Flexible Function-Level Acceleration of Embedded Vision Applications using the Pipelined Vision Processor

Robert Bushey
Embedded Systems Products and Technology
Analog Devices Inc. (ADI)
Norwood (MA), USA
Email: robert.bushey@analog.com

Hamed Tabkhi
Departement of Electrical and Computer Engineering
Northeastern University
Boston (MA), USA
Email: tabkhi@ece.neu.edu

Gunar Schirner
Departement of Electrical and Computer Engineering
Northeastern University
Boston (MA), USA
Email: schirner@ece.neu.edu

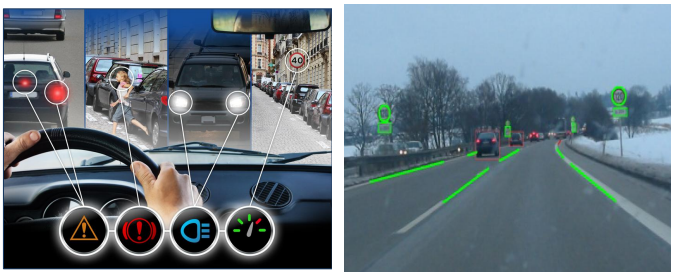
***Abstract*—The emerging massive embedded vision market is driving demanding and ever-increasing computationally complex high-performance and low-power MPSoC requirements. To satisfy these requirements innovative solutions are required to deliver high performance pixel processing combined with low energy per pixel execution. These solutions must combine the power efficiency of ASIC style IP while incorporating elements of Instruction-Level Processors flexibility and software ecosystem.**

This paper introduces Analog Devices BF609's Pipelined Vision Processor (PVP) as a state-of-the-art industrial solution achieving both efficiency and flexibility. The PVP incorporates over 10 function level blocks enabling dozens of programmable functions that can be allocated to implement many algorithms and applications. Additionally, the pipelined style connectivity is programmable enabling many temporal function permutations. Overall, the PVP offers greater than 25 billion operations per second (GOPs) and very low memory bandwidth. These capabilities enable the PVP to execute multiple concurrent ADAS, Industrial, or general vision applications. This paper focuses on the key architecture concepts of the PVP from individual function-block construction to the allocation and chaining of functional blocks to build function based application implementations. The paper also addresses the benefits and challenges of architecting and programming at the function-level granularity and abstractions.

I. Introduction

Embedded vision computing is recognized as a top tier, rapidly growing market. Embedded vision refers to deploying visual capabilities to embedded systems for better understanding and analysis of two and three-dimensional surrounding environments [1]. Market examples are Advanced Driver Assistance System (ADAS), industrial vision and video surveillance. ADAS market forecasts call for a global value growth from $10 billion in 2011 to $130 billion in 2016 [2] (13-fold projected growth). Examples of ADAS applications are pre-crash warning and/or avoidance, lane departure warning (LDW), traffic sign recognition (TSR), and general object classification, tracking and verification (highlighted in Fig. 1).

Vision Processing and Video Recognition increase the demand for extremely high performance coupled with very low power. With a frame rate of 30FPS and a high definition resolution like 1280x960 Pixels/Frame, 37 million safety-critical pixels per second stream real-time into the front-end of an ADAS system. Many concurrent operations drive the pixel compute complexity well into the many billions of pixel operations per second (GOPs) range. Vision processing systems must tackle these extreme compute intensive challenges while consuming very little power (often less than 1W). Additionally, these solutions must be offered at remarkably low price points. With these requirements and constraints, embedded vision architects face many challenges as they set out to architect and implement vision processing solutions.



(a) Intelligent Automotive control system.

(b) Lane Departure Warning (LDW).

Figure 1: ADAS market applications

New, innovative architecture solutions are required to efficiency realize vision algorithms. Solutions solely based on Instruction-Level Processors (ILPs) burn too much energy per operation often making a software-only solution infeasible. Conversely, ASIC style IP approaches can yield very high performance and great power efficiency but lack the highly desirable flexibility and software eco-system offered by ILPs. As a result, we have taken an architectural approach composing our heterogeneous multi-processor system on chip (MPSOCs) out of flexible and programmable ASIC style IP combined with one or more ILPs.

