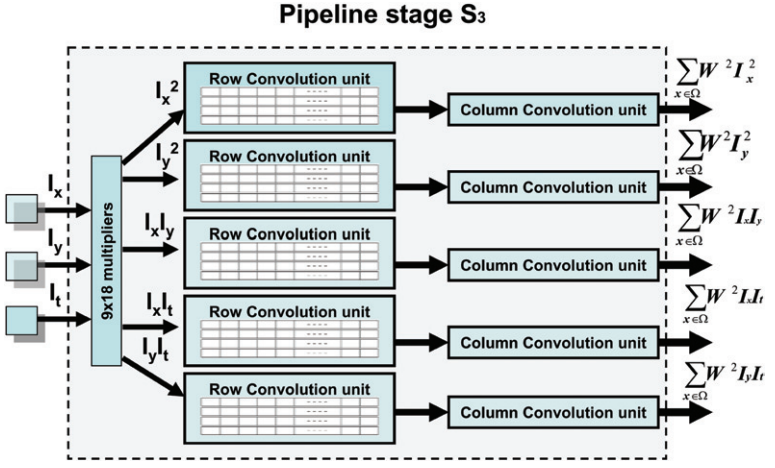

Figure 3. Architecture schematic of the least squares matrices construction. Pipeline stage S_3.

Table 2. Basic pipeline stages resources consumption on a Virtex II XC2V6000-4 (results taken from the DK synthesizer (Celoxica 2006)). Non-motion core indicates the logic associated to the MMU, Video input controller, VGA signal generation, user interface, etc. Last rows indicate the implementation of stage S_4 with different parameters and data representation, where man stands for mantissa and exp for exponent.

Pipelined stages	Equivalent gates	FFs	Memory bits	Max clock frequency (MHz)
S_0 Gaussian smoothing + Non-motion core modules	66 213	2081	20 208	45
S_1 FIR temporal filters	156 266	529	73 728	67
S_2 FIR spatial derivatives	454 087	639	221 184	60
S_3 Construction of least squares matrices	478 034	1873	221 184	51
S_4 Floating point unit (11 man + 7 exp)	57 167	3488	0	45
S_4 Floating point unit (17 man + 7 exp)	131 193	4938	0	36
S_4 Floating point unit (23 man + 7 exp)	207 428	7698	0	34
S_4 Fixed point unit (36 bits)	345 981	1080	0	31

4.4 *Global system superpipelined datapath description*

We have described the main computing stages on the previous subsections. The parallelism of the system is expanded according to the algorithmic structure and the final architecture is described in figure 5. Note that the MMU units are critical allowing the management the memory accesses of the different elements as described on the figure.

The synchronization among the different processing units (frame-grabber, motion processing, VGA, and user interface) is accomplished by using specific

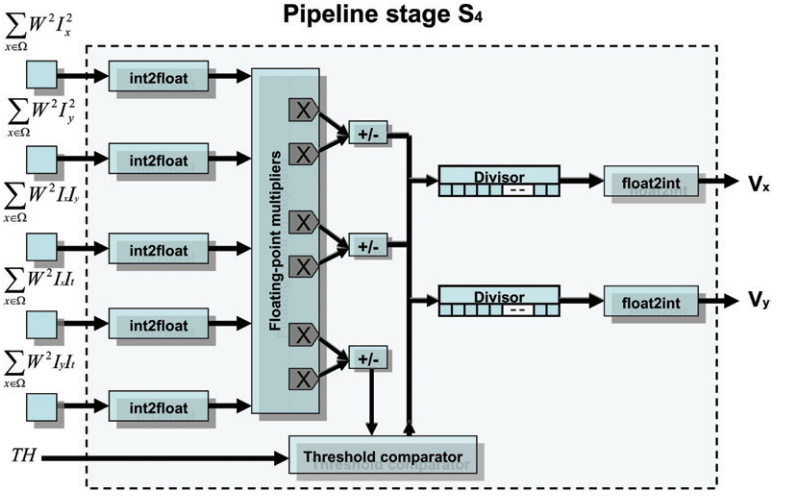

Figure 4. Architecture schematic of the floating-point unit. Pipeline stage S_4.

