Implementation of a Hardware-centric Vision System Architecture

Azman Muhamad Yusof[1,2], Ali Yeon Md Shakaff[1,3] and Saufiah Abdul Rahim[3]
[1]*Center of Excellence for Advance Sensor Technology (CEASTech), UniMAP.*
[2]*Department of Electronics Engineering Technology, Faculty of Engineering Technology, UniMAP.*
[3]*School of Mechatronic Engineering, UniMAP.*
azman@unimap.edu.my

***Abstract*—Currently, most implementations of vision systems still heavily rely on software - computer algorithms run on general purpose microprocessors, like on personal computers. This is understandable since personal computers (PC) are readily available, and software implementations provide flexibility, especially when trying out various algorithms. The need to have real-time vision-based systems influenced developers and researchers towards hardware-based - or at least hardware-assisted - vision systems that are capable of processing huge amount of data from an imaging device in real-time (i.e. embedded vision system). Platforms like DSPs, GPUs and FPGAs are among the commonly used development platforms for a hardware-centric vision system, while ASIC implementations - tagged with a huge development cost - usually have the best performance. This paper compares various possible platforms that are readily available and can be used to develop hardware-centric vision systems. This includes DSPs, GPUs and FPGAs, with some insights on ASIC implementation. Consequently, two implementations of the proposed hardware-centric vision system architecture are presented. Both implementations managed to process incoming image stream from camera module at 30 frames per second.**

***Index Terms*— Embedded Vision System; Hardware-Assisted Vision System; Image Processing Hardware; Machine Vision.**

I. INTRODUCTION

Vision is defined as the ability to identify objects and their relative positions [1]. It could be the most valuable sensory mechanism that a system can have. Its data can potentially provide a tremendous amount of information from a single sample. Naturally, it also requires substantial processing power in order to extract useful information out of it. The fact that visual data is in 2-dimensional form makes processing in vision systems a real challenge. Stereo vision doubles that complexity, with the reward of having third dimension information.

With the advances in mobile robot systems, among other reasons, the need for a robust computing platform other than personal computers (PC) became more apparent. Although some have built a chassis big enough to hold PCs for these applications, that is apparently not an option in the long run. Humanoid robots [2] and autonomous cars [3] are great examples of why a robust, yet portable, hardware-based vision system (embedded vision system) is required. Therefore, efforts on implementing a practical vision system should also consider hardware-related issues like system platform, architecture and interfacing.

Implementations of vision systems have always been based on human vision, thus trying to imitate the things that actually take place in the actual biological system. The human vision has the ability to see the surrounding environment without any conscious thinking (i.e. brain efforts) - they merely 'see'. This is sometimes referred to as early vision, while others only classify it as low-level image processing (or image pre-processing).

On the other hand, it is the conscious signs of vision (i.e. the ability to identify objects and their relative positions) that executes complex image processing tasks to provide valuable information for decision making (i.e. action to be taken or reaction). Apparently, in an actual biological system, such 'boundary' (i.e. between low-level image processing and high-level decision making) does not exist. These classifications are made for the more structured development of these 'artificial' vision systems.

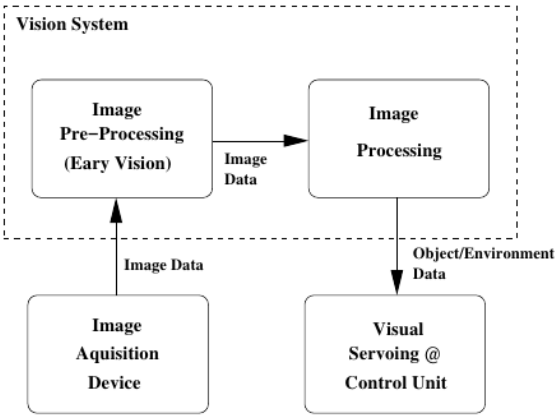

Figure 1: Components of a basic Vision System and its data flow

Figure 1 shows typical components of a basic vision system. The components have been categorised according to its task in the system. It is worth noting here that the control unit can be implemented within the vision system for a more compact embedded vision system. A vision system usually receives image data in a streaming fashion from the imaging device. This is the primary reason why most current vision systems have latency and why it can be quite difficult to achieve 'real-time' performance. Nevertheless, data streaming is also currently the only practical way of transferring such amount of data. Notice that, for a system to decide on a suitable action or reaction, only meaningful data of the surrounding object(s) or environment is needed. This is no longer in image form and, in fact, can be in any abstract form.