Embedded vision applications are comprised of a vast and continuously evolving reservoir of algorithms. Designing

dedicated hardware IP for individual applications is cost prohibitive due to very expensive (and rising) fabrication & mask Non-Recurring Engineering (NRE) costs. At the same time, a proper architecture solution needs to provide enough performance and efficiency for each application within any targeted market, increase productivity and reduce the overall design and development costs. This dramatically increases the pressure for new, innovative solutions that strike the balance of performance, flexibility and efficiency.

In this paper, we introduce the Pipeline Vision Processor (PVP) within Analog Devices Blackfin BF60x [3]. The PVP provides a power and performance efficiency comparable to custom hardware design as well as flexibility within embedded vision domain with special focus on Advanced Driver Assistance System (ADAS) systems. The PVP is a composition of eleven programmable function blocks. The PVP blocks are connected through a customized MUX-based interconnection to create a macro datapath for many vision algorithms. This article overviews the architecture concepts of PVP from function-block construction via function block chaining to PVP system integration into an MPSoC.

The remainder of this paper is organized as following. Section II briefly reviews the related work. Section III outlines the PVP decomposition. Section IV provides the detail regarding the different PVP architectural aspects. Following that, Section V outlines the PVP potential for the research community. Finally, Section VI concludes this paper.

II. Related Work

With the demand for power-efficient high-performance architectures, significant research effort has been invested into utilizing hardware accelerators and building heterogeneous architectures. Accelerator-Rich CMPs [4], Accelerator-Store [5], Platform 2012 [6], QSCORE [7] are a few examples. Hybrid ILP with offload hardware accelerator engines are attractive for many markets, such as the IBM power EN for cloud computer systems [8], or Texas Instruments (TI) DaVinci platform [9] with specialized vision/video processing accelerators. These approaches demonstrate significant performance improvement while reducing energy per operation through execution of application hot-spots on the custom hardware. However, in contrast to the PVP concept, the hardware accelerators are considered as co-processors instead of autonomous processing elements. As such the accelerators always depend on the main-ILP for operating on data streams.

III. PVP Overview

This section overviews the architecture of Analog Devices' Pipeline Vision Processor (PVP) [3]. Analog Devices' BF60x SoC combines two ILP Blackfin DSP cores with the PVP as part of the video subsystem (VSS) for high-performance vision analytics as shown in Fig. 2. The PVP is an autonomous heterogeneous processing node with direct access to the system memories. Many different pixel pipelines can be composed with the VSS leading to very efficient application realizations with low memory subsystem bandwidth.

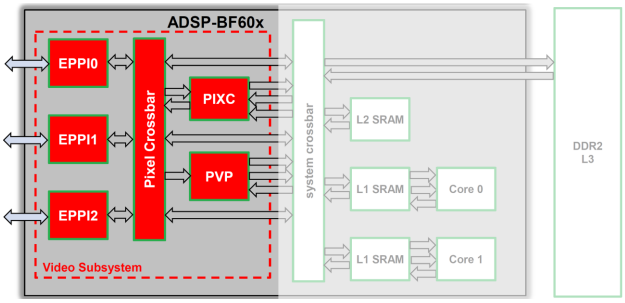


Figure 2: PVP system integration

PVP offers more than 25 billion operations per second (GOPs) with very low memory bandwidth and configurable macro datapaths. PVP operates on frame rates up to 1280x960x30Hz (16bits) and supports multiple concurrent applications across several embedded vision markets (through concurrent pipelines). Representative applications (potentially concurrent) include: lane departure warning (LDW), traffic sign recognition (TSR), high-beam/low-beam (HBLB), object/pedestrian identification and tracking, robotic and machine vision.

