Technical University of Crete
School of Electronic and Computer Engineering

Development of an advanced platform for controlling the imaging parameters of CMOS image sensors

Katerina Trilyraki

Thesis Committee
Associate Professor Costas Balas (ECE)
Associate Professor Yiannis Papaefstathiou (ECE)
Assistant Professor Matthias Bucher (ECE)

Chania, June 2014

Abstract

In the last few years it is witnessed an unbowed trend to use digital cameras in more and more aspects of our everyday life. This has led to the development of a special group of devices that constitute combinations of embedded computers and video cameras; the so-called Smart Cameras. Smart Cameras are embedded platforms with small form factor, equipped with a single or multiple video sensors and enough computational resources to perform dedicated operations onboard and at site. In the context of this thesis, a hardware platform was developed, capable of controlling both the behavior and output of Metal-Oxide-Semiconductor (CMOS) image sensors. The platform was developed using commercial-off-the-self products, where a CMOS image sensor was interfaced to a heterogeneous multicore processing module. Four applications were developed for this platform, embedding video streaming with signal processing operations, effectively transforming the hardware platform into a smart camera. In "De-noising during capture" application, multiple frames were averaged during acquisition to address the problem of image noise reduction. The second application, "Ultra-fast Spectrometer", utilizing only the rows of the image sensor's array, converts the camera to an ultra-fast reconfigurable spectrometer. In "Human Vision Emulation", we developed an application in which both an overview of a scene (not in full resolution) accompanied with an area of interest in full resolution is simultaneously displayed in accordance with the human vision sensitivity model. Finally, in "Pseudocolor Mapper", the platform calculates a pseudocolor map based on the red to green channel ratio, which is an indicative index for locating veins in human body.

Acknowledgements

Firstly, I would like to thank my advisor Costas Balas for his trust and guidance during the course of this thesis.
Next, I would like to thank the members of the "Optoelectronics Laboratory" namely, Vassilis Kavvadias, Giwrgos Epitropou, Giwrgos Papoutsoglou, Dimitris Iliou, Nikos Tsoulidis ,Thodoris-Marios Giakoumakis and Christos Rwssos, who create a fun working environment, shared ideas with me and supported me. Special gratitude must be addressed to ph.D candidate Thanasis Tsapras; for without his guidance and help this thesis would not have been completed.
Big thanks go to my TUC-friends Billy (Dancing Queen),Eleni (Ksemparki),Eleni (Paoki),Katerina (Ksan8ia), Kostas (Mpouklas), Marina (Karma), Nikos (Ypoulos), Nikolas (Pri…) , who withstood me and gave some great moments to remember.
Last but not least, I would like to thank my family and friends for their love, support and constant encouragement.

Contents

List of Figures

List of Tables

1 Introduction

In the last few years it is witnessed an unbowed trend to use digital cameras in more and more aspects of our everyday life. This fact is attributable to two reasons: the first one is the decline in the price of image sensors that resulted in the deployment of cameras in objects of personal use, like smart phones and PDAs, or of public use like traffic and security surveillance. The second reason is the increase in computational power per processor unit enabling small footprint processor not only to handle bulky imaging data but also fulfill computationally intensive imaging tasks. These two facts led to the development of a special group of devices which constitute combinations of embedded computers and video cameras; the so-called Smart Cameras. Smart Cameras are embedded platforms with small form factor equipped with a single or multiple video sensors and enough computational resources to perform dedicated operations onboard and at site. Additional advantages of these platforms are their low production costs, their low power consumption and their robustness to environmental stress. There is a wide area of applicability for these types of devices: care in residential home for the elderly, industrial robotics, domestic home care, and clearly all types of surveillance of public places, like traffic surveillance or access control in public transport systems. The number of applications related to such platforms is extraordinary large and the list is steadily getting longer.
The target of this work is to create a smart camera platform; More particularly, we are interested in setting up a smart camera platform in order to, primarily, explore both the platform and the image sensor capabilities and extents and then to add smart features in it that are mainly based on the flexibility of the readout architecture of CMOS image sensors.