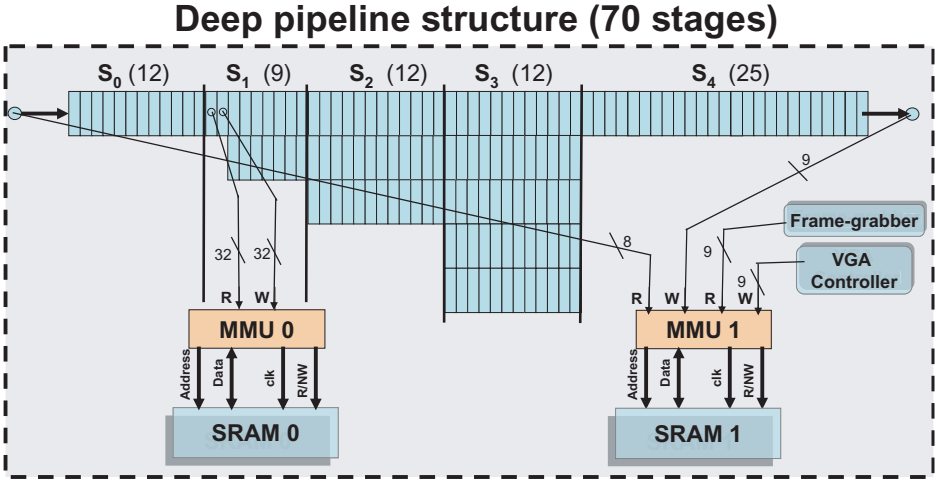

Figure 5. Architecture schematic of the global system. We represent the number of functional units (as parallel pathways), fine grain pipeline stages (indicated as rectangular cells and in brackets in the top labels corresponding to each coarse grain pipeline stage), and memory access elements (MMU channels are included).

external memories as data buffers, which solves the problem associated with the different clock frequencies. The memory interchange strategy makes use of delays between processing units as a synchronization technique. As described in § 4.1, VMPs requires a constant number of cycles to feed the data input register. If we use an exactly constant number of cycles for all the processing units, our circuits can make use of this to properly access input/output data registers without requiring additional synchronization logics. In order to maximize processing performance, we have used one cycle in all our processing stages which is feasible thanks to the MMU packing and unpacking data strategy which eliminates all the technology dependencies. This enables the design of a very deep pipeline processing structure without using branch predictions that would degrade the overall performance. The high system throughput is based on this deep pipeline and on the parallel scalar units of different stages designed according to the Lucas and Kanade algorithmic complexity. Well balanced units are used to achieve a final system throughput of one estimation per clock cycle. Only on specific points (for example VGA controller) interprocess communication is needed. In these situations, we use an interface module between the two processes, with a synchronized and buffered point-to-point communication scheme. This module blocks the communication until both modules (sender and receiver) are ready and data are transferred, allowing synchronizing hardware controllers with different clocks or other characteristics.

5. System resources and performance

The whole system has been successfully implemented and tested on a stand-alone board for image processing applications (Celoxica 2006). This board is provided with a Virtex II XC2V6000-4 Xilinx FPGA as processing element, also including video input/output circuits and user interfaces/communication buses. Table 2 summarizes the estimation of the system resources utilization for each pipeline stage to determine the critical circuits in terms of frequency and resources cost. Note that these results generated with DK are pre-place and route. The key-idea of this work methodology is to give a quick overview of the complexity of the different processing stages. DK synthesizer allows a fast look into the resources consumption and timing which helps to accelerate the designing process of very complex designs.

