De-Mystifying accelerated Smart Vision Systems with All Programmable SoCs

Mark Timmons (Author)
System Architect
Xilinx ISM (Industrial, Scientific & Medical)
United Kingdom
markt@xilinx.com

Dr Gerhard Lienhart (Author)
Senior Software Engineer
Silicon Software GmbH
Mannheim, Germany
lienhart@silicon-software.de

Abstract—Complex SoC devices, such as the Zynq® All Programmable SoC family from Xilinx®, are being chosen by designers for the next generation of smart and intelligent, embedded smart vision systems. SoCs offer new levels of processing acceleration that were not possible in older multi-chip architectures due to the abundant and tightly coupled connectivity of the ARM® Dual Cortex A9 processing system and the high performance programmable logic. Programming such complex devices can present a challenge for many designers.

This paper describes how high performance platforms can be realized with greatest productivity and minimal FPGA design knowledge.

Keywords—SoC; Smart Vision; low power vision; Zynq; acceleration

I. INTRODUCTION

Higher levels of acquired pixels in high performance camera systems and a drive to more embedded, low power solutions are increasing the number of smart, intelligent cameras that are being designed for machine vision and computer vision applications. A number of market trends are also making the SoC a more desirable solution for the smart vision application space:

a) More pixels at a faster acquisition rate, i.e. need to increase the speed and quality of the image analysis

b) Vision systems applications moving outside the traditional factory floor which drives the need for more embedded, low power, scalable platforms

c) More intelligence being pushed to the edge, i.e. analytics and smart functions being placed in the camera and not a PC

SoCs, such as the Zynq All Programmable SoCs [1], are emerging as an ideal platform for providing scalable, high performance systems taking advantage of the programmable logic fabric for high efficiency parallel pixel processing, with acceleration tightly coupled with the dual core Cortex A9 processing system for frame based processing. Such embedded SoC solutions offer a new level of integrated, low power design that cannot be achieved with other technologies. Single chip Smart Vision SoC solutions reduce the overall power consumption by using the latest low power silicon technologies tightly coupling power efficient ARM® processors and programmable logic. A reduction in system complexity and interconnect is also an advantage in overall system power consumption.

II. WHY THE NEED FOR A SOC SOLUTION AND THE DESIGN CHALLENGES?

In Fig. 1 we can see a diagram depicting the typical building blocks for an Embedded Vision Pipeline that is acquiring the pixel information on the left hand side, pre-processing the captured image before analysing the content using vision based algorithms, and arriving at a decision on the output.

Fig.1. Typical embedded vision pipeline

Such embedded vision algorithms are diverse and complex, but tend to be a series of defined processing stages. Early stages of the image chain process every pixel and tend to use relatively simple algorithms, though the processing rate at such a stage is very high especially in high resolution, high...
frame rate applications. Middle stages of the pipeline deal with features and segments rather than pixels so much lower data rates but much more complex algorithms. Final stages then deal with objects or even video frames so the data rates once again are reduced but the algorithm complexity is furthermore increased. So even in a single vision application we can see that the there is a vast diversity of data types, data rates and algorithms.

If we now consider the relative performance (approximations) of the different design approaches that could be implemented in a SoC device we can clearly see, in Fig. 2, that special attention has to be made in order to take the right implementation approach to get the best performance and highest efficiency.

<table>
<thead>
<tr>
<th>Design approach</th>
<th>RISC Proc</th>
<th>Proc. w/ accels</th>
<th>Folded datapath</th>
<th>Pipelined datapath</th>
<th>Replicated datapath</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock/sample</td>
<td>1000:1</td>
<td>100:1</td>
<td>10:1</td>
<td>1:1</td>
<td>1:10</td>
</tr>
<tr>
<td>Data Rate</td>
<td>200Ks/s</td>
<td>2Ms/s</td>
<td>20Ms/s</td>
<td>200Ms/s</td>
<td>2 Gs/s</td>
</tr>
<tr>
<td>Applications</td>
<td>ARM® A9 processors 1-2 Gops</td>
<td>PL Fabric 10 – 500 Gops</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig.2. Design approaches for different sample, data rates