In the next section, various existing implementations of vision systems will be discussed, especially on the hardware platform used. The following section will cover the proposed

architecture of an excellent hardware-centric vision system. Consequently, some analysis on practical implementations of the proposed architecture will be presented, before more advanced features are discussed in the following section.

II. Existing Implementations

Many, if not most, vision systems have been implemented on personal computers (PC). The reason for this is most probably due to availability factor rather than suitability factor. In addition to that, there were not as many options for processing elements back then as it is today. As soon as PCs became powerful enough to handle basic image processing algorithm, it became the natural choice of research platform. However, with the availability of other processing elements like DSPs, GPUs and FPGAs, more researchers have opted to explore these alternatives.

A. Digital Signal Processors

Digital Signal Processors (DSP) are merely customised microprocessors that are equipped with basic multiply-accumulate (MACC) blocks and specialised single-instruction-multiple-data (SIMD) arithmetic instructions to accelerate computations of digital signals. The nice thing about a DSP is that it is programmable and developing software for a DSP is not much different from developing one for a PC.

In [4], Texas Instruments' TMS320DM642 Evaluation Module has been used in a fabric defect detection system (FDDS). The system utilises the DSP kit's direct memory access (DMA) feature to facilitate image transfer from an onboard camera input to the onboard SDRAM memory module. The detection algorithm was implemented in C language using Code Composer Studio IDE, with the help of some predefined video processing library that comes with the module. The compiled code can then be downloaded to the DSP module for execution. The FDDS system in [4] has been reported to be capable of processing only two frames per second, with each frame covering 4 cm2 area of fabric. The image resolution is not mentioned.

Another implementation of vision system on DSP platform has been used in image haze removal [5]. The system has been developed using TMS320C6678 development board, which is a multi-core DSP what is capable of parallel processing. The board has 8 C66x DSP cores with 1.25GHz for each, along with 4M shared L2 SRAM and 2GB DDR3 memory. The algorithm has also been developed in C language using Code Composer Studio. The proposed image haze removal system is capable of processing 600x400 image frame in less than 50ms, but it should be noted that some downsampling process was used, which makes the effective resolution a lot less than that.

B. Graphics Processing Units

Graphics Processing Units (GPU) are mainly used to render images (i.e. synthesis) for display units. However, since most image analysis operations generally use the same arithmetic operations as in the image synthesis operations, GPUs become a viable candidate to process vision data. In fact, the processing power of GPUs is more suitable than DSPs due to the nature of their purpose.

In [6], an embedded development board based on NVIDIA's Tegra K1 is used in the detection of defective orange. The development board is equipped with HDMI video output, gigabit ethernet port and USB 3.0 port. The Tegra K1 itself contains a 32-bit quad-core ARM Cortex-A15 with 192-core Kepler GPU, capable of running up to 2.3GHz clock frequency. The system uses an industry colour gigabit ethernet camera BFLY-PGE-13S2C with 1288x964 image resolution that is connected through an ethernet switch to the development board. It is running a Linux-based Operating System (OS) that allows many software libraries that are available for PCs. The algorithm is developed mainly in C/C++ language based on OpenCV library customised for Tegra. The reported processing time for a single orange is less than 30ms, and the detection success rate is about 95%.

A face detection system has been developed using GPU in [7]. The project also highlights the use of CUDA programming language, which has been introduced by NVIDIA to allow GPU usage for other general processing work, as well as working together in parallel. The GPU used in this project is NVIDIA GeForce 310M, which is configured as a co-processing element along with Intel i5 Core as host CPU. The image frame to be processed is supplied by the host CPU. This project compared GPU performance against standard CPU implementations and concluded that a speedup of at least 16 times could be obtained using GPU, with some cases going up to over 20 times.

C. Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGA) are meant as a way to prototype digital logic circuits on hardware fabric, which makes it inherently faster than its equivalent software counterpart executed on a similar hardware fabric. Naturally, FPGAs can easily achieve parallel processing capabilities by having multiple processing logic blocks implemented on it. However, because of this low-level feature, it is relatively hard to implement sophisticated algorithms that require variable sequencing and iterations.

