Reconfigurable Streaming Architectures for Embedded Smart Cameras

Sek M. Chai, Nikolaos Bellas, Greg Kujawa, Tom Ziomek,
Linda Dawson, Tony Scaminaci, Malcolm Dwyer, Dan Linzmeier,
Embedded Systems Research, Motorola Labs,
(*sek.chai@motorola.com*)

Abstract

Smart cameras using FPGAs require an automation method to simplify the design process and to ensure both computation and memory performance are met. Reconfigurable logic allows exploration of different hardware accelerators and memory-hierarchy configurations based on application needs. This paper presents a streaming architecture template that is generated from high level program descriptions. A smart camera development platform, the software architecture, and demonstration template are also described.

1. Introduction

Embedded smart cameras using computer vision methods for video analysis [1,2] can benefit from reconfigurable FPGA platforms to provide the necessary performance and flexibility [3]. The levels of integration of modern FPGAs have advanced to a point where all functions of a complex System on Chip (SoC) can be mapped onto a single die. FPGA manufacturers have embedded scalar processor cores, multipliers, and SRAM memories in order to speed-up commonly used algorithms. They also offer peripherals, fixed IP functions, and even synthesizable processor cores for further customization. Architecture designs for smart camera applications can be configured on the FPGA platform to better optimize the memory subsystem and computation structures.

This paper presents a streaming architecture template which consists of hardware accelerators and memory subsystem to support the computation and bandwidth requirements. The design process consists of a stream programming model with the familiarity of a high level programming language. The hardware accelerators are generated from user defined kernels while the streaming memory subsystem is capable of automatic prefetching and alignment. Following the design process allows a larger segment of engineers that may not have expertise in system architecture and hardware design to prototype on FPGAs. Since particular hardware structures are abstracted out with a software-only front end interface, application development becomes less complicated.

Furthermore, applications related to video analysis are often limited by bandwidth because of the imbalance between processor and memory performance [4]. Even though FPGAs continue to provide larger numbers of configurable logic blocks that can be mapped to processing elements to speed up computation, the interconnect delays and slow memories can become bottlenecks. The streaming model decouples the descriptions of memory access sequences from the computation within a kernel, thus making the customization of each of these two components (computation and memory access) easier and more amenable to optimization. The system is then synthesized for an FPGA in a development kit with integrated image sensors and peripherals for video analysis.

The structure of the paper is as follows: Section 2 presents the related work relevant to this paper; Section 3 gives a brief presentation of the system architecture, stream programming model, and architecture template; Section 4 describes a smart camera development platform and the associated software platform; Section 5 concludes the paper.

2. Related Work

Stream processing is a computational model that operates on sequences of ordered data (streams) using computation kernels (filters) [5]. While both industry and academia have studied the concurrency of computation and data movement, this streaming model provides a new and interesting framework that brings together both task and data level parallelism within the same context. The programmer explicitly defines the data accesses and computation separately, thereby exposing concurrency and locality for the compiler to schedule both in hardware.

The computation and data movement characteristics of smart camera applications are a good match for the stream model of computation. Making data movement explicit and describing which portions of the application can be computed in parallel enable compilers to optimize data movement and match it to the available hardware accelerators.

A number of streaming processor architectures have been developed over recent years. Examples of stream processors include RAW [6], Imagine [7], Merrimac [8], and the RSVP™ architecture [9,10]. Stream processors are similar to vector processors in their ability to hide latency, amortize instruction overhead, and expose data parallelism by operating on large sets of data. However, stream

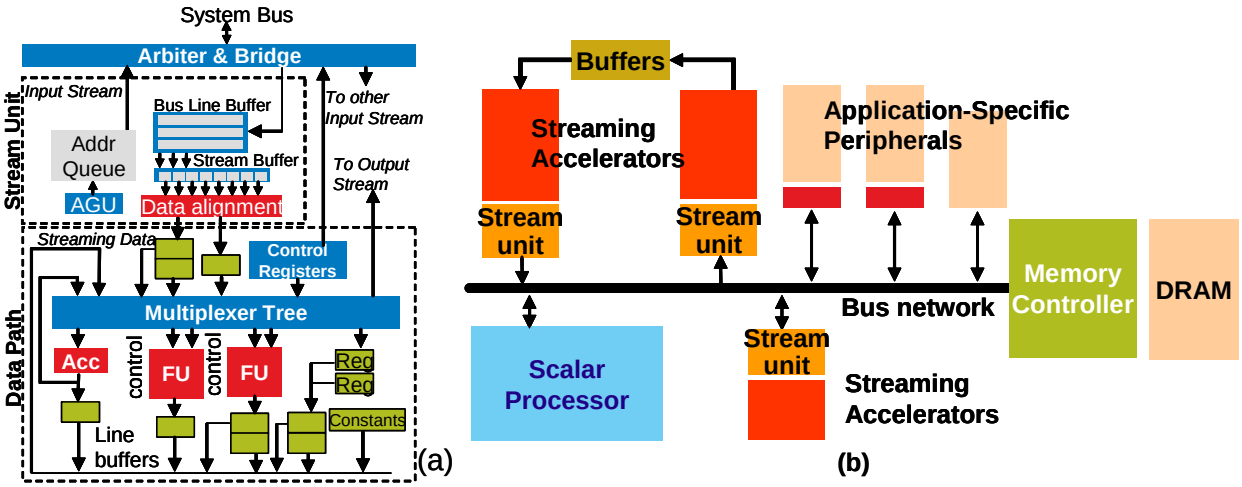

Figure 1. (a) Generated streaming accelerators, (b) Integration of streaming accelerators in SoC design