1.1 Thesis contribution

This thesis describes the development of a hardware platform capable of controlling the imaging parameters of a Metal-Oxide-Semiconductor (CMOS) image sensor. More particularly, our aim was using commercial off-the-self products to develop a low cost platform that would offer the greatest possible extent of controllability over the behavior and the output data of a CMOS image sensor. In other words, we wanted to develop a smart camera for CMOS image sensors. At this point it is worth mentioning, that there is a wide variety of such platforms available, but they are usually offered as a unique pair of camera module and processing module limiting the universality of the platform and the extent to which a programmer can intervene to that high-end product. After extended research, we ended up with a heterogeneous multicore processing platform that is not exclusively targeted for imaging/video applications and a CMOS image sensor capable of being interfaced to that platform.
The motivation for gaining control over the CMOS image sensor parameters (behavior) and output derived from the unique attribute of CMOS image sensors, namely random accesses and retrieval of arbitrary –sized areas of the sensor's imaging array (windowing). This possibility is very useful, since it has been proven that in most image/video processing algorithms, most of the times, only a window of an image is the actual needed data. In addition, windowing can be used in order to make efficient use of the bandwidth either that is towards a display or a connection to the outside world; only the needed data can be transferred instead of the whole image.
From motivation to the development of smart applications there has been an interval dedicated to evaluating the possibilities and the performance of the processing module and the image sensor.

The applications that outsmarted the hardware platform developed are:

- **De-noising During Capture:** Noise removal by averaging a number of incoming frames so as to render a noise-reduced frame.
- **Ultra-fast Spectrometer:** Utilizing rows of the imaging array of the sensor as a spectrometer detector, an ultra-fast reconfigurable spectrometer can be developed.
- **Human Vision Emulation:** As it had been putted, "if the eye can do it, so can the machine."[1].The human eye is not uniformly sensitive; The central area of the retina (photosensitive tissue of human eyes) is called macula and it is responsible for detailed vision such as reading or seeing details straight ahead .The remaining area of the retina outside of the macula is responsible for peripheral vision. Functionally ,the macula is responsible for "what is it" vision, whereas the function of the peripheral retina which is responsible for "where is it" vision. In accordance, we developed an application in which the image sensor displays both an overview of a scene (not in full resolution), accompanied with an area of interest in full resolution.
- **Pseudocolor Mapper:** In case "the eye cannot do it, maybe the machine will". There are cases that information can be better represented in a non-conventional way (interpolation

of red green and blue values). Other relationship between color bands can be used depending on what we are interested to image. In this context, the platform can perform pseudochromatic coloring based on red to green channel ratio, an indicative index for locating veins in human body.

1.2 Thesis outline

In **Chapter 2** we present all the background information needed for this thesis. We give an overview of the functionality of the two dominant technologies built around solid state imagers, namely Charged-Coupled Devices (CCDs) and Metal-Oxide-Semiconductor (CMOS) in order to be able to compare them with respect to different aspects and justify the decision to work with CMOS image sensors. In **Chapter 3** a reference to related work is made. In the next chapter, **Chapter 4**, a brief description of the structure and functionality of the hardware platform will take place; firstly the image sensor and then the processing platform will be described. **Chapter 5** follows with the description of the base software layer along with the development of the fundamental functionality of video streaming. The chapter concludes with the performance evaluation of video streaming. In **Chapter 6** the four applications developed based on video streaming are described. The description starts with the "De-noising during capture" application, followed by "Ultra-fast Spectrometer" and "Human Vision Emulation" applications. Last but not least, the "Mapper" application is described. Finally, **Chapter 7** acts as an epilogue for this thesis, presenting our conclusions accompanied with proposed future improvements.

2 Background

2.1 Solid State Imagers

The arrival of high-resolution solid state imaging devices, primarily Charged-Coupled Devices (**CCDs**) and afterwards complementary Metal-Oxide-Semiconductor (**CMOS**) image sensors has heralded a new era for image recording technology and overrode preceding technologies such as film, video tubes and photomultipliers. Both technologies, on which CCDs and CMOS image sensors (**CIS**) are based, were proposed between the early and late 1970s, back then CCDs became dominant, mainly because they delivered far superior images with the fabrication technology available. CMOS required more uniformity and smaller features than silicon wafer foundries could give at that time. It was in the early 1990s that lithography was developed to the point that designers could begin making a case for CMOS imagers again.