The work presented in [8] sums up the advantages of using FPGAs in vision system compared to other platforms. The object detection system being used is a pedestrian detection system that has been implemented on Convey HC-2ex machine, which boasts a hybrid-core architecture that consists of two Intel Xeon E5-2643 four-core processors and four Xilinx Virtex-6 LX760 FPGAs. All CPUs and FPGAs have their local memory, but those are globally addressable as 256-GB virtual memory. The FPGA development work is done in Verilog HDL using Xilinx ISE suite. The FPGA-based system is capable of processing VGA resolution images (640x480) at about eight frames per second when using floating-point implementation, and at about 68 frames per second when using fixed-point implementation. This fact exhibits the versatility of an FPGA implementation that allows developers to customise data representation and internal storage format.

Meanwhile, in [9], a vision-based robot tracking monitoring system has been developed using Xilinx Virtex-4 XC4VFX100-11 FPGA. It uses 4 Gigabit Ethernet camera to cover a robot arena of 6m x 6m. They are cable connected directly to the RAPTOR Development board, each camera providing 1024x1024 image pixels per frame. Each robot is marked with a circle (robot marker), a pentagon (direction) and a barcode (unique robot identification). For a 2048 x 2048 pixels camera image, the system managed to run at a maximum frame rate of 152 frames per second. Using four cameras, the maximum frame rate for the complete system is

limited to 119.2 frames per second. When compared to a software implementation based on the OpenCV library on a state of the art PC equipped with a 3.2 GHz Intel i7 quad-core CPU, a speedup of more than 30 can be achieved.

D. Other Alternatives

ASIC implementations are the most ideal regarding performance since the processing elements can potentially be placed on the same integrated circuit (IC) fabric as the image sensors. The processing element can be designed to process per-pixel information even before being streamed to the next processing level. However, doing this requires a considerable cost regarding time and money - not to mention the need for IC design experience.

In [10], a System-on-Chip (SoC) that implements a microprocessor with a customised instruction set or an Application Specific Instruction set Processor (ASIP) has been introduced. Having CMOS image sensors on-chip reduces the cost of image transfer that is usually inherent in many conventional vision systems. However, using serial peripheral interface (SPI) as the only data interface creates a limit to the image stream bandwidth. This is compensated by expanding the SPI port to allow 2, 4 or 8 output lines which can be used by external processing elements like FPGA, while maintaining backward compatibility with an older microcontroller that needs standard SPI. Even though no practical applications have been presented, the advantages of this 'Vision Chip' are encouraging.

One other processing element that is worth mentioning and can potentially be used for a vision system platform is Cell processors. Cell processors are microprocessors based on general purpose Power Architecture Core (i.e. used in PowerPCs) that has been combined with co-processing elements to enhance multimedia and vector processing capabilities. They can be seen as a combination of GPU and general microprocessor on the same IC. Cell processors have the processing power to be used in vision systems, but unfortunately, they are not so readily available to all.

III. HARDWARE-CENTRIC ARCHITECTURE

As with many other optimum solutions, the preferable approach to implement a vision system is converging towards combining multiple processing units. Hardware-centric component could provide performance, while the software-centric component provides feature-rich solutions for implementing the complex algorithm. Nevertheless, implementations of vision systems are usually very objective-dependent and differ from one another based on its purpose [11]. The purpose of this proposed implementation is to focus more on the hardware implementation (i.e. hardware-centric) part of a vision system.

The most critical decision that has to be made when implementing a hardware-centric vision system is selecting a platform. This is because how a system is and can be, developed will be based on that particular decision. Referring to Figure 1, the implementation that needs to be considered is the components inside the dotted-lined box. The following subsections cover specific parts of the proposed implementation, including platform selection.

A. Hardware Platform

The proposed implementation will make use of FPGA's inherent ability to have multiple processing elements working in parallel. This allows the system to execute at a lower frequency (lower power consumption) while maintaining data throughput.

The fact that development work on FPGA needs some digital logic design knowledge does not pose any problems since there are tools (e.g. Matlab® toolbox) that can synthesise digital logic circuit based on a standard sequential algorithm. However, to make tweaks to design, it is best if everyone can be described or structured from the ground up.

The versatility of FPGA allows the system to implement a camera interface module which allows an imaging device to be connected directly to the system. Since a common imaging device with digital interface usually produces image data pixel-by-pixel in streaming fashion, a reasonably fast processing element can execute some processing while the image is being transferred.

Figure 2 shows how processing element like FPGA can take advantage of pixel data streaming. As shown in Figure 2(a), most implementation usually uses a standard PC (or another controller that takes care of image capture) to grab a frame and store it on system memory, before passing it to the processing element. The processing element can then pass the processed data back to the main processor for high-level processing or directly to display.