If we examine the merits of the A9 processors and programmable logic fabric (PL) individually for such processing tasks it can be seen:

- **ARM® Processor Limitations for Pixel Processing**
  - Poor access locality -> small caches perform poorly
  - Generic processors limited in parallel operations and number of cores
  - High number of cores approach consumes too much power (in excess of vision system power budget typically)

- **ARM® Processor Benefits for Low Rate, Complex Code**
  - Can execute large programs, time-shares the ALU
  - Caches take care of memory abstraction with reasonable performance

- **FPGA Limitations for Complex Code**
  - Large programs are labor intensive to code, explicit memory model

- **FPGA Benefits for Pixel Processing**
  - Can do 100 to 1000 operations every clock cycle, without resource sharing
  - Can stream data, and separate between on chip and off chip memory
  - High Level Synthesis and OpenCV libraries for C/C++ programming

It can be seen that the parallel and pipelined processing capabilities of the PL part of a SoC can be leveraged for such front end processing and the ARM® processing system can be used for the complex, lower sample rate algorithmic stages of the design. The programming of an ARM® system is well known to most embedded design teams but producing a SoC PL design can be seen as a challenge in terms of implementing a system with the highest productivity, especially if there is limited FPGA design knowledge in the engineering team. New methods of programming such systems are required that can produce a platform with scalable performance and that can be programmed using abstracted software orientated programming tools and/or languages. This allows designers to target such heterogeneous SoC platforms in a much higher productive manner whilst still meeting high performance targets.

**III. SOFTWARE ORIENTATED PROGRAMMING METHODS**

Abstracted programming methods for programmable logic are becoming established as a more productive means to producing algorithmic functions that can be part of an overall complex, high performance SoC application. For Smart Vision Systems such a software orientated approach can be leveraged in order to derive a solution that meets the needs in terms of performance, power, and cost but with a significant reduction in development time. For SoC based solutions it is necessary to be able, if performance demands, to partition the algorithms or vision function over the processing system (PS) that contains the Dual Cortex ARM® A9 and the programmable logic (PL) where whole or part of the function is accelerated. Such software orientated flows assist the designer in creating these partitions taking PL(FPGA) target functions and implementing with software orientated methodology. Experienced users of these tools have demonstrated that a reasonably complex algorithm can be coded in days rather than weeks when comparing the design flow to the traditional FPGA (VHDL/Verilog) approach.

Another advantage of such abstracted flows is that many of these tools also accelerate the verification process of the hardware accelerator implementation. This is an area where the traditional FPGA (VHDL/Verilog) designer can spend the majority of their design time, so any speed up with this part of the design process is greatly improving the time to market.
There are a growing number of abstracted programming flows available for targeting programmable logic.
- C/C++ based design flows such as Vivado® HLS, Impulse C, etc.
- OpenCL flows such as SDAccel [2]
- GUI based tools tailored towards graphical input such as Silicon Software’s VisualApplets® and MathWorks® Simulink/System Generator
- Model based design and rapid prototyping with Embedded and HDL Coder from The MathWorks®

This paper will examine two of these flows to highlight the productivity advantages of each flow and the depth of imaging support available in order get the greatest productivity from these available tools.

IV. TECHNIQUES FOR ACCELERATING SOFTWARE CODE
A. High Level Synthesis C to RTL Design Flow, Leveraging OpenCV

Open source Computer Vision (OpenCV) is widely used to develop Computer Vision applications and the available library of optimised functions had grown to a number in excess of 2500. These functions are written in such programming languages as C/C++, Python and Java. In general though this type of system architecture that is used in OpenCV is more optimised for desktop processors and GPU’s, i.e. heavily reliant on a very high amount of memory access, as such is not suitable in its raw form for embedded, low power SoC platforms. The challenge to overcome is how to use such available resources to produce an efficient embedded system that leverages an algorithm partitioned approach into the hardware and software domains of the SoC.