Charged-coupled devices and CMOS image sensors have the ability to convert an optical image to electronic signals by taking advantage of the same physical phenomenon, namely, the photoelectric effect. The **photoelectric effect** occurs when electrically charged particles are released from or within a material when it absorbs electromagnetic radiation.

In terms of image sensors functionality, the photoelectric effect can be described as the phenomenon in which photons of visible light interact with crystallized silicon to promote electrons from the valence band into the conduction band (Figure2.1). More precisely, when a broad wavelength band of visible light is incident on specially doped silicon semiconductor materials (from which image sensors are constructed), a variable number of electrons are released in proportion to the photon flux density incident on the surface of the semiconductor [5].

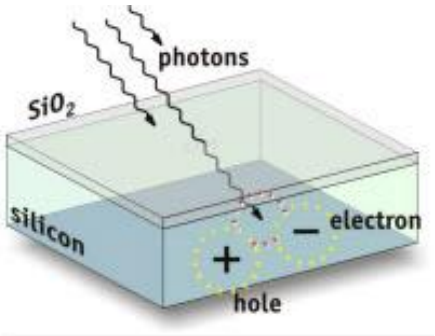


Figure2.1: CCD and CMOS imagers both depend on the photoelectric effect to create electronic signals from light.

In effect, the number of electrons produced is a function of the wavelength and the intensity of light striking the semiconductor. In other words, the electronic signals produced by an image sensor are analogous to the properties of light deriving from the scene that the sensor images.

Both CCD and CMOS convert incident light (photons) into electronic charge (electrons) by the same photo-conversion process. The produced electrons are collected in a potential well until the illumination period is finished afterwards they are either converted into a voltage (CMOS sensors) or transferred to a metering register (CCD sensor). The measured voltage or charge (after conversion to a voltage) is then fed to an analog-to-digital converter, which produces a digital representation of the scene imaged by the sensor. It can be inferred from the above-mentioned that the factor that differentiates these two types of sensors is what happens after illumination period is finished.

The key element of a digital image sensor is its photosensitive element. There are two basic photosensitive pixel element architectures utilized by both CCD and CMOS imagers (Figure2.2):

- **Photodiode:** a metallurgical junction (pn-junction in the silicon substrate) is used to form a photodiode. A depletion layer or "potential well" is formed around this junction by means of diffusing impurities or "implants" into the silicon.
- **Photogate:** a voltage induced junction by means of a MOS capacitor. A voltage is applied to a polysilicon gate to induce a potential well in the silicon.

In either photosensitive element, photon-generated charge will decrease the width of the depletion layer (fill the well) proportional to the amount of light being absorbed by the pixel. In general, photodiode designs are more sensitive to visible light, especially in the short-wavelength (blue) region of the spectrum. Photogate devices usually have larger pixel areas, but a lower fill factor and much poorer blue light response (and general quantum efficiency) than photodiodes. However, photogates often reach higher charge-to-voltage conversion gain levels and can easily be utilized to perform correlated double sampling to achieve frame differencing.

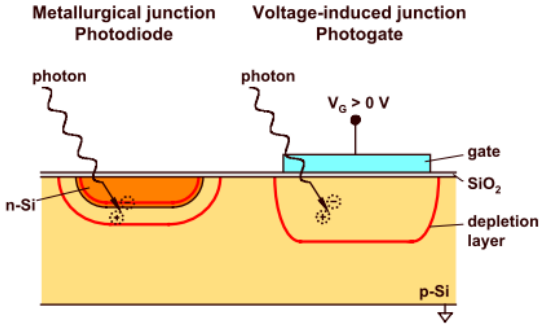


Figure2.2: The photosensitive element on the left pixel is a photodiode and on the right a photogate.

Facts that enables photogate designs to have reduced noise features when operating at low light levels, as compared to photodiode sensors. Photodiode-based sensors are useful for mid-level performance consumer applications that do not require highly accurate images with low noise, superior dynamic range, and highly resolved color characteristics[5].