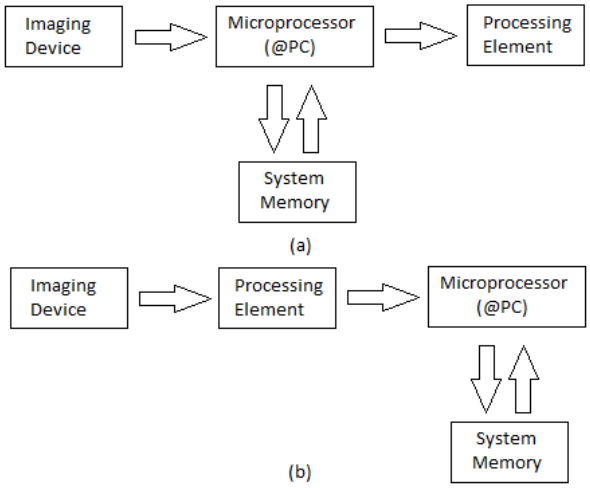

Figure 2: (a) Common vision system implementation, (b) Proposed Implementation

On the other hand, the proposed implementation assigns the processing element (in this case, the FPGA) to directly capture the input frame from the imaging device before passing it to the main processor for high-level processing. This allows for the low-level image processing (early vision) to occur in-stream, while the image is being transferred.

However, interfacing a processing element directly to an imaging device can introduce other concerns like clock domain crossing.

B. Clock Domain Crossing

When creating an interface module for an imaging device (camera module), the easiest way to get image data is by sampling the control output signals and register pixel data when it is available and valid. This is shown in Figure 3.

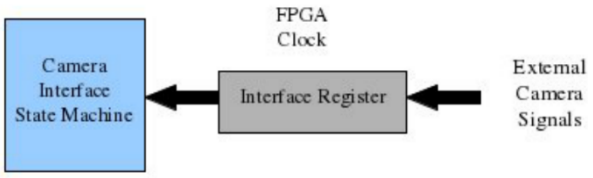

Figure 3: The state machine samples control and data output from the camera module

The sampling is done using FPGA clock and the camera module usually (and it is, in this case) has its onboard clock. This is something that is not desirable because a digital system works better in a synchronous single-clock domain. The sampling of camera module signals may produce unexpected errors, even if the FPGA clock is running more than twice the frequency of the clock module (Nyquist rate). There are two options to overcome this: the first is to use a camera module that can be controlled by an external clock (i.e. using FPGA clock signal), and the second is to separate the two clock signals using a dual clock First-In-First-Out (FIFO) buffer as shown in Figure 4 below.

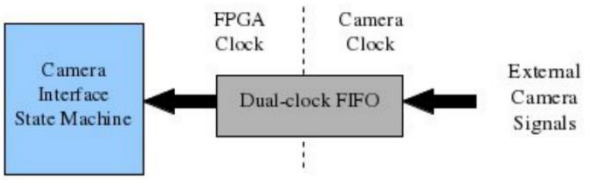

Figure 4: The FPGA clock domain and camera module clock domain separated using dual-clock FIFO buffer

The FPGA clock domain is now clearly separated from the camera module signals. The internal state machine only needs to wait for camera data to be available in the FIFO buffer and read from it. This proves to be a better design compared to the previous register-based interface.

C. *In-stream Processing*

This concept is only possible if the processing element has direct access to the imaging device. Most, if not all, imaging device transfers an image pixel-by-pixel at a specified rate. So, instead of storing everything in memory, a powerful processing element can execute some pixel manipulation procedure while image data is being transferred into the system.

There are four common levels of processing in a vision system:

1. Pixel manipulation - each pixel can be processed independently. For example, thresholding or grayscale conversion. This type of processing can be inserted at any stage and only adds latency to the overall processing. The output is still a pixel value (image data).
2. Local neighbourhood - pixels are grouped (usually an *m x m* square) for better interpretation. For example, edge detection or blurring. This can be implemented using FIFO line buffers and also only adds latency. The output is pixel information related to its neighbouring pixel (filtered image data).
3. Global neighbourhood - the whole frame is needed to produce processed information. For example, histogram and pixel counting. Output can be either per-pixel information (filtered image data) or per-image information (abstract data).
4. Inter-frame processing - a sequence of images is needed for this. This is specific to vision systems, in which past frame (or frame information) sometimes need to be retained for at least another one frame period. For example, optical flow computation. The output is usually in abstract form but can be per-pixel information.