If we look a little deeper into how OpenCV based image processing works we generally see two limitations when considering an embedded SoC architecture:
- It is generally built around sequential processing of frame buffers that reside in external memory. Therefore for in cost optimized, embedded processor architectures that have moderate cache systems the transactions to memory become highly intensive with the frame based memory processing used with OpenCV. e.g. if we consider a standard 720P60 frame it could vary in size between ~1M pixel/frame to ~4M pixel/frame for a high level of bits per pixel such as 32.
- Operational limitations can also be seen by taking the fully sequential approach adopted by OpenCV as a pure CPU based solution the parallelism is limited by the number of CPU cores in the device. Once again if we consider a 720P60 stream we can see that this would result in a throughput in pixels of around 60 M pixels/sec which means that a CPU would need to approximately 16 cycles per pixel, when running at 1GHz. This situation is compounded when we consider that complex imaging algorithms needs many operations per cycle.

It is clear that to get the highest efficiency in terms of performance and power from SoC architecture a different approach is needed. Alternative method of data handling and operation are required such as the prefetching of data into caches and the use of FPGA based parallel processing to eliminate temporary image frame buffers and thus reduce the amount of reliance on memory accesses, i.e. the bottleneck of the embedded system. If we consider such changes for algorithmic functions used in image processing we can generally conclude the following:
- Most pixel processing functions can be chained together with minimal buffering, i.e. each buffer written and read by one particular function
- Some algorithms such as lens distortion correction or stereo rectification requires some buffering but this tends to be much less than a frame
- There are algorithms that must process multiple frames such as optical flow and background subtraction. This must be considered when architecting the system partitioning.

B. NEON Optimisations of the OpenCV/C code

Before we look into how we can target the PL of the SoC for hardware acceleration we should also consider what performance increase can be achieved by optimizing using the available resources in the PS.

As a first step for optimisation using the Cortex A9 ARM® processor we can also look towards the NEON SIMD (Single Instruction, Multiple Data) support that is present as part of the SoC architecture. These computational engines can be used to significantly provide a speedup over generic OpenCV/C code.

Below in Fig. 3 we can see some examples from Uncanny Vision [4] that show what can be achieved by carefully designing the code to explicitly pre-fetch image data and then leverage the NEON instructions for acceleration.
Fig. 3. Benchmarks for NEON accelerated OpenCV functions

Using NEON on its own is unlikely to provide the total solution needed to get the performance levels desired for high resolution, high frame rate video so the question remains, how can the available tools in the market help the designer get the benefit of the parallel processing of the PL but with as abstracted tool flow?

C. Using Vivado HLS and leveraging OpenCV/C code

Vivado HLS (High Level Synthesis) is a design flow from Xilinx that provides the synthesizing of unmodified C/C++ code directly to RTL for use in the PL of the Zynq All Programmable SoC. This tool can be used by Smart Vision System designers to provide acceleration of algorithmic functions that can be built quickly from their original C model and then integrated into the overall ARM® AXI bus infrastructure that is part of the SoC. The output of the HLS tool can be made so that an AXI IP wrapper is automatically generated to allow simplified connectivity to the IP core/accelerator so that it can be easily accessed from the ARM® processing system complex.

In additional the Vivado HLS tool contains a number of video libraries which make it easier for the designer to build a variety of accelerated video processing applications. These libraries are implemented as synthesizable C++ code and roughly correspond to video processing functions and data structures implemented in OpenCV.

Many of the video concepts and abstractions are very similar to concepts and abstractions in OpenCV. In particular, many of the functions in the OpenCV imgproc module have corresponding Vivado HLS library functions. A full list of the library functions including the OpenCV accelerators can be found here:


The above OpenCV functions have been modified and synthesized through Vivado HLS to create the supporting Vision accelerator library. Additional OpenCV functions can be accelerated using the same method and inserted into the library. Changes are required to the native OpenCV functions because OpenCV uses dynamic buffer allocation which cannot be synthesized by the HLS tool. Also HLS functions are designed to be tailored towards a streaming architecture which is more efficient and higher performant in PL.

Though these changes need to be applied the OpenCV interface remains similar in context to the original source with the C++ code contained in the hls namespace. For example:

OpenCV library:
```cpp```
cvScale(src, dat, scale, shift);
```cpp```

HLS video library:
```cpp```
hls::Scale<...>(src, dat, scale, shift);
```cpp```

Fig. 4. Changes to original Open CV code (1)

Additionally some constructor arguments have corresponding or replacement template parameters. In the example below, Fig. 5, the ROWs and COLS would also below specify the maximum size of the image being processed.