In summary to this point, charge can be "photo-generated", how this photo-generated charge will be handled depends on what type of image sensor t is used; CCD or CIS.

2.2 CCD area imager sensors

A Charge-Coupled Device (CCD) is a type of charge storage and transport device. It consists of a large number of light-sensing elements arranged in a two-dimensional array on a thin silicon substrate. The semiconductor properties of silicon allow the CCD chip to trap and hold photon-induced charge carriers under appropriate electrical bias conditions. Individual picture elements, or pixels, are defined in the silicon matrix by an orthogonal grid of narrow transparent current-carrying electrode strips, or gates, deposited on the chip. The fundamental light-sensing unit of the CCD is a Metal Oxide Semiconductor (**MOS**) capacitor operated as a photodiode and storage device. A single MOS device of this type is illustrated in Figure 2.3.

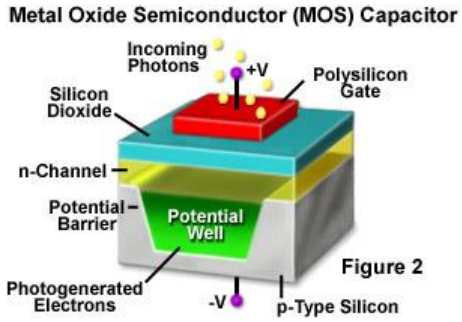


Figure2.3: Fundamental light-sensing unit of CCD

Image generation with a CCD camera can be divided into four primary stages or functions:

- charge generation through photon interaction
- collection and storage of the liberated charge
- charge transfer
- charge measurement

During the **first stage**, electrons and holes are generated in response to incident photons in the depletion region of the MOS capacitor structure. In the **second stage**, the electrons generated in the depletion region are initially collected into electrically positive potential wells associated with each pixel. Before stored charge from each sense element in a CCD can be measured to determine photon flux on that pixel, the charge must first be transferred to a readout node while maintaining the integrity of the charge packet. So during the **third stage**, charge is moved across the device by manipulating voltages on the capacitor gates in a pattern that causes charge to spill from one capacitor to the next or from one row of capacitors to the next. The translation of charge within the silicon is effectively coupled to clocked voltage patterns applied to the overlying electrode structure, the basis of the term **"charge-coupled"** device. At the **fourth stage**, each charge packet is delivered to the CCD's output node where it is detected and read by an output that converts the charge into a proportional voltage. After the output amplifier fulfills its function of magnifying a charge packet and converting it to a proportional voltage, the signal is transmitted to an analog-to-digital converter (ADC) [3].

A fast and efficient charge-transfer process is crucial to the function of CCDs as imaging devices. As a result there are three basic variations of CCDs architecture (see Figure2.4) based on different charge transferring schemes:

- **full- frame:** A full-frame CCD has the advantage of nearly 100-percent of its surface being photosensitive, with virtually no dead space between pixels. The imaging surface must be protected from incident light during readout of the CCD, and for this reason, an electromechanical shutter is usually employed for controlling exposures. Charge accumulated with the shutter open is subsequently transferred and read out after the shutter is closed, and because the two steps cannot occur simultaneously, image frame rates are limited.
- **frame transfer:** Frame-transfer CCDs can operate at faster frame rates than full-frame devices because exposure and readout can occur simultaneously with various degrees of overlap in timing. They are similar to full-frame devices in structure, but one-half of the rectangular pixel array is covered by an opaque mask, and is used as a storage buffer for photoelectrons gathered by the unmasked light-sensitive portion. A camera shutter is not necessary because the time required for charge transfer from the image area to the storage area of the chip is only a fraction of the time needed for a typical exposure. The major disadvantage of this sensor type is that only one-half of the surface area of the CCD is used for imaging.