Fig. 3 shows a coarse-gain decomposition of the PVP including number and type of function blocks and their connectivity. The PVP contains eleven function blocks (FBs): four 2D convolution (CNV), arithmetic unit (ACU), polar magnitude and Angle (PMA), 2D pixel-edge classifier (PEC), two threshold-histogram-compression (THC), two 2D integral image (IIT). Some FBs have multiple instances (e.g. four CNV blocks, CNV is part of many algorithms). Furthermore, the PVP supports input and output stream formatters to aid in receiving input pixels and writing results to the memory subsystem.

PVP offers customized FB-to-FB communication requiring no interaction with the system memory. The communication is MUX-based with support for runtime re-configurability. FB-to-FB communication within PVP has been architected based on typical flows across target applications. After initial configuration, the PVP operates independently on the input data streams like an autonomous processor core. Concurrent applications run in parallel data pipelines. The PVP includes multiple DMAs for data streams. The PVP can fetch block and datapath configuration independently for on-the-fly re-configuration without ILP core intervention.

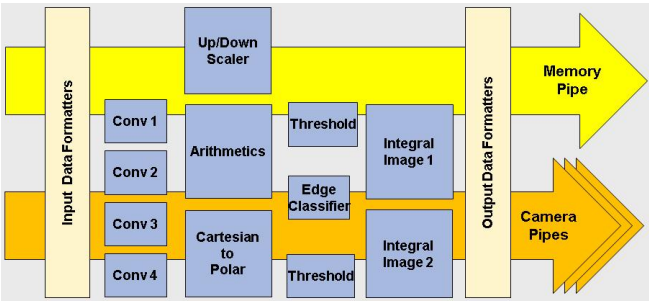


Figure 3: PVP overview

IV. PVP Architecture

This section discusses the PVP in more detail, including: block level exploration, inter-block communication, PVP control and memory access, system integration and finally software level abstraction.

A. Function Blocks

Analog Devices has identified function blocks based on market requirements taking into account general vision algorithms and applications requirements, and incorporated customer feedback and general vision trends. The blocks have been initially targeted across several key vision markets and support a rich set of edge detection and feature extraction algorithms. This section summarizes the functionality of the primary blocks.

Input Formatter (IPF): IPFs can recieve data directly from the video input interface (the Enhanced Parallel Peripheral Interface (EPPI)) and from memory through DMA. IPFs incorporate pre-processing including color or luminance components extraction, pixel windowing, frame counting, and control the frame processing.

2D Convolution (CNV): PVP features four convolution blocks. Convolution blocks support 2D convolution for varying pixel ranges (1x1, 3x3, 5x5) and up to 16-bit coefficients. CNV block coefficients can be configured to realize Gaussian image smoothing, and calculating the first and second derivatives of pixel ranges.

Arithmetic Unit (ACU): supports general purpose integer operations (ADD, SUB, 32-bit multiply, 32-bit divide), shift and logical operations, as well as internal 37-bit ACC and Barrel shifting scaled to 32-bit results. Having the ACU inside PVP avoids extra interaction with the host ILP. The ACU also supports multiple pipelined arithmetic operations to aid in the pixel pipeline algorithm mapping.

Polar Magnitude and Angle (PMA): converts two 16-bit signed inputs in Cartesian format (x,y) into Polar form (Magnitude, Angle). The PMA can be employed to identify non-zero pixel crossing in many edge detection algorithms.

Pixel-Edge Classifier (PEC): PEC supports edge detection including: non-linear edge enhancement filtering in a pixel neighborhood, edge classification based on orientation, sub-pixel position interpolation, vertical/horizontal sub-pixel position into one byte per pixel. PEC operates either in first derivative mode (PEC-1) or second derivative mode (PEC-2).

Threshold-Histogram-Compression (THC): PVP features two THC blocks that implement a collection of statistical and range reduction signal processing functions including pixels classification, rounding up to nearest threshold and finding maximum values.

Integral Image (IIT): Two IIT blocks calculate a 2-dimensional (2D) integral over the input pixels window. Alternatively, IIT computes the horizontal 1D sum.