For in-stream processing, only the first two types are suitable candidates. The third type can be included but only if the whole frame need not be buffered for processing, like pixel counting. As for the fourth type, it is best if this is implemented in the high-level processing element, where memory management unit may be required. This subsection will subsequently focus on implementing the second type because this type is the most commonly used filtering method used for early vision.

Image data streams are usually structured in rows, column by column. This makes processing windowed region of the image a little bit tricky. For an *m x m* filter size, we need to buffer *m* rows of the pixel. This is shown in Figure 5. Notice that the FIFO buffers introduce data latency equal to *m* image row period.

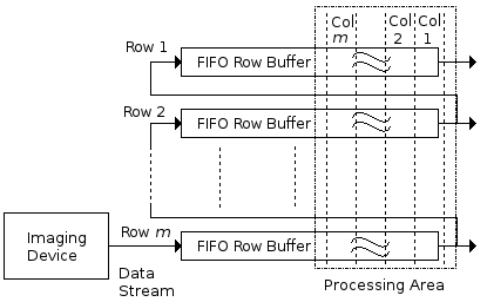

Figure 5: The Basic Idea of In-Stream Processing on FIFO buffer

Ideally, *m x m* processing elements need to be placed in the area marked by a dot-dash line in Figure 5. However, this is somewhat not doable in FPGA design because FIFO element is usually part of the core FPGA component library that has been optimised for FPGA implementation. Instead, the design can be restructured as shown in Figure 6.

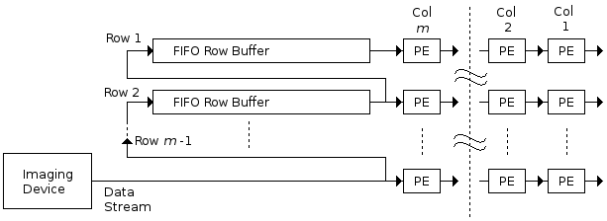

Figure 6: Actual Implementation of In-stream Processing

Since the processing elements need to be placed separately from the FIFO buffers, each of them needs to have a data latch to hold the pixel value (as shown in Figure 6). There is an advantage if doing this. Since the pixel data in row *m* is now latched by the processing elements (PE), there is no longer a need to buffer that last line. So, the increase in size with the addition of extra *m x m* data latches is already compensated by the removal of a full FIFO row buffer (less FIFO area). This also causes the data latency to be reduced to *m*-1 image row period plus *m* clock period.

D. Separable Filter

Image filters usually perform a convolution operation against the pixels surrounding a pixel being processed. It is a sum of the products of neighbouring pixels (including current pixel) with filter coefficients. Generally, for a 3 x 3 image filter, processing a single pixel would require nine multiplications and eight additions. Since convolution is associative, 'breaking down' the 2-dimensional image filter into two vectors should reduce that processing load. Consider a 3 x 3 kernel of Sobel filter for x-direction.

$$s = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (1)$$

The kernel in Equation (1) can be rewritten as a product of a row vector *h* and a column vector *v*, as shown in Equation (2).

$$v = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad h = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$s = v * h \quad (2)$$

The required computation is now three multiplications and two additions for the first vector, and another three multiplications and three additions for the second vector. That is about 30% reduction in processing complexity. Although in our parallel design, this does not matter much, it is still an advantage since we do not need as much multiplier in the processing elements.

IV. Implementation Analysis

This section discusses two implementations of the hardware-centric architecture presented in the previous section. This is to show that the architecture does not rely on any specific FPGA device. The two implementations use different FPGA boards that were designed for different purposes. The first board is a general purpose FPGA development board that imitates a motherboard of PCs, while the second is a customised FPGA board meant as a co-processor to a mobile robot controller board.

A. Xilinx ML310 Development Board

The Xilinx ML310 development board is very similar to a standard desktop computer motherboard. For example, it has the typical USB, LPT and COM communication ports, PCI slots, DDR SDRAM memory, and even ports for standard mouse and keyboard. The features that are of interest in here are the 256MB DDR SDRAM, CF card slot, the serial COM port and, of course, the FPGA Virtex-II Pro chip XC2VP30.