processors support a more complex access pattern by allowing the programmer to explicitly define the data movement.

There are also other classes of streaming architectures with origins from reconfigurable FPGA platforms. These architectures rely on the flexibility of the platform to synthesize streaming accelerators based on programmer definition. In comparison to the above mentioned architectures, a set of compiler tools create optimized hardware configurations rather than map computation onto ASIC designs. These architectures are associated with the programming language or compiler tool that allows software developers to configure hardware for stream computation. Examples include SCORE [11], ASC [12], and Streams-C [13].

This paper describes a streaming architecture template comprised of hardware accelerators and a memory subsystem to perform the computational heavy lifting for a scalar processor. The accelerators are generated from program instructions while the memory subsystem is based on a description of data shape in memory. This programming model uses an extension on the format presented in [9, 10], and is described in Section 3. Using the stream computation model, which separates description of computation and communication, the accelerator and memory subsystem can be optimized separately on the FPGA for different applications.

The FPGA platform allows different configurations of the accelerator and memory subsystem. While there are many FPGA development boards available from either FPGA or third party vendors, it is the understanding of these authors that none are made specifically for embedded computer vision. Readers are referred to [14] for a review of available FGPA development boards. FPGA board support packages with key components for image processing (hardware and software templates for image processing, integrated device drivers for one or more image sensors, and standard application example) are usually not available on a generic development kit. This paper presents these elements in an FPGA platform (Watson) to support the streaming accelerators. Once optimized for a particular class of computer vision applications, the design can be ported into standard or structured-ASIC design flows for fabrication and productization.

3. System Architecture

3.1. Design flow

The design process consists of the application programmer describing the application in a high level language such as C. For computationally intensive kernels, such as loops, hardware accelerators are generated to lift the heavy computation load from the scalar processors. Scalar processors are reserved for normal conditional code since that code is not easily parallelizable and generated hardware is not efficient [15]. Our streaming architecture template, shown in Figure 1, is dedicated for high throughput, parallelizable code, with support for all kinds of parallelism (instruction, data, and task).

The overall design flow and automation tool is described in [16]. The design flow is not unlike [17] with the exception of the streaming architecture templates. This paper is focused on the streaming architecture template and its use in embedded smart cameras. In Section 4, an example application is accelerated in hardware using this flow. Currently, the kernels are described using a stream data flow graph (sDFG). A compiler can then create a parameterized hardware description of the hardware accelerator. This description is used to generate RTL (Verilog), and later synthesized onto the FPGA.

3.2. Streaming Hardware Architecture Template

There are two parts to the architecture template: the streaming data path and the stream unit (Figure 1a). Computations are mapped to the data path, while the

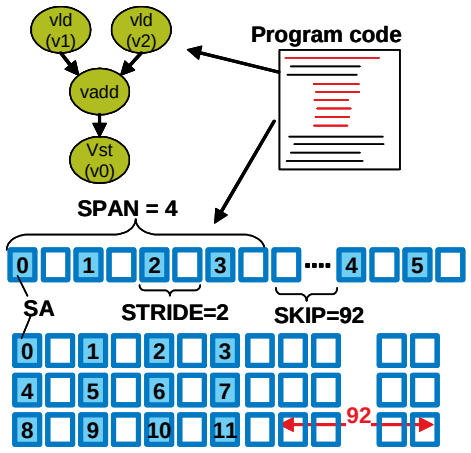

Figure 2. Kernel computation (sDFG) and memory access (stream descriptors) are extracted from program code

stream unit forms the system's memory hierarchy. Each data path and stream unit have their own generation process.

The architecture uses stream units to prefetch data from memory and turn data streams into FIFO queues of stream elements for the data path. Each stream unit handles all issues regarding loading/storing of data including: address calculation, byte alignment, data ordering, and bus interfacing. The stream unit consists of one or more input and output stream modules, and is generated to match the characteristics of the programmer's description of the stream data (stream descriptors), the characteristics of the bus-based system and the streaming data path. An address generation unit actively creates memory addresses for the system bus while bus line buffers temporarily store bus data. Stream buffers are used to reorder and align the data based on computational needs. Readers are referred to [18] for more details about the reconfigurable elements of the stream unit.