Output Formatter (OPF): PVP features four OPF Blocks, three assigned for the video pipe and one for the memory pipe. The output formatters receive the data results from the PVP processing blocks and apply final formatting. For each OPF

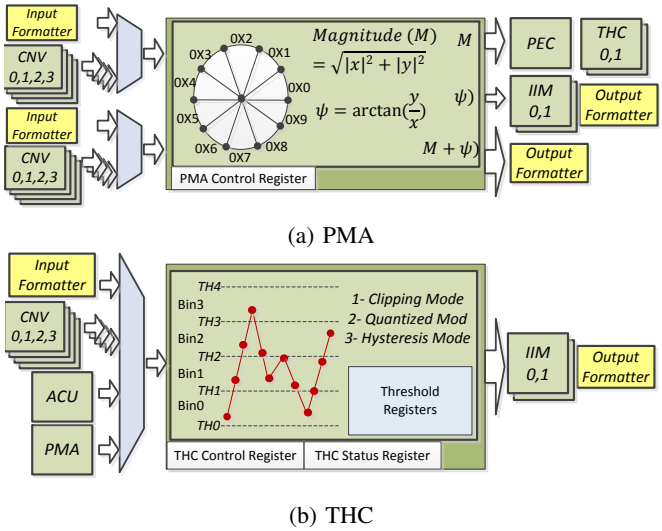


Figure 4: The structures of PMA and THC functions.

an individual DMA channel is assigned for writing the results to memory.

As indicated, each function block can operate in different operating modes varying on input and output. To control the behavior, each block contains control and configuration registers. They adjust the performed computation giving each function block a specific personality, thus enabling construction of different macro pipelines. Some blocks (e.g. CNV, IIT) contain internal data buffers to store the required data structures to be operated on by the block. With the 2D data nature, these buffers store a few pixel rows before performing the operation (e.g. 1D and 2D derivatives). Although, the block internal buffers introduce an initial latency, the PVP has a throughput of one pixel per cycle.

To visualize more detail, Fig. 4 illustrates the structure of two function blocks: PMA and THC. The PMA block (highlighted in Fig. 4a) translates from Cartesian coordinates (x, y) to polar coordinates (angle ψ, magnitude M). PMA receives Cartesian coordinates (16 bit signed each) from two inputs, and produces three outputs: M (16 bits), ψ (5 bits) and $M + \psi$ (21 bits). Furthermore, the PMA block offers control registers to control the operations and the desired output ports on a frame by frame basis.

The *THC* function block is shown in Fig. 4b. *THC* has more functional diversity than *PMA*, and offers three operation modes: clipping, quantification and hysteresis modes, selectable through control registers. *THC* also offers variable thresholds (programmable at run-time) operating on histogram values. The *THC* realizes basic statistical and control flow operations (if-

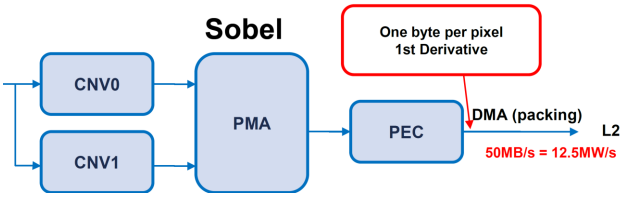


Figure 5: Block allocation for Sobel edge detection

then-else conditions). It has low computational complexity and thus is not an intuitive choice for HW acceleration. However, by adding *THC* blocks to PVP, it allows for keeping the computation/communication local, avoiding unnecessary interactions with an ILP for simple operations, and thus significantly contributes to efficiency.