Having a large memory for storage is always useful when executing image processing functions. The 256MB DDR SDRAM is more than enough for any embedded vision system to operate, but the excess memory could be used to hold multiple frames for advanced processing and debugging purposes should the need arises. On the other hand, a large memory with single access bus is not desirable for a system that can have parallel processing blocks executing at the same time.

The XC2VP30 is a member of the Virtex-II Pro Xilinx FPGA family. Its most outstanding feature is the availability of two internal 32-bit RISC PowerPC core. It also has around 30,800 logic cells and almost 13,700 configurable logic blocks (CLB). In addition to that, there are 136 18x18-bit multipliers and 136 18kb block RAM on it.

The test setup is shown in Figure 7. Interface board for a CMOS camera and an LCD has been built and connected to the Xilinx ML310 Development Board. The LCD is not necessary for the vision system, but at the development or testing stage, it is essential to be able to verify what our system actually 'sees'.

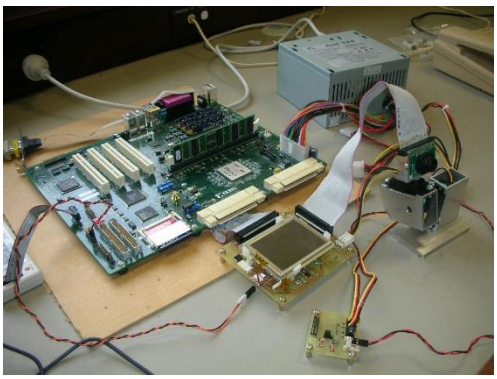

Figure 7: The test setup for ML310-based Vision System

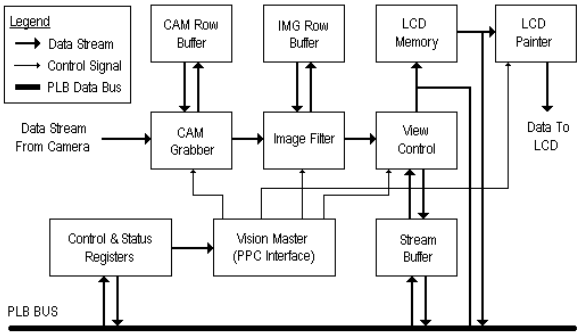

Figure 8: Block Diagram to show in-stream processing on ML310-based Vision System

The overall implementation shown in Figure 8 is based on using the PowerPC core that is available on the FPGA. A custom controller for both the LCD and the camera has been developed on the FPGA along with grayscale conversion and an edge detection filter module. The camera has a CIF resolution of 352x288 while the LCD can display a 320x240 12-bit colour image. The camera data input stream is actually in Bayer pattern, and therefore, the camera interface module (camera controller) needs to have a demosaicing (CFA interpolation) procedure as well.

All the modules used in the system were developed using VHDL from scratch, except for the Processor Local Bus (PLB) interface which is better off using the provided core library. The PLB is only used to write the processed image to RAM, and for the main controller (PPC405) to send control signals (i.e. configurations) to the vision module. The PPC405 itself is executing a control program written in C. The vision module operations can be controlled through the serial port RS232 interface that is available on the ML310 development board.

The output of the Sobel edge detection feature as seen on the LCD module is shown in Figure 9. It should be noted that

the display is a live feed that is streaming QVGA resolution (320x240) image at 30 frames per second.

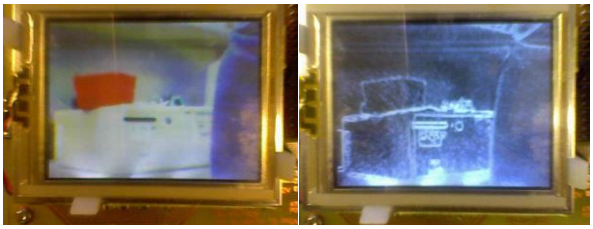

Figure 9: Original image (left) and Filtered Image (right)

Table 1 shows the FPGA resource utilisation for this particular implementation on ML310. The number of logic slices available on FPGA usually indicates the size of logic circuits that can be implemented. It usually consists of registers (memory) and logic blocks (Look-up Tables or LUT). The data shows that most of the FPGA resources have been used for basic image capture and display.

Table 1
FPGA Resource Utilization in ML310 Implementation

	Used	Available	Percentage
Logic Slice	7,452	13,696	54%
Block RAM	79	136	58%
4-input LUT	11,684	27,392	42%

Although there are still some spaces available for more filters to be implemented, it would be more beneficial to offload the display buffer to an external component (maybe one complete with its controller). Since the display is most probably not needed, this would make the design closer to the final implementation. This is useful in determining the optimal memory requirement for the final system, as can be seen in the second implementation.