A data path is generated to execute a given kernel defined in the program code. There is a network of functional units that produce and consume streaming data elements. A reconfigurable link is formed by a tree of multiplexers and line queues to direct proper data elements to each functional unit. The control logic is distributed and spatially near the corresponding functional unit, multiplexer, and line queues. This was done to avoid long interconnects in critical paths. The reconfigurable parameters of the data path include the following: type of functional units (ALUs, multipliers, shifters, etc), the custom operation performed within a type (e.g. only addition or subtraction for an ALU), the width of the functional unit, the size and number of storage elements, the interconnect between functional units, and the bandwidth to and from the stream unit. Readers are referred to [19] for more details about the reconfigurable data path and the sDFG mapping process.

Figure 1b illustrates an example SoC that can be placed into the FPGA along with the generated hardware accelerators. A set of peripherals is used to control external components of the embedded smart camera on the development board, described later in Section 4.

3.3. Stream Programming Model

Computation and data accesses for the streaming accelerator are defined separately in the program code, as shown in Figure 2. We are using a stream data flow graph (sDFG) language to express operations composing the streaming operations in a machine independent manner. A sDFG consists of nodes representing basic arithmetic, logic, and load/stores operations. The directed edges of the sDFG represent the dependency of one operation on the output of the previous operation [9,10]. The use of a graphing language is very useful in describing signal processing applications and has been used in computer vision applications [20].

Memory access patterns are expressed using a programming API, called a stream descriptor. A stream descriptor is represented by the tuple (Type, Start_Address, Stride, Span0, Skip0, Span1, Skip1, Size), where:

- Type indicates the element size in bytes (Type is 0 for bytes, 1 for 16-bit half-words, etc.).
- Start_Address represents the memory address of the first stream element.
- Stride is the spacing, in number of elements, between two consecutive stream data elements.
- Span0 is the number of elements that are gathered before applying the skip0 offset.
- Skip0 is the offset applied between groups of span0 elements, after the stride has been applied.
- Span1 is the number of elements that are gathered before applying the skip1 offset.
- Skip1 is the offset applied between groups of span1 elements, after the stride and the Skip0 have been applied.

The Stride, Span, Skip, and Type fields define the shape of the data stream in memory, while Start_Address defines the location of the first data element. The grouping and order in which data is accessed defines a Stream Record and corresponds to the desired alignment for the computation kernel. Multidimensional or even non-regular shapes can be created by extending the defined semantics of each stream descriptor. Figure 2 shows an example of a memory access pattern described by a stream descriptor, which is loaded into the stream unit (shown previously in Figure 1) to move data on the system bus.

Stream descriptors have been used to optimize transfers from I/O devices [21] and from memory [22]. Similar techniques to describe the shape of memory accesses have also been used for trace generation [23,24]. Stream descriptors are a language extension to specify memory

Table 1. Highlighted features of the FPGA-based smart camera board

Attributes	Description
FPGA	FPGA device (one of three different Virtex 4 FX devices, XC4VFX40, XC4VFX60, XC4VFX100) in FF1152, 35x35 mm size Ball Grid Array (BGA) package, with embedded Power PC (PPC) core.
DDR Banks	Double Data Rate (DDR) SDRAM, 128 Mbytes, 200 MHz, MT46V16M16FG chips
Image Sensors	Up to four pairs of image sensors: parallel or low voltage differential signaling (LVDS) connections
UART	Universal Asynchronous Receiver Transceiver (UART); 120kbps at RS-232 output levels.
FLASH Memory	32 Mbytes, 120nsec, MT28F128J3BS chips
Ethernet	10/100/1000 BaseT, Marvell 88E1111 PHY, RJ45 connector with integrated magnetics
USB OTG	Universal Serial Bus (USB) on-the-go (OTG) Revision 2 compliant, 480 Mbit/sec
Display	RGB Video Monitor Output. Analog Devices ADV7123, 15-pin D-shell connector
GPIO	General Purpose Input/Output (Eight inputs, eight outputs), DIP switch and LED indicators
Compact Flash	Three mode operation: Memory mode, Input/Output mode and True IDE mode
Power Supply	Wide voltage range from +5V to +30V. Designed to withstand harsh automotive environments

access patterns, which is used by dedicated stream units to prefetch and assemble data. The stream descriptors and compiler manipulations are active research areas. Readers are referred to [9,10] for more details.

4. Smart Camera Development Kit

4.1. Hardware Platform

The generated hardware accelerator and peripherals, described in Section 3, are synthesized onto an FPGA on a smart camera development board (Watson). This section presents the hardware aspects of the platform. The Watson platform offers a wide variety of peripherals, highlighted in Table 1. Many combinations of these peripherals may be used in a particular system, but the full description is beyond the scope of this paper. Figure 3 illustrates the main Watson board and four image sensor boards.