Function blocks can be composed into a macro pipeline for realizing applications. Fig. 5 shows a simple allocation for a Sobel Edge Detection algorithm [3]. Two CNV blocks calculate horizontal and vertical components of the first derivative gradients in parallel. Their outputs are routed to the *PMA*. At the output of *PMA*, the non-zero crossing pixels are potential edge pixels. To identify the actual edge pixels *PEC* processes the *PMA* output for edge classification. In the next subsection, we describe how the PVP realizes highly efficient block-to-block communication. ¿¿¿¿¿¿ .r1279

B. Interconnecting Function Blocks

Function blocks in the PVP are connected through a customized block-to-block MUX-based interconnect. In result, the transferred data among PVP block remains local, not hitting the memory subsystem and thus significantly reducing system memory bandwidth. A run-time configurable MUX-based interconnect realizes the PVP block-to-block communication. Input data to individual ports can be routed from different source function blocks selected by input MUXs. The connectivity provides a number of ways to connect block outputs as inputs to other blocks. In this way, the blocks can be directly fused together. For example, *PMA* (Fig. 4a) can get input data both from the input formatter or the CNV blocks. Only selected communication paths are realized where semantically meaningful. For example, *PMA*'s magnitude output port only connects to *THC* and *PEC*.

Fig. 6 outlines the function block connectivity between the PVP blocks. The block-to-block communication paths have been architected based on typical vision flows across many applications. The pipeline structure is composed by programming multiplexers from the output of blocks to the input of other blocks. Instead of an all-to-all connectivity among all function blocks, PVP offers a selective connectivity derived by the possibility of application decomposition. For the purpose of connectivity organization and efficient PVP-internal pixel pipe connectivity allocations, the blocks have been grouped into five stages. The first and last stages belong to *IPF* and *OPF* respectively, the remaining blocks spread between three middle stages: Convolution blocks (*CNV*) in second stage, *ACU*, *PMA* and *PEC* in the third stage, and *IIM* and *THC* in the fourth stage. The PVP offers connection between blocks within the same stage as well as forward connection including bypassing single our multiple stages.

C. PVP Control and Configuration

After initial configuration, the PVP operates independently as an autonomous processor on the input/output data streams. Furthermore, the PVP supports runtime application switching at granularity of each new image frame. The PVP can be

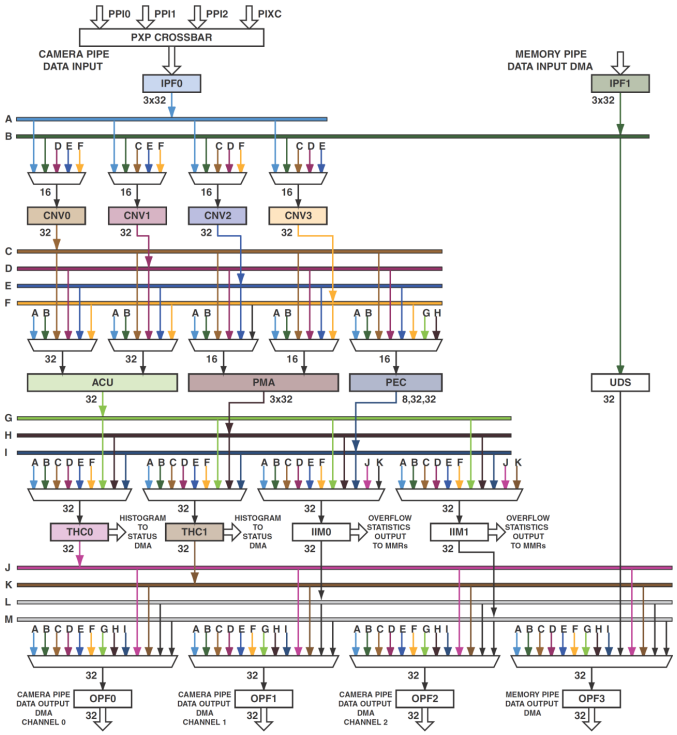


Figure 6: The PVP programmable MUX-based block to block connection.

configured in two different ways: register-based and DMA based programming.

In register-based programming, the host ILP programs the PVP by writing the configuration data into Memory Mapped Registers (MMRs). The MMR registers provide the configuration knobs for individual blocks (e.g. CNV, THC) as well as construction of the PVP datapath (interconnecting the PVP function blocks). After configuration, the desired PVP operation starts with the next coming frame.