- **interline transfer:** In the interline-transfer CCD design, columns of active imaging pixels and masked storage-transfer pixels alternate over the entire parallel register array. Because a charge-transfer channel is located immediately adjacent to each photosensitive pixel column, stored charge must only be shifted one column into a transfer channel. The interline-transfer architecture allows very short integration periods through electronic control of exposure intervals, and in place of a mechanical shutter, the array can be rendered effectively light-insensitive by discarding accumulated charge rather than shifting it to the transfer channels

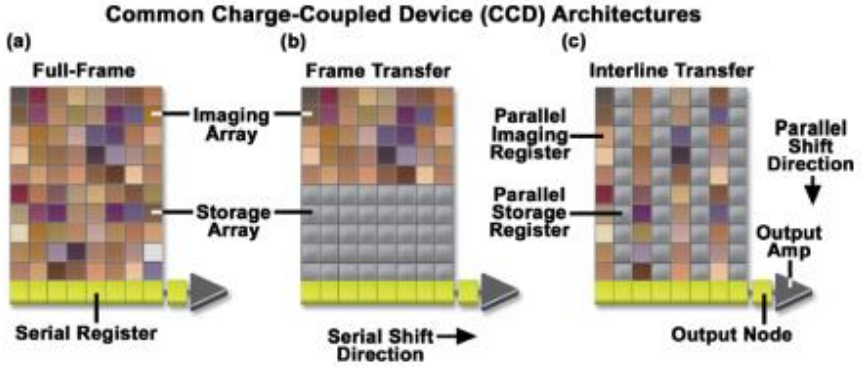


Figure2.4: Common transfer architectures of CCDs

2.3 CMOS image sensors

In general terms, a two-dimensional CMOS Image Sensor (CIS) has an architecture very similar to a SRAM or DRAM chip regardless of the specifics of the pixel architecture. Through the vertical scan circuit any particular row of pixels can be addressed and with the horizontal scan circuit a single pixel out of the selected row can be addressed. In this way, individual pixels can be randomly selected and have their signal outputted.

There are three main architectures for CMOS imagers [4]:

- **Passive Pixel Sensor (PPS):** This architecture suggests the simplest structure for the pixel(see Figure2.5 (a)),; a photodiode, a transistor and two interconnects. The operation of this pixel is very simple; after addressing the pixel by opening the row-select (RS) transistor, the pixel is reset along the column bus and trough RS becoming ready for the next exposure period. This type of pixels are characterized by a large fill factor and suffer from high noise levels due to the mismatch between the small photodiode's capacitance and the large vertical and horizontal buses capacitances.

- **Active Pixel Sensor (APS):** In this architecture every pixel gets its own amplifier (see Figure2.5(b)), which is configures as a source follower and its purpose is to boost the photodiode signal fed to the column line, solving the problem of capacitance mismatch mentioned in the previous bullet. So an active pixel contains a photodiode, three transistors and four interconnects. The operation of this pixel is as follows: After a pixel is addressed by opening the row-select (RS) transistor, its charge is sensed and fed to the column bus by the source follower, afterwards the pixel is being reset by the reset (RST) transistor. The introduction of an amplifier in each pixel increases the non-uniformity between pixels analog nature resulting in the fixed- pattern noise effect. Also APS CMOS suffer from a large reset noise component.
- **Digital Pixel Sensor (DPS):** Going one step further in integration, in this architecture every pixel gets its own analog to digital converter (ADC).Every column has a complete digital bus as one can see in Figure2.5(c).