OpenCV library:
```cpp```
cv::Mat mat(rows, cols, CV_8UC3);
```cpp```

HLS video library:
```cpp```
hls::Mat<ROWS, COLS, HLS_8UC3> mat(rows, cols);
```cpp```

Fig. 5. Changes to original Open CV code (2)

As mentioned OpenCV code is written in a very memory centric way and relies on lots of memory transactions to a common frame buffer. This is not efficient in an embedded SoC therefore the code needs to be re-written to take advantage of the streaming efficiencies that come from FPGA based architecture. The below figure highlights the changes that have to be made in order to re target the code for streaming architecture.

Fig. 6. original OpenCV code random access and in-place operation on 'dst'

The code is rewritten from the above and the process becomes a streaming one.
Fig. 7. Example algorithm now with streaming architecture
All the necessary steps to convert an OpenCV function into
the video library are covered in application note XAPP1167 [3].

Now that we understand what changes are needed to the
original OpenCV code let’s examine how this could be used in
a practical way. In Fig. 6 looking from left to right we can see that the PC based
OpenCV application can be ported onto the SoC’s ARM’s
processors and ran natively, on Linux. Once executing and
using the available software profiling tools we can examine
the bottlenecks in the algorithms and then identify candidates
to take through the Vivado HLS flow in order to create an
accelerated version of the function or functions. This provides
a powerful tool flow as the designer can continue to reference
and correlate against the original PC based model to ensure
that the correct functionality is valid.

Fig. 8. Design process for taking OpenCV function and
accelerating onto embedded platform
High levels of vision based acceleration can be achieved
with the discussed flow and below are some benchmark data
which highlights the performance levels that can be achieved.
The below data was captured using a 1080P60 target reference
design on the Xilinx ZC702 evaluation board. These functions
are implemented in hardware where the throughput remains
constant regardless of the CPU loading. Acceleration in excess
of 50 times can be seen over the OpenCV running on the
ARM™ processor as a pure software function.

Fig. 9. Example OpenCV functions and the benchmarked
relative acceleration using Vivado HLS

D. Verification with Vivado HLS
As discussed abstracted programming flows for SoC not only
allow rapid creation of accelerated vision functions but also
greatly speedup the verification process. Vivado HLS has in
built support for taking the original C model test bench,
possibly designed on a PC and automatically generates a HDL
test bench for final verification of the function. As the
algorithm/vision function can be refined and tuned at the
model level this automation of the verification stage means
that the designer can be confident that the end result,
accelerated in hardware (PL), is cycle accurate with the
original model. This dramatically speeds up the design
process. Fig. 9b below highlights how the test bench
generation fits into the overall Vivado HLS flow.

V. EMBEDDED VISUALAPPLETS®
As mentioned there are a number of ways to support software
orientated, abstracted programming on SoC’s. An alternative
solution comes from machine vision expert Silicon Software
[5] who have developed a drag and drop graphical
programming environment for FPGA’s and SoC’s that is
meeting the challenge of abstracting the programming level in
a way that is easily handled by video savvy software engineers
and image processing engineers. This programming tool
means that engineers not at all familiar with FPGA technology
and architecture can build powerful vision processing
applications in short time scales reducing the overall time to market.

We see below in Fig. 10 that the programming environment is very much of a flow chart nature and video pipelines can be assembled easily with limited knowledge of the underlying hardware, e.g. there is no need to add FIFO’s, synchronization logic, pipeline adjustment, handling of different data models and attributes, as this is all done automatically by the tool to alleviate this hardware orientated task away from the programmer.

As mentioned previously to get a huge performance increase it is desirable to take advantage of the parallel architectures that can be created in the flexible FPGA & SoC hardware. Silicon Software has designed the tool with this in mind and therefore VisualApplets® looks to exploit parallelism intrinsically in an automatic manner that doesn’t need direction from the programmer therefore maximizing the performance of the generated code with minimal design effort.