In DMA-based programming, the PVP fetches configuration data through specialized configuration DMA channels. The DMA-based method is derived from the descriptor-based programming method. For the PVP, it does not make any difference whether a memory-mapped register has been written by the host ILP (MMR-based programming) or written by configuration DMA. By deploying DMA-based programming, the PVP can operates autonomously and even be reconfigured on-the-fly independent from the ILP core interaction.

To support on-the-fly reconfiguration, PVP employs the double buffering techniques for the MMRs. While PVP is working on its current frame, the new configuration data can be written to the PVP MMRs. The reconfiguration is applied by the end of processing current frame and before starting the processing of next coming frame. In result, new configuration are written any time and will be properly synchronized with pipe progress by hardware. The data structure for configuration data is called the block configuration structure (BCS). The PVP control DMA supports fetching and storing of multiple BCSs to facilitate runtime PVP datapath reconfiguration; comparable to multi-thread scheduling and execution in ILP cores.

D. PVP System Integration

Analog Devices has paired the PVP with two Blackfin DSP cores to expand both flexibility as well as efficiency in creating more complex applications (entire vision flow). In fact, the PVP adds a new degree of heterogeneity to the Blackfin cores only based SoCs. The integration supported by large on-chip memory (4.3Mbit SRAM) to keep the data on-chip as much as cost and area allows. Highly efficient system bandwidth features a rich peripheral set and connectivity options including ADIs Enhanced Parallel Peripheral Interfaces (EPPI) specialized for streaming input pixels from the video camera.

In the current integration, the PVP often performs the more compute intensive portion of applications which map well to the PVP blocks, and the Blackfin cores mainly execute the higher level analysis (which is ILP suitable). Miscellaneous functions that are not supported by PVP blocks also have to be mapped to the system ILPs. Input/output streaming through DMA channels allows interfacing between the Blackfin cores and the PVP to create more complex applications. Much of the interactions occur through the interrupts of the PVP DMA channels. For example, the DMA completion interrupt signal announces that the data output DMA stored all results in system memory.

The PVP supports two architecture concepts, the camera pipe and the memory pipe at the same time (outlined in Section III). The camera pipe directly receives the input pixels from camera interface without requiring any processor interaction. This offers efficient stream processing, as the overhead of reading input pixels from system memory has been removed. With direct access to the I/O interfaces, it is conceivable to build a complete application only out of function blocks in the PVP. The PVP supports up to three parallel camera pipes with their own dedicated datapaths and it can support up to three concurrent application kernels. However, the individual PVP blocks can be only mapped to one of the concurrent datapaths.

Conversely, in the memory pipe mode the input data is fetched from system memory using streaming DMAs. At each point of time, PVP can only execute one application in the memory pipe mode. However, camera pipes and memory pipes can operate concurrently. The pipes have separate control mechanisms, separate datapaths, and operate with independent timing. Please note that each PVP block may be assigned only to one pipe at a time. But, there is support for dynamic reconfiguration and allocation of PVP blocks.

Fig. 2 presents how a combination of PVP and Blackfin cores provide an efficient solution for road sign detection – highly demanded by the ADAS market. The application is composed of four algorithm kernels: monochrome, edge map, region of interest, scale and deskew. Both memory pipe and camera pipe operate concurrently. During pre-processing pixels are received directly from the camera or sensor, translated from RGB to monochrome and fed to the PVP (camera pipe)for performing edge detection algorithm. Following that, the PVP writes back the edge map data to system memory. During mid-processing the Blackfin cores identify the region of interest (road sign) and separate it from the whole scene. After that, the PVP is again in charge for deskew processing of objects slanting too far in one direction. But this time, the PVP operates in memory pipe mode; only region of interest data is read from memory for the purpose of deskewing. The output of PVP is written back to memory for final post-processing (e.g. template matching) on the Blackfin cores.