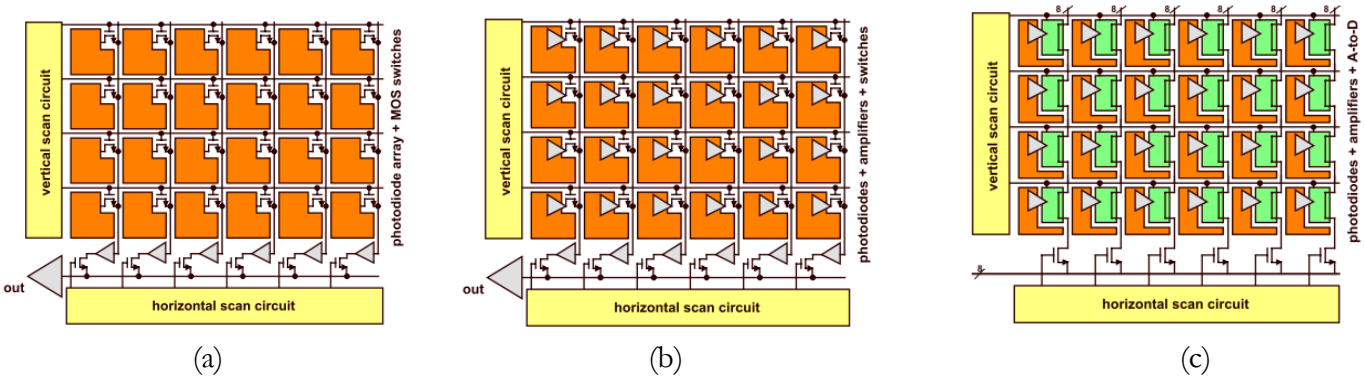


Figure2.5: Three main architectures of CMOS image sensors

The most popular designs are built around Active Pixel Sensor (APS) architecture which includes many flavors of pixel structure such as the pinned photodiode 4-transistor pixel, the pinned photodiode 5-transistor pixel and the logarithmic pixel. So we will give a brief description of a CMOS image sensor functionality according to this architecture.
Each pixel (imaging element) contains a photodiode and a triad of transistors that converts accumulated electron charge to measurable voltage, resets the diode and transfers the voltage to a vertical column bus (Figure 2.6). More precisely, the types of these transistors are [5]:

- **Source follower transistor:** a simple amplifier transistor that converts the electrons generated by the photodiode into a voltage that is output to the column bus (the load of the source follower being external to the pixel and common to all the pixels in a column)
- **Reset transistor:** the reset transistor is used to control integration or photon accumulation time, and a row-select transistor that connects the pixel output to the column bus for readout

- **Row-select transistor:** a row-select transistor that connects the pixel output to the column bus for readout.

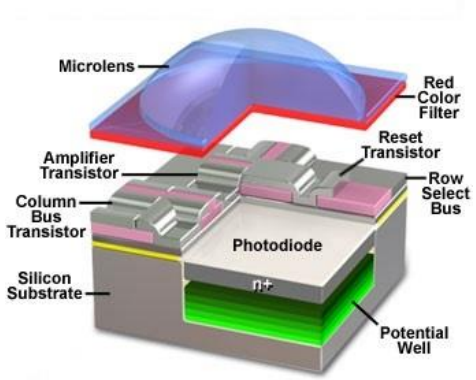


Figure2.6: CMOS typical active sensor pixel accompanied with microlens and a (red) filter color filter.

At this point we have to mention that all of the pixels in a particular column connect to a sense amplifier.
In operation, the first step toward image capture is to initialize the reset transistor in order to drain the charge from the photosensitive region and reverse bias the photodiode. Next, the integration period begins, and light interacting with the photodiode region of the pixel produces electrons, which are stored in the silicon potential well lying beneath the surface (see Figure2.6). When the integration period has finished, the row-select transistor is switched on, connecting the amplifier transistor in the selected pixel to its load to form a source follower. The electron charge in the photodiode is thus converted into a voltage by the source follower operation. The resulting voltage appears on the column bus and can be detected by the sense amplifier. This cycle is then repeated to read out every row in the sensor in order to produce an image.
Having described both CCD and CMOS architecture and functionality a comparison between then in general and in the scope of this thesis follows.

2.4 CCD versus CMOS

2.4.1 Architecture

Both CCD and CMOS devices perform the task of converting light into charge. The next step is to read the value (accumulated charge) of each cell in the image. In a CCD device, the charge is actually transported across the chip and read at one corner of the array. An analog-to-digital converter turns each pixel's value into a digital value. In most CMOS devices, there are several

transistors at each pixel that amplify and move the charge using more traditional wires. The CMOS approach is more flexible because each pixel can be read individually.

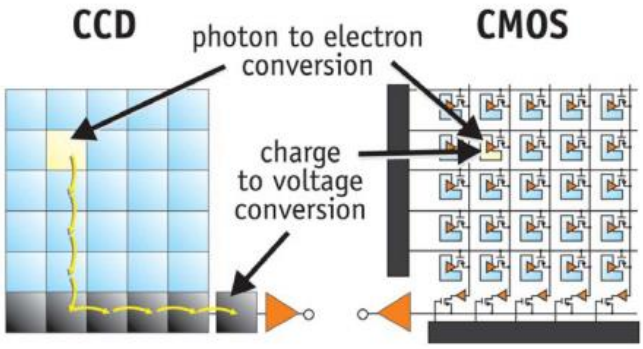


Figure2.7: Architectural comparison of CCD and CMOS image sensors.