This section will describe three unique attributes of the Watson platform: multiple image sensor support, separable memory banks, and FPGA system reconfiguration. As in many reference platforms such as [25], there are certain features that are standard. However, the authors believe that these three features are key enablers to deploy certain embedded smart camera applications.

4.1.1 Multiple Image Sensor Interfaces

The Watson platform implements two types of image sensor interfaces, a 10-bit parallel data bus plus control/clock signals, and a low voltage differential signaling (LVDS) serial bus interface known in the industry as BusLVDS. The LVDS interface greatly reduces the signal and pin count of the sensor interface and also allows the sensors to be placed remotely, up to five meters away, from the processing engine. Four ribbon-cable connectors are located on the bottom of the Watson board and on each of the sensor daughter cards to provide for the parallel interfaces to the image sensors. Although the Watson board originally implemented four RJ45 connectors (located on the top of the board) to provide for the LVDS interfaces, these connectors have since been replaced by 9-pin IEEE-1394b beta (Firewire-B) connectors. Extensive testing has shown that the RJ-45 connectors and cabling are too lossy

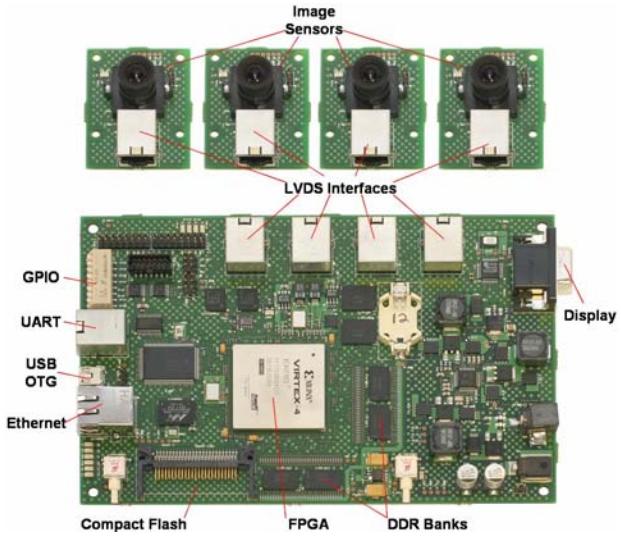

Figure 3. FPGA-based smart camera development kit

at the current LVDS bit rates of 320 and 486 Mbits/sec and that IEEE-1394b connectors and cables are much more suitable for use at these high bit rates. Several image sensor daughter boards with Micron image sensors were designed onto the Watson printed circuit board panel, ranging in resolution from VGA, MT9V022 & MT9V111, to three megapixels, MT9T001 [26]. Depending on each sensor's available hardware interfaces, either the parallel or the LVDS interfaces may be used to collect the sensor data for processing inside the FPGA. The Watson board is capable of supporting up to eight Micron image sensors when each of four pairs of sensors are configured to operate in Micron's proprietary dual-sensor (lockstep) LVDS mode. The daughter cards implement only one sensor per board so the overall system is limited to four image sensors at this time. It should be noted that the available bus bandwidth in the FPGA determines how many sensors can be used simultaneously so a tradeoff between the number of sensors in the system and the frame rates of each sensor interface constitute the necessary compromises.

All of the logic required to implement both the parallel

Table 2. Flash and RAM usage

Description	Flash	RAM (runtime)
Bootloader	129KB	--
Bootloader settings	1KB	--
Linux Kernel	670KB (compressed)	1.5MB (uncompressed)
Linux Filesystem	4.7MB (compressed)	16MB (13MB data, 3MB unused)

and LVDS serial interfaces is contained within the Micron sensor and the FPGA. Those Micron sensors that support LVDS mode contain an embedded serializer that generates the differential data and clock signals directly to the Firewire-B cable. The FPGA contains the LVDS receivers and a custom deserializer module that outputs parallel interface signals identical to those generated by the Micron sensor when it is operating in the parallel interface mode. This enables the use of a common sensor interface module (SIF) within the FPGA that is programmed by software to either accept data directly from the sensor (parallel mode) or from the deserializer module (LVDS mode). The SIF supports sensor resolutions up to 16 megapixels and interfaces to DDR memory via cache-line DMA transfers on the bus without software intervention. Software only needs to program setup information into the SIF registers to match that of the sensor being used.

4.1.2 Separable memory banks

Two banks of Double Data Rate (DDR) DRAM reside on the Watson platform, each consisting of 64 MB. The external databus width is 32 bits and has a maximum clock speed of 200 MHz. The memory banks may be configured as two 64 MB banks of 32 bit wide memory for a single Power PC (PPC) core, one 32 bit wide, 64 MB bank for each PPC core or one 64 bit wide bank of 128MB for a single PPC core. The configuration is application dependent so if a wider memory bandwidth is necessary for a particular design, a 64-bit wide interface may be used to maximize the throughput.