E. PVP software Stack

One key focus to ease adoption is to simplify PVP usage and programming. ADIs Image Processing Toolbox provides hundreds of optimized functions for image and vision

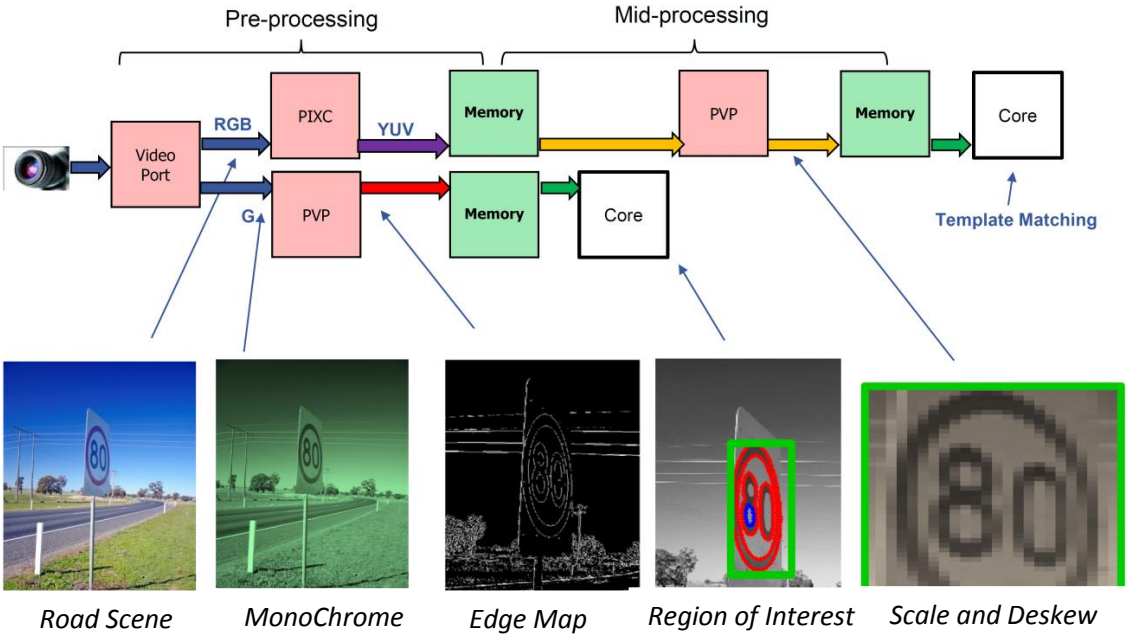


Figure 7: PVP-based Heterogeneous processing for Road Sign Detection.

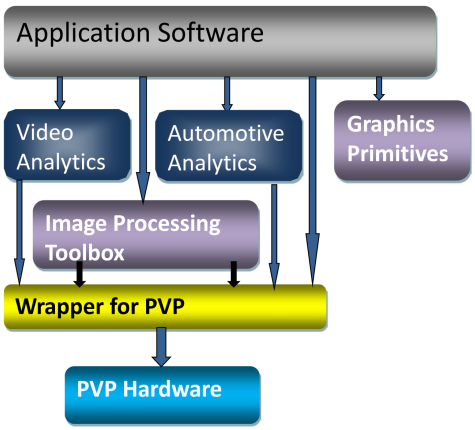


Figure 8: The PVP software stack.

analysis including operations like histogram, morphological transformations and 2D convolution. Video and automotive analytics toolboxes are also available to support vision related applications. Where appropriate the software tools will map functions to the PVP hardware to enable greater performance and power efficiency vs. utilizing the DSP ILPs.

C and HW function-level APIs raise the level of programming abstraction and efficiency thus further easing the programmers burden. And, once again, many familiar API calls are mapped to the PVP hardware reducing the amount of required PVP configuration. For example, many OpenCV like APIs are supported by the software tools and thus aid in the PVP programming and mapping process.