The computations use fixed point arithmetic on stages S_0 to S_3, and floating point on stage S_4. We include the values of the requirements for pipeline stage S_4 with different approaches. Rows number 5, 6 and 7 include an approach using a customized floating point representation. It uses one bit for the sign and customized bits for the mantissa and the exponent as indicated on table 2. The final row presents an implementation using fixed point arithmetic. Due to the fact that the input data to this stage uses 18 bits data, its implementation with fixed point arithmetic requires at least 36 bits to avoid loss of accuracy. This large bit-width reduces the maximum achievable clock rate and therefore our performance. Of course, further pipelining of this stage can improve the performance but resources consumption is already quite high and, from table 2 it is clear that floating point representation is a better choice for this stage. The dynamic range of the floating point version is larger, which is very important to represent small numbers after the division operation without requiring bits extensions. Several bits configurations of the floating point data have

been implemented. The number of bits dedicated to the mantissa allows us to define the accuracy vs. performance of the system. In the final version, we have employed a mantissa of 11 bits because, due to the fact that optical flow is prone to noise, good results can be achieved only by using the most significant bits of the data as can be seen from the illustrative results of figure 6. We numerically evaluate representations, fixed point and floating point (11 bits case). Using the Yosemite flow-through synthetic sequence with known ground truth, we found an angular error of 3.42° and 3.37° respectively, which represents a fairly similar accuracy. This indicates that the customized floating point version is a better candidate for our computing architecture.

Figure 6(a) corresponds to a diverging tree sequence produced by the simulated approaching of the camera to the tree. All the spurious deviations of the flow from a central expansion are artifacts produced in areas with low image structure. Figure 6(b) corresponds to a driving scenario. We have tested the global accuracy of the flow using the configurations of S_4 indicated in table 2 and they lead to similar results (using the benchmarking sequences of Barron *et al*. (1994)).

After the previous pre-place and route resources estimations (tables 1 and 2), table 3 shows the final hardware costs of the whole designed system (processing motion core, MMUs, frame-grabber, VGA output, and user-interface) and its performance. These final results are obtained after the place and route process. The image resolution can be selected according to image input camera standard or processing capabilities. This architecture is scalable, being possible to reduce the system parallelism (and performance) to fit on a smaller device.

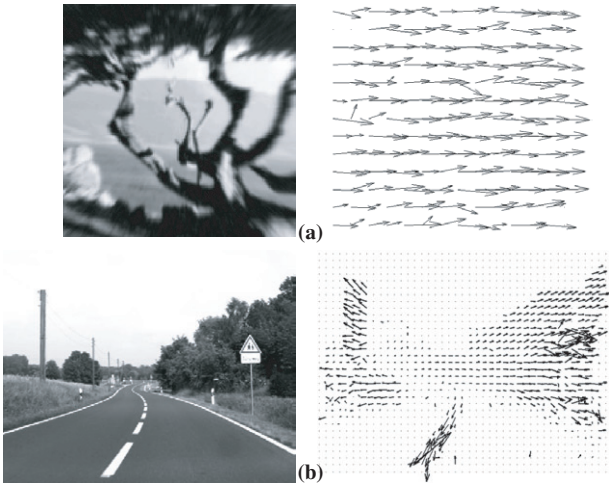

Figure 6. Optical flow processing results (11 bits for the mantissa in S4) for two sequences. (a) Translating tree sequence, synthetic sequence used in Barron *et al*. (1994) and available at: ftp://ftp.vislist.com/SHAREWARE/CODE/OPTICAL-FLOW/. Note that, although the movement is to the right, some flow vectors have vertical components. These components mainly appear on image areas where the aperture problem is present and only normal flow to the image edges can be computed. (b) Real sequence of a camera mounted on a car and looking forward. The expanded flow represents the ego-motion pattern and is useful for computing heading information.

Table 3. Complete system resources required on a Virtex II XC2V6000-4 after place and route. The system includes the optical flow processing unit, memory management unit, camera Frame-grabber, VGA signal output generation and user configuration interface. (Mpps: mega-pixels per second at the maximum system processing clock frequency, EMBS: embedded memory blocks.).

Slices (%)	EMBS (%)	Embedded multipliers (%)	Mpps	Image Resolution	Fps
8250 (24%)	29 (20%)	12 (8%)	45.49	640×480	148
				1280×960	37

Furthermore, the processing core can be replicated (more than 75% of system resources of a Virtex II XC2V6000 are available) and the frame-grabber can be easily modified (thanks to the MMU architecture) to split the image and send it to several processing units. This high level scalability allows multiplying the processing performance on this board. However, since the architecture already fulfils the requirements to compute in real-time sequences at a high frame-rate, we have not addressed this issue.