Fig. 10. Screenshot of the design entry in the VisualApplets® tool

To aid the designer with rapid development of application the tool is accompanied with an extensive library of over 200 operators, which are coded in an efficient way that makes them close in resource sizing to hand crafted HDL designs. This means that the produced output from the tool is efficient in terms of the FPGA resources that it consumes. Many VisualApplets® operators [6] can be parameterized which allows setting up their behavior at runtime. After synthesizing an image processing chain each parameter gets implemented as a register which can be accessed by a unique address through a single register interface.

A sophisticated resource management system will automatically link the available resources according to the demands of instantiated operators and notify the user when there are not enough resources left to realize the design.

A. Verification

It can be clearly seen that such a tool can add significant productivity gain over conventional HDL design entry methods but we should also consider the verification time that is required as part of any design. Typically a designer can spend up to 70% of their time in the verification phase of the project and VisualApplets® can also help massively there too.

Once again this is handled completely in an intuitive, graphical based environment by adding sources and probes to the flowchart based design. The simulation data input and verification is done with actual source images, not signals, as can be seen below in Fig. 11. The simulation is performed with exact match to the hardware implementation where results can be observed at any point in the flowchart.

Fig. 11. Screenshot of simulation probes and viewing window output in VisualApplets®

B. Targeting VisualApplets® to the Custom Hardware Platform

The embedded VisualApplets® (eVA) framework provides an easy to use mechanism for introducing VisualApplets® to Third Party hardware platforms. The framework allows the user to set up an integration of the VisualApplets® system for creating image processing pipelines into their own FPGA design.

This is done in a three step approach with support tools from Silicon Software:

1) Specify an IP Core configuration. The IP Core interface can be freely composed of standardized interface components and customized for the requirements of the given platform by various parameters.

2) Generating the eVA IP Core and the top-level FPGA design where the core is instantiated as a black box. This enables the generated VisualApplets® application to coexist with the rest of the customer’s system seamlessly.

3) Installing the generated hardware library to VisualApplets®
With this scheme the integration effort only needs to be done once for each platform. After that with VisualApplets® any image processing pipeline can be composed. VisualApplets® will fit the resulting pipeline to the specified interface so the synthesis can be done without having to modify the top-level user design anymore. As a result the user can develop image processing applications within the graphical development environment of VisualApplets® where the bitstream for configuring the FPGA and the runtime software interface are built automatically without the need to handle any tool of the FPGA design flow manually.

As mentioned the VisualApplets® core for the hardware platform needs to be done once by the design team. Fig. 12 below shows the design flow for building a custom VisualApplets® core for a new hardware platform. There is a core generator called eVA CoreGen which creates a black box interface (VHDL or Verilog), a simulation entity for emulating the communication ports of the generated core interface and the hardware interface library for VisualApplets®. The core generator is directed by a hardware definition file. Based on the generated core interface the user can build the top-level FPGA design for their specific hardware platform. Including the resulting netlist and a constraint file for the place and route tool the core generator is called again, now assembling an eVA platform installer for adding the platform to VisualApplets®. Once core generation and installation for a custom hardware platform have been done the hardware is supported by VisualApplets® without any further steps and can program the system as explained.

To ease evaluation there are some predefined eVA platform installers such as for the ZC702 board from Xilinx.

![Flow for Generating the VisualApplets Core for the custom hardware](image)

**Fig. 12. Flow for Generating the VisualApplets Core for the custom hardware**

**C. Software Integration**

eVA will also assist on the software side by providing a code generator for the runtime software.

In general VisualApplets® operators contain dynamic parameters which can be modified during runtime. On the FPGA side VisualApplets® implements a slave interface for register access through which design parameters are communicated. In order to communicate parameters from/to the VisualApplets® core write and read accesses to design parameters must be translated to accesses on the register slave interface of the core. VisualApplets® automatically generates such interface code so the user can directly access parameters by their name and does not need to take care of the low level register accesses. For embedded platforms there are three options for generating interface code:

1) For platforms controlled by a Windows or Linux host VisualApplets® can generate an applet specific runtime interface component which can be opened by runtime interface software installed on the target system. This is supplemented with an automatically generated SDK code example for setting up the applet parameters.