4.1.3 FPGA system reconfiguration

The Watson platform is designed for maximum flexibility for a variety of applications. One of the key components to Watson is to have the ability to remotely change both the software and hardware-configuration when updating algorithms or applications. Configuration circuitry has been incorporated into the Watson platform to allow this feature, i.e. remotely updating software and hardware. Platform flash devices are used to configure the FPGA at power up. Once these devices detect that power has been applied to the system, they proceed to configure the FPGA with the selected hardware configuration. When the system is up and running, a different hardware configuration may be programmed into the platform flash devices which effectively changes the makeup of the application hardware upon the next configuration cycle.

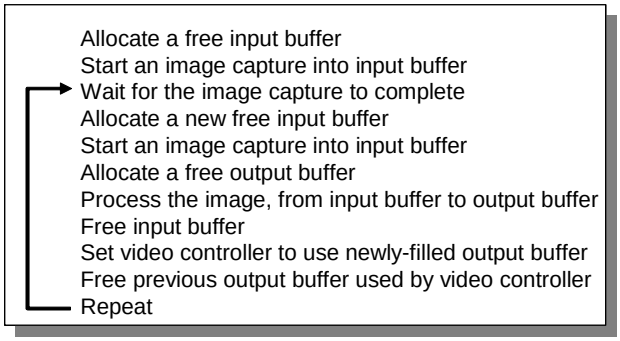

Figure 4. Pseudocode for example Live-View software

4.2. System Software

The computation kernels (sDFG) and stream descriptors, described in Section 3, can be integrated into the system software on the Watson platform. While there are many aspects of the software that could be described, this section describes only the Linux operating system (OS) and demonstration template since they are the key components for embedded computer vision applications.

4.2.1 Linux OS

The Linux OS (kernel version 2.4) is being ported to the Watson platform as part of the development kit to support application developers. The porting task involves porting a bootloader to the Watson platform, porting the Linux kernel and setting up a file-system suitable for a disk-less system. Software device drivers for the image sensor interfaces, described in Section 4.1.1, will be provided to capture images from external sensors. In addition, device drivers for I^2C transfer protocol using the GPIO to control external Micron image sensors are also provided. Using the provided device driver API, application developers can control the image sensors to set exposure gain, region of interest, and other operating parameters.

Table 2 shows a Flash and RAM usage-list to offer an insight to available memory for application developers. While there are many variations on how an application can be setup, the information in the table shows the base memory usage for the OS, bootloader, and filesystem. Additional memory would be necessary to set up frame buffers for the image sensor data and display. The frame buffers are dependent on image sensor size and the region of interest for video analysis. Optimization of Flash and RAM usage can be made to further reduce the size of the basic OS setup.

4.2.2 Live View Template Software

A base demonstration application can reduce the learning curve for users with examples on functional code. Two such templates will be available. Both consist of the basic image capture process from image sensor interface, storage and retrieval from memory, and transfer to an output device.

One template runs directly on the PowerPC embedded

Table 3. Peripherals and Bus Components

Module Name	FPGA Slices	Module Name	FPGA Slices
DCR	1	WatchDog	89
JTAG	1	BRAM wrapper	262
PLB bus	264	DDR Controller	824
OPB bus	72	LVDS module	155
Bus bridge	677	Ethernet	2708
GPIO	148	Image Sensor IF	314
Flash Controller	190	LCD Controller	187
Compact Flash	685	Interrupt Controller	134
Reset	39	Clock Generator	40
Timer	269	UART	303
Subtotal	**7362 (29%)**		

Table 4. Generated hardware accelerators

Streaming Kernels	FPGA Slices	Throughput (Bytes/cycle)	Frequency (MHz)
Binarization	189	1	218
Open Filter	731	0.15	188
Edge Detection	1677	0.07	174
Quantization	236	2	219
Column DCT	1148	1.23	171
Row DCT	1129	0.94	149
(LPF) Color Processing	2687	1.1	121
(HPF) Color Processing	2529	1.33	177

in the Virtex 4 FPGAs, without Linux or any other OS, and therefore does not use the Linux device driver. This software displays the sensor image on a monitor via the video connector. Figure 4 shows this live-view process and the use of frame buffers in memory. The buffer sizes depend on the sensor (e.g. number of pixels multiplied by the bytes per pixel). For the video controller, the buffer size is VGA (640x480 pixels) at 4 bytes per pixel. Excluding the various image buffers, the live-view template runs under 64KB, which includes stack space, string constants, and other compiled items. For more advanced video analysis, multiple frame buffers that are sized to a smaller region of interest may be used for different hardware accelerators. For example, separate frame buffers can be allocated for different hardware accelerators to perform DCT and low-pass filters.