V. PVP Road-map

The BF609 MPSoC generally, and the PVP specifically, offer a new blend of flexibility (e.g. through ILPs and configurable PVP functionality) as well as extreme performance and energy efficiency traditionally provided only by ASIC style IP all in one package. Additionally, the PVP raises the level of abstraction on both the hardware and software sides, thereby improving overall application development productivity.

The PVP offers more than 25+ billion operations per second while consuming less than 200mW. Compared to ILPs, in addition to reducing the overhead per operation (by raising granularity of programmability), the PVP also significantly reduces the volume of memory subsystem accesses through efficiently routing data within PVP. At the same time, compared to ASIC style hardware IP, the PVP offers much more flexibility by offering concurrent programmable macro datapaths.

New research opportunities arise with the PVP, for architecting and programming at a higher level of functional granularity. The efficiency of a PVP-based architecture relies on the early identification of meaningful function blocks and their possible compositions, basically defining flexibility and usability. New research is needed shifting from optimizing individual applications to identifying common functions within many applications across markets. The challenge is to define a small enough set of sufficiently composable functions that provide meaningful computation for a given market. One approach is to analyze existing programming frameworks (e.g. OpenCV) for frequently used functional primitives (as candidates to be function blocks) and their composition.

At the same time, raising the programming abstraction above instructions opens opportunities to construct frameworks that simplify programming and utilization. Of particular interest are the allocation of function blocks (i.e. selecting from a user specification) and the simplified configuration of function blocks. While this hides computation complexity, previously drowned challenges appear in composition and communication. As such, bandwidth management and traffic policing is needed to facilitate real-time operation. Scheduling aspects can be studied when multiple applications concurrently execute on a PVP-style architecture (static scheduling currently supported). However, function blocks could dynamically join different data streams, which demand a dynamic scheduling, offering challenges around context switching and related policies.

VI. Conclusions

This paper introduces the Pipeline Vision Processor (PVP) which is integrated on Analog Devices Blackfin BF60x. The PVP was designated and fabricated for high-performance embedded vision processing and is intended to support a broad array of embedded vision applications across several key markets including Automotive and Industrial areas. The PVP has been architected and is programmed at a function-level granularity offering both efficiency and flexibility. While offering more than 25 billion operations per second and up to four concurrent applications, the PVP consumes less than 200 mW. This paper outlined the basic architectural features of the PVP: function-block construction, chaining function blocks and PVP system integration. The PVP opens a new area of exploration to both academia and industry for architecting and programming at a function-level granularity and abstraction.

References

[1] J. Bier, "Implementing Vision Capabilities in Embedded Systems," *Berkeley Design Technology Inc.*, Nov. 2012.

[2] ABI Research, "Advanced Driver Assistance Systems Markets to Approach 10 Billion in 2011," Available: http://www.abiresearch.com.

[3] A. D. Inc.(ADI), "ADSP-BF60x Blackfin Processor Hardware Reference Manual," *Reference Guide, Part Number 82-100113-01*, Jun. 2012.

[4] A. Gerstlauer, C. Haubelt, A. Pimentel, T. Stefanov, D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1517–1530, Oct. 2009.

[5] "System Derivers," *International Technology Roadmap for Semiconductors (ITRS)*, 2011.

[6] J. Keinert, M. Streub&uhorbar;hr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith, "Systemcodesigner: an automatic esl synthesis approach by design space exploration and behavioral synthesis for streaming applications," *ACM Transaction on Design Automation of Electronic Systems*, vol. 14, no. 1, Jan. 2009.

[7] Mathworks, "Simulink Getting Started Guide," *International Technology Roadmap for Semiconductors (ITRS)*, R2012b.

[8] L. Indrusiak, A. Thuy, and M. Glesner, "Executable system-level specification models containing uml-based behavioral patterns," in *Design, Automation Test in Europe Conference (DATE) Exhibition*, Apr. 2007, pp. 1–6.

[9] G. Agarwal, "Get smart with TI's embedded analytics technology," *Texas Instruments (TI) white paper*, 2012. Available: www.ti.com/dsp-c6-analytics-b-mc.