2.4.2 Image Quality

Despite improvements in CMOS, CCD still maintains an advantage, albeit a shrinking one, for applications that require high-quality images. The key reasons for this fact are summarized in the following bullets [8]:

- CCD sensors have been mass-produced for a longer period of time, so they more mature and more optimized. As a consequence, CCDs tend to render high-quality and low-noise images.
- CMOS sensors have a larger pixel-to-pixel non-uniformity because each pixel has its own readout circuitry and in most cases column ADCs are used (in CCD many pixels share the same output; ADCs are external.). Although offset and gain corrections are possible, linearity differences in CMOS are inherent due to manufacturing process and very difficult to eliminate.
- Dark current of the best CCD sensors is still better than of CMOSCMOS sensors also have more problems with optical crosstalk due to the thicker stack on top of the pixels. This results in a lower accurate color reproduction, especially with small pixels. Also, the MTF of a CMOS sensor is typically worse than that of a CCD. This is due to higher optical and electrical crosstalk; a problem than again increases as the pixel size gets smaller.
- In CMOS image sensors only a small portion of the photodiode is actually capable of absorbing photons to generate charge, the fill factor or aperture of the CMOS chip and photodiodes represents only 30 percent of the total photodiode array surface area. The consequence is a significant loss in sensitivity and a corresponding reduction in signal-to-noise ratio, leading to a limited dynamic range.

CMOS sensors are constantly being improved and they have got to the point where they reach parity with CCD devices in terms of image quality in most applications, but still CCDs maintain an advantage.

2.4.3 Power consumption

CMOS sensors traditionally consumes little power, on the other hand CCDs use a special process that consumes lots of power. A CCD consumes as much as 100 times more power than an equivalent CMOS sensor.

2.4.4 Level of integration

CMOS and CCD image sensors differ in the degree of integration available to the designer when using each technology. Simply put, a CCD is comprised of the pixel array and an analog output stage. A CMOS imager on the other hand may consist of not only the pixel array and an analog output stage, but the complete analog signal chain as well as the complete digital control logic section. In other words, CMOS image sensors have the ability to integrate a number of processing and control functions which lie far beyond the task of photons collection [7] .

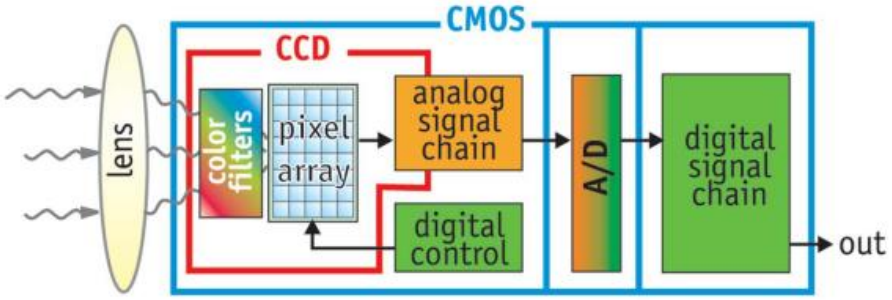


Figure2.8: Orange areas highlight the analog circuits whereas green colored the digital circuits

2.4.5 Manufacturing Cost

Very often it is claimed that CMOS image sensors are fundamentally cheaper than CCD devices because they can be made on standard process and leverage the economies of scale associated with running large volumes of wafers foundry. This is a dangerous argument, firstly, because standard digital CMOS manufacturing processes are optimized for good digital signal processing and not for good imaging performance. Thus, if CMOS image sensors with good imaging performance are to be rendered, one has to move away from the mainstream process and make the appropriate customizations, fact that results in an analogous cost raise. Secondly, even if one assumes that