5.1 *Performance comparison with other approaches*

The implementation of the optical-flow algorithm with FPGAs has only been addressed by some authors in very recent years. In our previous work (Díaz *et al.* 2004, 2006b), a basic implementation of the L&K model was proposed, and we presented a detailed study about the performance vs. system resources trade-off. Although the performance was quite high (1776 Kpps), neither the image resolution nor frame-rates perform the high frame-rate requirements addressed here. The iterative algorithm of Horn and Schunck (1981) (H&S), has also been implemented by different authors. Martín *et al.* (2005) presented a system implementation that fits quite well the specification of a standard frame-rate optical flow system capable of processing up to 3932 Kpps. The main disadvantage of that approach is that the model itself obtains poor accuracy as shown by Barron *et al.* (1994). Using the block-matching approach, the implementation described by Niitsuma and Maruyama (2004) achieves 30 fps of image size 640×480 but with high hardware cost (90% slices of a XC2V6000 FPGA) and without sub-pixel accuracy. Finally, the model described here, running in software on an Intel Pentium 4 HT, 3200 MHz has been tested. This software was implemented in simple plane C, using the main compiler options to tune the processor but without addressing optimizations using MMX and SSE instructions. It is able to compute 3 fps of 640×480 pixels (914 Kpps). Though the software can be further optimized, the main problem is that it consumes all the computing resources of the machine.

Our system has been experimentally tested running up to 45 MHz and due to the fine-grain pipelined architecture, we have computed 45 Mpps, which is more than one order of magnitude higher than previous approaches. Since the referenced works are very recent (some of them using the same evaluation devices), the outstanding performance of our approach does not rely on the technological improvements but rather on a very efficient processing architecture that uses extensively the parallel resources of the FPGA device.

6. Conclusions

This paper presents a high parallelized architecture for high frame-rate motion estimation, carefully designed to provide one pixel motion estimation per clock cycle. We have analysed the different processing stages, design alternatives and illustrated the system results. To the best of our knowledge, the presented system outperforms in more than one order of magnitude any previous approach, validating the proposed architecture. The necessity of a system for high frame-rate optical flow processing has been clearly motivated because of two main reasons: it decreases temporal aliasing and it fits better the first order gradient constraint assumption. Current image sensors allow a very fast image acquisition and simple gradient based optical flow approaches seem to be one of the most suitable alternatives for real-time processing systems onto customized hardware.

According to this, we have implemented an improved version of the L&K model (Brandt 1997), which complements the capabilities of high frame-rate cameras providing real-time image motion analysis. We have presented a novel architecture that addresses the real-time optical flow computation of high frame-rate and high resolution sequences using a FPGA device as processing element. We have described the architecture and illustrated how parallelism and superpipelined structures can be defined for image processing applications. This work presents an optic flow specific datapath of 70 pipelined stages which is able to compute more than 45 Mpixels/s. The system consumes less than 25% of the hardware resources of a 6 million gates FPGA device. Furthermore, the system design is very modular and the computing parallelism and precision can be easily changed to adapt the datapath to low cost FPGA devices.

Finally, we have evaluated the system resource consumption and performance of an implementation on a stand-alone platform which fulfils the high frame-rate optical flow requirements. The comparison with previous works clearly shows the outstanding performance of the system and opens the door to a wide range of application fields.

Future works will cover the utilization of such systems on real-world applications using moving robotics platforms, such as robot navigation, tracking, as well as structure extraction from motion analysis.

Acknowledgments

This work has been supported by the grants DEPROVI (DPI2004-07032) and DRIVSCO (IST-016276-2).

References

A. Bainbridge-Smith and R.G. Lane, "Measuring confidence in optical flow estimation", *IEE Electron. Lett.*, 10, pp. 882–884, 1996.

A. Bainbridge-Smith and R.G. Lane, "Determining optical flow using a differential method", *Image Vis. Comp.*, 1, pp. 11–22, 1997.

J.L. Barron, D.J. Fleet and S. Beauchemin, "Performance of optical-flow techniques", *Int. J. Comp. Vis.*, 12, pp. 43–77, 1994.