2) For any target platform containing a processor VisualApplets® can generate ANSI-C source code for a parameter interface specific to the applet. Here both Linux and standalone applications can operate in the same way, i.e. there is no need for external runtime interface software. This approach has been used for the Zynq All Programmable SoC demo application described below controlling the VisualApplets® core from the application running on the ARM™ core.

3) For systems which provide a GenICam™ producer for accessing the VisualApplets® core VisualApplets® can generate a GenICam™ camera description XML file for the applet so parameters can be accessed via GenICam™ API. The generated code requires Version 2.0 of the GenICam™ standard.

**VI. USE CASE ANALYSIS – IMPLEMENTATION OF AN EMBEDDED SMART VISION SOLUTION**

**A. Introduction**

High speed machine vision inspection is a very common application in the manufacturing space and typically the kind of systems deployed are using a high speed camera (line scan is common) at a medium to high resolution level with a high number of lines/second. The images are then collected and then typically transferred over high bandwidth link/s to a PC for vision based processing and analytics.

A key trend in the market is to increase the speed and accuracy of production lines therefore what is seen is an ever increasing amount of image data that has to be captured and then sent to the PC for processing. Such system architectures are pushing the limits of connectivity in terms of cost and performance. A smarter solution is to capture and process the images at the camera, i.e. with a Smart Vision solution. This is
not a new approach but such high performance systems have needed the power of an x86 CPU in the past in order to meet the performance requirements for the applications. This has limitations on cost, size & thermal design and therefore there is a growing demand for a more embedded approach. SoC's can offer a level of pixel performance using parallelism that is not possible in standard off the shelf embedded processors. In the following use case example we investigate the use of a SoC, which has then been implemented in a real life demonstration, to show the performance that can be achieved in one embedded, integrated platform.

B. Summary of demo application

To mimic the typical setup for an object inspection application a rotating drum was used to represent a conveyor belt. The drum has a printed covering that has many colored circles mimicking candy. Around the drum there are also defective parts, i.e. 4 wrong objects with different types of errors (wrong color, wrong size, broken, merged). In Fig.13 below we can see a sample screenshot of the captured video and the analysis GUI that is presented to the viewer. The task is to scan all objects and identify all defects using a color line scan camera with a line width of 2048 pixels and line rate up to 41 KHz and perform real-time image analysis for detecting and classifying objects.

In addition real-time control of a linear actuator and stroboscope is done also by the SoC to highlight to the user the defective part, whilst the drum was rotating at high speed, i.e. the linear actuator positions the strobe over detected faulty object, then a flash is triggered when the object passes. The Image processing and actuator control on the SoC PL guarantees real time feedback with extremely low latency and triggering precision equal to the resolution of the recorded image.

C. Image Processing Steps in the demo

If we examine one of the segmentation functions that are done by the realized inspection system we can see how this has been implemented with the use of the powerful VisualApplets® tool.

The below image in Fig.14 show the initial captured image from the 2K Tri-Linear line scan camera and the subsequent processed image that is the output of the image processing operators that have been implemented in the VisualApplets® tool to identify the blue objects, i.e. the binary result plane of the blue objects after a filter has been applied. Five independent binary color planes are handled in parallel. The color binarization is based on the robust HSL color-space (http://en.wikipedia.org/wiki/HSL_and_HSV).

In Fig. 15a the full core image processing pipeline is shown with the processing steps as follows:

1) Color: White balance adjustment
2) Transform: Color space conversion; transfer pixel data to HSV color space.
3) Detect: Binarization into five color planes according to four defined color ranges plus a range covering any other color.
4) Filter: Morphological filter for optimizing binary image using 3x3 erosion, 5x5 dilation and 3x3 erosion again
5) Segmentation: Blob analysis; generate lists of detected objects with bounding box, center of gravity and blob area.
6) Classify: Sort out irrelevant blobs
7) Data: Reduce object data to feature set relevant for software analysis
8) Assembly: Pack image data and analyzed features for transfer to memory

![Fig.14. Initial captured image and binary result for blue objects, filter has been applied](image-url)
D. Distribution of tasks between Zynq PS and PL

For the application it was necessary to partition the functions across the SoC PS and PL to get the highest optimized solution in terms of performance, cost & power.