The second template is a simple client-server application. This software runs on Linux and interacts (via TCP) with a separate Java program that executes on a PC or other remote computer. The template software captures sensor frames via the Linux device driver and transmits the image data over a network. The Java software receives this data and renders it on the PC's display. It also allows the user to

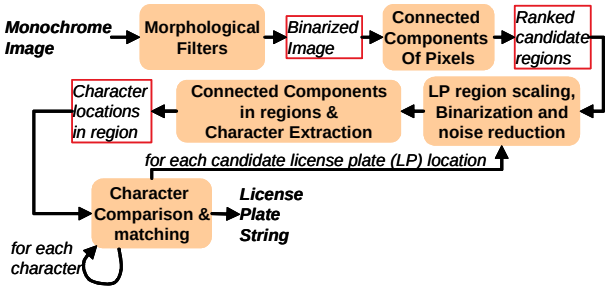

Figure 5. License Plate Recognition (LPR) is a series of kernels separated by decision logic

interactively view and modify the values of all sensor configuration registers.

4.3. Application Mapping

This section presents results from mapping an SoC onto the Watson platform using the design flow as described in Section 3. The intent of this section is to show the utilization of the FPGA for a basic design, as shown previously in Figure 1b. There are many other configurations with different system performance that can be explored in the design using streaming hardware accelerators. Table 3 shows the FPGA slices obtained from synthesis results for different peripherals and bus components. The list is shown for a single PowerPC (PPC) core design, subtotaling 29% of the total number of gates in the Xilinx Virtex XC4VFX60-11. For dual PPC core designs, additional peripherals and bus components would be needed, consuming approximately 42% of the total number of gates. Table 4 shows the synthesis results for various streaming kernels that can be mapped to hardware accelerators. The results are shown for a configuration consuming the minimum amount of area. Other configurations, derived from unrolling the loops and increasing the number of functional units, can be also be generated.

Several video analysis applications using computer vision methods can be accelerated in hardware with improved speedups in performance over a single scalar processor. For instance, in a lane departure warning application, hardware acceleration of the image warping and hough transforms can speed up the performance six-fold. In a license plate recognition (LPR) application, morphological filters that are mapped to streaming hardware accelerators can improve its frame-rate by approximately 1.18 times over a single ARM946ES (16KB I/D cache, 100 MHz) processor. As shown in Figure 5, the morphological filters are only one of many in a series of computation kernels in the entire application. Additional speedups are anticipated when other portions of the application, currently allocated to the scalar core, are accelerated in hardware. The current design process allows the user to select the portion of the processing chain to accelerate by coding the kernels in stream data flow graphs