V. Bonato, M.M. Fernández and E. Marques, "A smart camera with gestures recognition and SLAM capabilities for mobile robots", *Int. J. Electron.*, 93, pp. 385–401, 2006.
J.W. Brandt, "Improved accuracy in gradient based optical flow estimation", *Int. J. Comp. Vis.*, 25, pp. 5–22, 1997.
A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger and C. Schnorr, "Variational optical flow computation in real time", *IEEE Trans. Image Process.*, 14, pp. 608–615, 2005.
Celoxica Company. Web site and products information available online at: http://www.celoxica.com/ (accessed 26 April 2006).
C.R. Clark, C.D. Ulmer and D.E. Schimmel, "An FPGA-based network intrusion detection system with on-chip network interfaces", *Int. J. Electron.*, 93, pp. 403–420, 2006.
J. Díaz, E. Ros, S. Mota and R. Rodriguez-Gomez, "Highly paralellized architecture for image motion estimation", *LNCS*, 3985, pp. 75–86, 2006a.
J. Díaz, E. Ros, E.M. Ortigosa and S. Mota, "FPGA based real-time optical-flow system", *IEEE Trans. Circ. Sys. Video Tech.*, 16, pp. 274–279, 2006b.
J. Díaz, E. Ros, S. Mota, R. Carrillo and R. Agís, "Real time optical flow processing system", *LNCS*, 3203, pp. 617–626, 2004.
C. Dumontier, F. Luthon and J.P. Charras, "Real-time DSP implementation for MRF-based video motion detection", *IEEE Trans. Image Process.*, 8, pp. 1341–1347, 1999.
D.J. Fleet, *Measurement of Image Velocity, Engineering and Computer Science*, Norwell, MA: Kluwer Academic Publishers, 1992.
D.J. Fleet and K. Langley, "Recursive filters for optical flow", *IEEE Trans. Pattern Anal. Mach. Intellig.*, 17, pp. 61–67, 1995.
M.J. Forsell, "Architectural differences of efficient sequential and parallel computers", *J. Sys. Archit.: EUROMICRO J.*, 47, pp. 1017–1041, 2002.
B.K.P. Horn and B.G. Schunck, "Determining optical flow", *Artif. Intellig.*, 17, pp. 185–204, 1981.
IDT, SRAM ZBT memories, part number: 71T75602. Datasheet available online at: www.idt.com (accessed 26 April 2006).
T. Kerins, W.P. Marnane and E.M. Popovici, "Algorithms and architectures for a flexible elliptic curve cryptography processor", *Int. J. Electron.*, 93, pp. 349–372, 2006.
S. Lim, J.G. Apostolopoulos and A.E. Gamal, "Optical flow estimation using temporally oversampled video", *IEEE Trans. Image Process.*, 14, pp. 1074–1087, 2005.
H.C. Liu, T.S. Hong, M. Herman, T. Camus and R. Chellappa, "Accuracy *vs.* efficiency trade-offs in optical flow algorithms", *Comp. Vis. Image Und.*, 72, pp. 271–286, 1998.
B.D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision", in *Proc. of the DARPA Image Understanding Workshop*, pp. 121–130, 1984.
J.L. Martín, A. Zuloaga, C. Cuadrado, J. Lázaro and U. Bidarte, "Hardware implementation of optical flow constraint equation using FPGAs", *Comp. Vis. Image Und.*, 3, pp. 462–490, 2005.
S. Maya-Rueda and M. Arias-Estrada, "FPGA processor for real-time optical flow computation", *LNCS*, 2778, pp. 1103–1016, 2003.
B. McCane, K. Novins, D. Crannitch and B. Galvin, "On benchmarking optical flow", *Comp. Vis. Image Und.*, 84, pp. 126–143, 2001.
H. Niitsuma and T. Maruyama, "Real-time detection of moving objects", *LNCS, FPL 2004*, 3203, pp. 1153–1157, 2004.
E.P. Simoncelli, "Design of multi-dimensional derivatives filters", in *Proc. IEEE Int. Conf. on Image Processing*, Austin Tx, 1994, pp. 790–794.
J. Weber and J. Malik, "Robust computation of optical flow in a multi-scale differential framework", *Int. J. Comp. Vis.*, 14, pp. 67–81, 1995.