Referring to the utilisation data for block RAM as presented in Table 1, it should be noted that only 1 out of 58 percent block RAM usage is used for the simple Sobel filter (i.e. as row buffers). Other than the LCD memory, the PowerPC is also using 64kB of the remaining block RAM for its instruction and data memory. The external DDR SDRAM is yet to be used due to the requirements of the design architecture. The processing blocks in the stream are expected to have exclusive access to the memory buffer assigned to it. Thus, the need for dual-port memories is essential in its operation. Putting the processing blocks on a single bus that can access the external DDR SDRAM would introduce delay. This scenario is, in fact, has been the bottleneck of many systems that depend on data from memory. There are only two expected scenarios where the vast external memory can be used in the current architecture; (1) High-level image processing (or any other kind of image processing) that is executed by the PowerPC, and (2) Image stream is written to the external RAM and read by another type of filter block that is also a master on the PLB bus (on which the DDR SDRAM is connected to).

B. EyeBot M6 Controller Board

The EyeBot M6 controller board is a general purpose embedded system board that is equipped with stereo vision capabilities. The main controller device is an off-the-shelf Gumstix Connex 400xm-bt, a single board computer (SBC) that has a 400MHz Intel XScale PXA255-CPU, 64MB RAM and 16MB flash memory. The controller is configured to run Linux OS that is built using *buildroot*, a tool that can be used to generate Embedded Linux systems. In addition to that, it also has a Spartan-3E family Xilinx FPGA, the XC3S500E PG208, which is a low-cost FPGA with a relatively high logic density. It has about 10500 logic cells, almost 1200 CLBs, 20 18x18-bit multipliers and 360kb block RAM.

The FPGA on EyeBot M6 is clocked using a 50MHz crystal and has exclusive access to a 2Mb static RAM (SRAM) as well as the dual camera interface (stereo vision). There is no configuration memory for the FPGA – so, the FPGA is designed to be programmed by the 'host' processor running Linux. A Linux kernel driver is available to provide the interface required to do that.

The vision module implemented on Eyebot M6 is the same one (developed using VHDL) used on ML310. However, some changes were made mainly on the memory interface, display and access method. For one, the EyeBot M6 provides an exclusive SRAM module, which negates the need for internal Block RAMs (which the Spartan-3E does not have). Next, the implementation on EyeBot M6 does not need an LCD controller because the controller on Gumstix already handles that. As mentioned earlier, a vision system does not need it. Finally, the access method is different since there are no PLB controllers in this implementation. Instead, a simple memory addressing method has been implemented so that the PXA255 controller can access the processed image on SRAM.

Table 2 shows the FPGA resource utilisation for this implementation on EyeBot M6. The data shows that basic grayscale filter and Sobel edge detection modules take less than 10% of the available resources on the Spartan-3E FPGA.

Table 2
FPGA Resource Utilization in EyeBot M6 Implementation

	Used	Available	Percentage
Logic Slice	501	4,656	10%
Block RAM	0	20	0%
4-input LUT	648	9,312	6%

It is clear that a lot more can be put into this implementation and having the main controller off-chip (unlike ML310 implementation) helps a lot in achieving this.

V. DISCUSSION

It is evident that the only implementation that can be seen to be more efficient than in-stream processing is an ASIC implementation that has both processing elements and imaging device on the same silicon. This section discusses another advantage of FPGA-based implementation compared to other platforms and the comparison between the two implementations presented in the previous section.

A. Reconfigurable Computing

The reconfigurable nature of FPGAs makes them a compelling platform for any digital system. When used with another processor, the FPGA fabric can be reconfigured at runtime without changing any hardware interface.

In the past, even a small change in the system design would require the whole system to be re-synthesised and the FPGA to be re-programmed. Dynamic reconfiguration is a feature in which only the modified part of the FPGA needs to go through the process. If the modified section is not part of the

main processing block or if the rest of the system is not dependent on it, the reconfiguration process could be done at runtime. This introduces the idea of having image processing filters as hardware modules that can be dynamically inserted or removed as required.