**SoC PL:**
- Image acquisition from Camera Link (85 MHz Pixel rate, RGB format, max. 255 MBytes/s)
- Image analysis pipeline
- Ring buffer for image data in external memory
- HDMI output
- Control of linear actuator

**SoC PS:**
- Frame buffer management
- Processing results, Analysis of image objects feature set
- Preparing control commands for linear actuator
- Visualization
- Statistics

The image analysis pipeline in the SoC PL is directly driven by the pixel data coming from the camera interface so analysis is done immediately without the need to start it from software. Besides being the most efficient communication scheme this allows for extremely low latency of feature calculation, i.e. the bounding box for an object is available after less than 1ms after the object passed the camera. The communication of image analysis results from PL to PS is realized by appending feature set data into reserved memory after each line of the grabbed image, i.e. the application on the SoC PS receives images with increased width and interprets the right margin of the image as feature data. Image data and analysis results are written via direct memory access from the PL into memory which the application on the PS can access, avoiding any image copy and analysis results polling operation.

The key aspect in order to reach the performance levels demonstrated is the acceleration of the image analysis pipeline. Without its implementation in the PL of the Zynq it would not have been possible to reach the very high performance and determinism demonstrated.

E. Performance

SoC technology and VisualApplets® provides a powerful platform for accelerated vision processing applications. The practical example implemented as described highlight that such a system can provide an acceleration of greater than 50 times when compared to a software only solution running on the PS part of the SoC.

Estimated computation power of PL design:
1) **Color:** \[ 2 \times 3 \text{ Ops} \times 75 \text{ MHz} = 450 \text{ MOps/s} \]
2) **Transform:** \[ 2 \times 19 \text{ Ops} \times 75 \text{ MHz} = 2850 \text{ MOps/s} \]
3) **Detect:** \[ 2 \times (5 \times 11 + 12) \times 75 \text{ MHz} = 5100 \text{ MOps/s} \]
4) **Filter:** \[ 16 \times (9 + 25 + 9) \times 75 \text{ MHz} = 51600 \text{ MOps/s} \]
5) **Segmentation:** \[ 16 \times (8+4) \times 75 \text{ MHz} = \sim 14400 \text{ MOps/s} \]
6) **Classify:** Negligible
7) **Data:** Negligible
8) **Assembly:** Negligible

As a rough estimation of the processing power of the realized PL design we get a number beyond 70 GOps/s.

The pipeline is capable of sustained processing of 2 pixels per clock cycle at 75 MHz which for RGB input data results in an image data throughput of 450 Mbyte/s. This is even more than the source data rate of 255 Mbyte/s which the camera can deliver.

In order to benchmark the PS only implementation we can consider what could be achieved using a software only solution that is available running on Zynq, the HALCON machine vision library from MVTec. Taking the functions for HSV conversion, calculating color binary planes by thresholding over saturation and hue and finally performing a blob analysis we observed a throughput in the range of a few Mbytes/sec while we have not even accounted for morphologically filtering the binary planes.

Using such image processing tools and implementing a high performance, power efficient solution the performance is reaching a level which is able to process 72 frames per second at 1080P.
VII. CONCLUSION

- The tight coupling of Processors and FPGA in SoC’s are well suited for image processing and Smart Vision Systems.
- Programming the FPGA part of the SoC with High Level Synthesis (OpenCV libraries) and Embedded VisualApplets® raises the abstraction for embedded programmers and shortens time to market.
- VisualApplets® is an innovative software orientated tool that accelerates the time to market of vision based SoC designs and can provide a flexible platform for designers and also an open platform for the end users of the vision equipment.
- New tools will raise the level of abstraction to C/C++/OpenCL programming for the combination of ARM™ processors and FPGA.
- The power consumption of these SoC systems can be in the range of a few Watts.

REFERENCES

[2] Xilinx SDAccel Backgrounder
http://www.uncannyvision.com/uncannyvc/
[5] Silicon Software VisualApplets® Tool
http://www.silicon-software.info/en/products/content/41-visualapplets.html
https://www.youtube.com/watch?v=k0lh2sOnzJ8
https://www.youtube.com/watch?v=vyBfKvis2tY