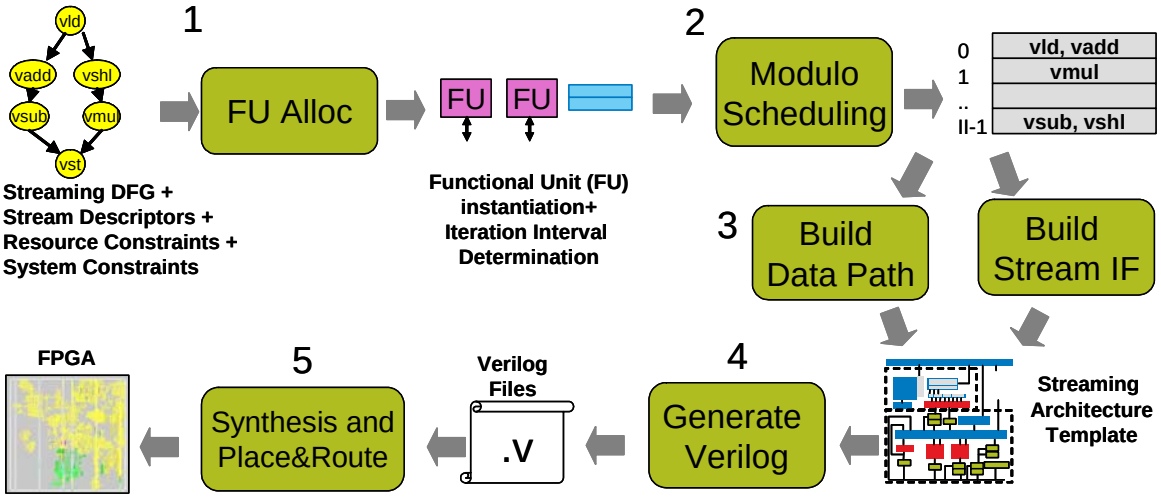

Figure 6. Example design flow (as a sequence of steps) to generate streaming hardware accelerators

```
.DataPathBegin
// Functional Units
fu(adder0, adder, 1);
fu(logic0, logic, 1);
fu(sin1, InStream, 1);
 ... more ...
// Functional Unit Slices
sfu(adder0.0, {vsub, vabs, vnop}, {32,32,16}, 1, 1);
sfu(logic0.0, {vmin, vmax, vif, vnop}, {16,16,32,16}, 1, 1);
sfu(sin1.0, {vld, vnop}, {16}, 0, 1);
 ... more ...
// Line Queues
queue(Qadder0_0_0, adder0.0, [31..0], 1);
queue(Qlogic0_0_0, logic0.0, [15..0], 1);
queue(Qsin1_0_0, sin1.0, [15..0], 1);
 ... more ...
.DataPathEnd
.ControlPathBegin
cstep(3)
//
// Operations for each Functional Unit
ctl_ops(adder0, {{vsub.s32.u32.u16}, {vnop}, {vabs.u32.s32.u16}});
ctl_ops(logic0, {{vmax.u16.u16.u32.u16}, {vnop}, {vmin.u16.u16.u32.u16}});
ctl_ops(sin1, {{vld.u16}, {vnop}, {vnop}});
 ... more ...
// Operands for each Functional Unit Slice Input
ctl_opnds(adder0.0.A, {{Qlogic2_0_0.0}, {vnop}, {Qadder0_0_0.0}});
ctl_opnds(adder0.0.B, {{Qlogic3_0_0.0}, {vnop}, {vnop}});
ctl_opnds(logic0.0.A, {{Qsin1_0_0.0}, {vnop}, {Qlogic0_0_0.0}});
ctl_opnds(logic0.0.B, {{Qsca_z15_0_0.0}, {vnop}, {Qsca_z0_0_0.0}});
ctl_opnds(logic0.0.C, {{vnop}, {vnop}, {vnop}});
 ... more ...
// Controls for each Queue
ctl_queue(Qadder0_0_0, {{1}, {0}, {1}});
ctl_queue(Qlogic0_0_0, {{1}, {0}, {1}});
 ... more ...
.ControlPathEnd
```

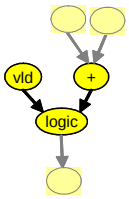

Figure 7. Data path template description

(sDFG). Readers are referred to [18] for more details of the hardware acceleration of the LPR application.

Figure 6 illustrates the example design flow as a sequence of five steps. The sDFG, stream descriptors, and other resource constraints (such as maximum gate count and maximum bandwidth) are used by a compiler to allocate a set of functional units (step 1). Then using modulo scheduling techniques similar to [27,28,29], a sequence of events are arranged so that the functional units can operate properly (step 2). The sequence of events is similar to a series of VLIW (very long instruction word) operations. The streaming hardware accelerator, consisting of a data path and stream unit, is then selected (step 3). An interim hardware description file is used to describe the components within the accelerator. As shown in Figure 7, sDFG nodes and arcs have been associated with hardware resources such as functional units and queues. A set of state machines is also listed to generate the proper control signals. The hardware is then generated (step 4) and synthesized (step 5) into the FPGA.

The performance numbers provided in this paper are for a nominal design configuration within the system-design space. Given larger gate resources, the performance can improve by leveraging the inherent task and data parallelism of the applications. With the design flow and Watson platform described in this paper, an application designer can easily explore the different hardware accelerator configurations and prototype their computer vision methods quickly. Using the streaming hardware accelerators, the hardware platform and system software described in this paper form an integrated smart camera development kit.

5. CONCLUSION

This paper presents a streaming architecture template and development platform dedicated for embedded computer vision applications. The design process is a part of a design automation flow whereby hardware accelerators are used to execute computationally intensive kernels. In addition, the memory hierarchy is configured to support automatic data prefetching and alignment. The software architecture and live-view demonstration template are shown with reports on the memory usage and gate-count consumption on the FPGA. With this design process, application designers can now focus on application development rather than the complicated

design process.

There are many future research activities to fully automate the design process. A set of compiler tools are being investigated to automatically select user kernels for acceleration. Additional parameters for the stream descriptors are considered to describe a moving region of interest within an image window.

References

[1] M. Bramberger, A. Doblander, A. Maier, B. Rinner, "Distributed Embedded Smart Cameras for Surveillance Applications," *Computer*, February 2006, pp. 68-75.

[2] W. Wolf, B. Ozer, T. Lv, "Smart Cameras as Embedded Systems," *Computer*, September 2002, pp. 48-53

[3] W.J. MacLean, "An Evaluation of the Suitability of FPGAs for Embedded Vision Systems", *Computer Vision and Pattern Recognition (CVPR)*, vol 3, June 2005, pp.131-138.

[4] P. Ranganathan, S. Adve, N.P. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions," *Proceedings of the 26th Annual International Symposium on Computer Architecture* (ISCA'99), May 1999, pp. 124-135.

[5] S.P. Amarasinghe; W. Thies, "Architecture, languages and compilers for the Streaming Domain," PACT 2003 Tutorial, http://cag.lcs.mit.edu/wss03/.

[6] M. Bedford, et al, "Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams," *Proceedings of the 31st Annual International Symposium on Computer Architecture* (ISCA'04), June 2004, pp. 2-14.

[7] S. Rixner, W.J. Dally, U. J. Kapasi, et. al., "A bandwidth-efficient architecture for media processing," *Proceedings of the 31st annual ACM/IEEE International Symposium on Microarchitecture*, November 1998, pp. 3-13.