On Xilinx FPGA, the feature for this purpose, which is also known as partial reconfiguration, is available on Virtex-II chips onwards. However, using this feature is not an automated process. The system designer needs to manually partition the respective part of the system that will go through the reconfiguration process and ensure that the rest of the system is manageable throughout the procedure. In short, more work needs to be done to be able to incorporate this feature into a design.

B. Comparison of Implementations

From a development point of view, there are a few things to note. Table 3 shows that the ML310 consumes a lot more space compared to Eyebot M6. This is to be expected because everything is implemented on FPGA, which have a PPC405 PowerPC core to run the controller software system. Because of that, this implementation requires Xilinx EDK software to configure the PPC405 core configurations and the peripheral controllers around it.

Table 3
Comparison of Implementations

Platform	Used	Development Tools	Synthesis Time Estimated
Xilinx ML310	More than 50% usage.	Xilinx EDK & ISE	35 minutes
EyeBot M6	Less than 10% usage	Xilinx ISE & GCC-ARM compiler	10 minutes

The implementation on EyeBot M6 is based on the interface to Gumstix, with the image grabber and image filters implemented on FPGA. The fact that the FPGA is programmable at any time by Gumstix controller board makes it an excellent example of how reconfigurable computing can be beneficial.

Synthesis time is the time needed for the design software to build an FPGA bitstream image file that will be downloaded to an FPGA device. For testing designs on real boards, the bitstream image file needs to be re-synthesised if there are any changes made to the design. Note that the synthesis time when using Xilinx EDK could get up to 35 minutes per design, which can be frustrating at the early design stage if the implemented design does not work as expected as it often does.

VI. CONCLUSION

FPGA could be the ideal platform to implement a hardware-centric vision system. Both implementations show that they are capable of processing incoming image stream at 30 frames per second. The proposed architecture can easily be upgraded with the reconfigurable nature of FPGA. Added by the fact that current high-performance FPGAs are also fitted with DSP computation modules, the possibilities of having an artificial vision system that is similar to biological vision system are increasing quite rapidly.

ACKNOWLEDGEMENT

The ML310 Development Board was donated by Xilinx U.S.A. for evaluation. Many thanks go to CIIPS Laboratory (University of Western Australia) for allowing the use of EyeBot M6 controller board.

REFERENCES

[1] D. Marr, *Vision : A Computational Investigation into the Human Representation and Processing of Visual Information*. San Francisco: W.H. Freeman, 1982.

[2] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura, "The intelligent ASIMO: system overview and integration," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2002, pp. 2478–2483.

[3] A. H. A. Widaa and W. A. Talha, "Design of Fuzzy-based autonomous car control system," *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, 2017, pp. 1–7.

[4] J. L. Raheja, B. Ajay, A. Chaudhary, "Real time fabric defect detection system on an embedded DSP platform," *Optik - International Journal for Light and Electron Optics*, Volume 124, Issue 21, 2013, Pages 5280-5284.

[5] H. Zhou, R. Lai, S. Liu, B. Wang, Q. Li, "A new real-time processing system for the IRFPA imaging signal based on DSP&FPGA," *Infrared Physics & Technology*, Volume 46, Issue 4, 2005, Pages 277-281.

[6] D. Rong, Y. Ying, X. Rao, "Embedded vision detection of defective orange by fast adaptive lightness correction algorithm," *Computers and Electronics in Agriculture*, Volume 138, 1 June 2017, Pages 48-59.

[7] M. Chouchene, F. E. Sayadi, H. Bahri, J. Dubois, J. Miteran, M. Atri," Optimized parallel implementation of face detection based on GPU component," *Microprocessors and Microsystems*, Volume 39, Issue 6, 2015, Pages 393-404.

[8] X. Ma, W. A. Najjar and A. K. Roy-Chowdhury, "Evaluation and Acceleration of High-Throughput Fixed-Point Object Detection on FPGAs," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 6, June 2015, pp. 1051-1062.

[9] A. Irwansyah, O. W. Ibraheem, J. Hagemeyer, M. Porrmann, U. Rueckert, "FPGA-based multi-robot tracking", *Journal of Parallel and Distributed Computing*, Volume 107, September 2017, Pages 146-161.

[10] P. Reichel, J. Döge, N. Peter, C. Hoppe and A. Reichel, "An ASIP-based control system for Vision Chips with highly parallel signal processing," *2015 IEEE 24th International Symposium on Industrial Electronics (ISIE)*, Buzios, 2015, pp. 932-937.

[11] Yiannis Aloimonos, *Active perception*, 2013, Psychology Press.