[8] W.J. Dally, P. Hanrahan, M. Erez, et. al., "Merrimac: Supercomputing with streams", *Proceedings of the SuperComputing SC'03 Conference*, November 2003, Phoenix, Arizona, pp. 35-43.

[9] S. Chiricescu, R. Essick, B. Lucas, et. al., "The Reconfigurable Streaming Vector Processor (RSVP™)," *Proceedings of the 36th International Symposium on Microarchitecture*, December 2003, pp. 141-150.

[10] S. Chai, S. Chiricescu, R. Essick, et. al., "Streaming Processors for Next Generation Mobile Imaging Applications," *IEEE Communications Magazine*, vol 43, no 12, Dec 2005, pp. 81-89.

[11] E. Caspi, M. Chu, R. Huang, et. al, "Stream Computations Organized for Reconfigurable Execution (SCORE)", Field Programmable Logic (FPL), Villach, Austria, August 2000, pp. 605-614.

[12] O. Mencer, D.J. Pearce, L.W. Howes, W.Luk, "Design Space Exploration with a Stream Compiler", IEEE International Conference on Field Programmable Technology (FPT), Tokyo, December 2003, pp.270-277.

[13] M. Gokhale, J. Sone, J. Arnold, M. Kalinowski, "Stream-Oriented FPGA Computing in the Streams-C High Level Language", Field Programmable Custom Computing Machines (FCCM), pp 49-56, 2000.

[14] D. Bursky, "Spring 'board' to FPGA design success," *Electronic Design*, vol 54, no 3, Feb 16, 2006, pp.53-62.

[15] M. Vidiu, G. Venkataramani, T. Chelcea, S.C. Goldstein. Spatial Computation. Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Boston, MA, October 9-13, 2004, pp. 14- 26.

[16] N. Bellas, S. Chai, M. Dwyer, and D. Linzmeier, "Template-Based Generation of Streaming Accelerators from a High Level Representation," Field Programmable Custom Computing Machines (FCCM) poster, April 2006, 2 pages.

[17] V. Kathail, S. Aditya, R. Schreiber, B. Ramakrishna Rau, D.C. Cronquist, M. Sivaraman, "PICO: automatically designing custom computers", *Computer*, vol 35, no 9, Sept. 2002, pp. 39 – 47.

[18] S. Chai, N. Bellas, M. Dwyer, D. Linzmeier, "Stream Memory Subsystem in Reconfigurable Platforms", Workshop on Architecture Research on FPGA Platforms, Austin, TX, February 2005, 4 pages.

[19] N. Bellas, S. Chai, M. Dwyer, and D. Linzmeier, "FPGA implementation of a license plate recognition SoC using automatically generated streaming accelerators," Reconfigurable Architecture Workshop, Rhodes Island, Greece, April 2006, 4 pages.

[20] M. Sen, I. Corretjer, F. Haim, et. al, "Computer Vision on FPGAs: Design Methodology and its Application to Gesture Recognition", *Computer Vision and Pattern Recognition (CVPR)*, June 2005, pp. 133-140.

[21] S. Chai, A. López-Lagunas, "Streaming I/O for Imaging Applications", IEEE International Conference on Computer Architecture for Machine Perception, Palermo, Italy, July 2005, pp. 178-183.

[22] A. López-Lagunas, S. Chai, "Memory Bandwidth Optimization through Stream Descriptors", Memory Performance: Dealing with Applications, Systems and Architecture (MEDEA) Workshop, St. Louis, September 2005, pp. 59-66.

[23] J. Marathe, F. Mueller, T. Mohan, et. al., "METRIC: Tracking down inefficiencies in the memory hierarchy via binary rewriting", *International Symposium on Code Generation and Optimization*, March 2003, pp. 289-300.

[24] P. Havlak, K. Kennedy, "An implementation of interprocedural bounded regular section analysis", *IEEE Transactions on Parallel and Distributed System*, vol. 2, no. 3, July 1991, pp. 350-360.

[25] Xilinx Inc, "ML401/ML402/ML403 Evaluation Platform Users Guide", www.xilinx.com/bvdocs/userguides/ug080.pdf

[26] Micron Technology Inc, "1/3-Inch, Wide-VGA CMOS Digital Image Sensor", "640H x 480V, Ultra Low-Power, CMOS Digital Image Sensor Camera System-on-a-Chip", "3-Megapixel CMOS Digital Image Sensor", http://micron.com/products/imaging/products

[27] S. Ohm, F. Kurdahi, N. Dutt, "A unified lower bound estimation technique for high-level synthesis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol16, No. 5, May 1997, pp. 458-472.

[28] J. Jeon. D. Kim, D. Shin, K. Choi, "High level synthesis under multi-cylcle interconnect delay", *Proceedings of the 2001 conference on Asia South Pacific design automation*, 2001, pp. 662-667.

[29] S. Tarafdar and M. Leeser, "The DT-model: high-level synthesis using data transfers**",** *Proceedings of the 35th annual conference on design automation*, 1998, pp. 114-121