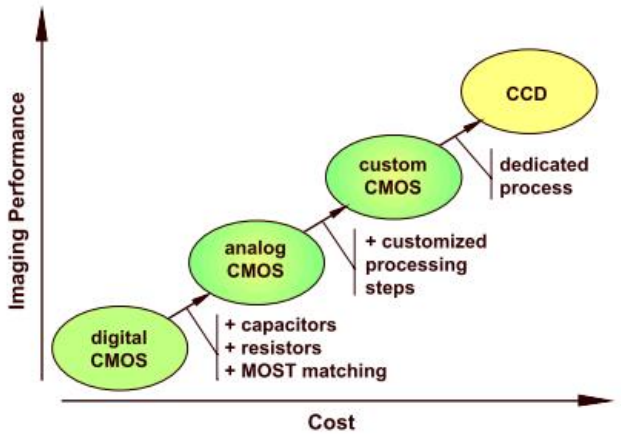


Figure2.9: Imaging performance in terms of Cost

wafers are run on a completely standard process, the cost difference between CMOS and CCD can only be gained back in the diffusion cost of the wafers and there are many more process steps required before a working device can be yielded. Furthermore, for small devices most of the cost goes into the deposition of color filters, deposition of micro-lenses, on wafer-testing, packaging and final testing. To sum up the above- mentioned, the cost advantage of CMOS technology becomes only marginal as far as the image sensors themselves are concerned. The cost of good imaging performing CMOS sensors is increasing in proportion to the customization in CMOS manufacturing.

2.5 CMOS in the scope of this thesis

The previous-mentioned categories of comparison between CCD and CMOS were not the one that determined our decision to work with CMOS image sensors. As it had been mentioned in Chapter1 we were mainly motivated by the ability of CMOS sensors to read portions of the imaging array (windowing). This ability of CMOS image sensors is closely related to two other advantages of CMOS over CDDs; these are higher frame rates and higher speed. A few words about the correlated ideas of windowing frame rate and speed follow.

Frame rate

One of the most versatile capabilities of CMOS image sensors is their ability to capture images at very high frame rates. For CCD sensors, the speed of delivery for pixel data sets the upper limit for frame rate. This limit arises because a CCD sensor must transfer out all of its pixel information in order to empty its transfer registers so that they
can accept the next image. For a given pixel rate, then, the larger the image the lower the frame rate. The same holds true for linear sensors, but the tradeoff is less. It is because CMOS sensors

convert charge to voltage at each pixel that its amplifiers do not need to be high speed in order to support a fast frame rate. Thus, CMOS sensors can achieve faster frame rates more easily than CCDs. Further, unlike CCDs, the image data of CMOS sensors can be cleared without having to be read. This allows the machine vision system to read out only a portion of the image information, working with an area of interest within the image. By reading out only the area of interest, CMOS sensors can support a faster frame rate without increasing the pixel rate[10].

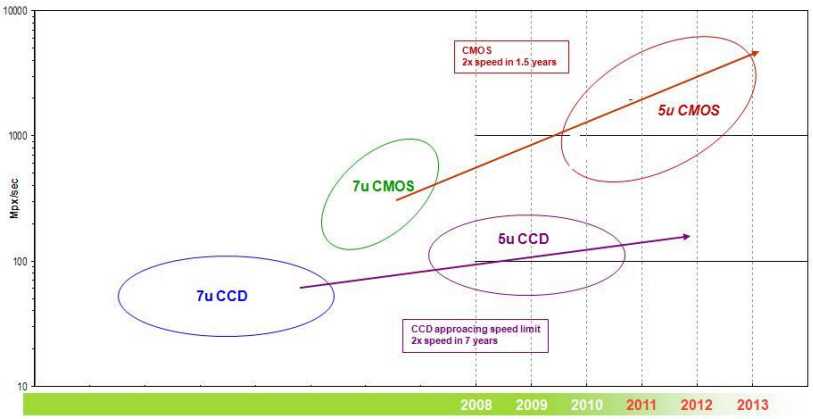


Figure2.10: Frame rate is increasing with CCD being unable to compete with CMOS

Speed

An area in which CMOS arguably has the advantage over CCDs because all camera functions can be placed on the image sensor. With one die, signal and power trace distances can be shorter, with less inductance, capacitance and propagation delays. To date, though, CMOS imagers have established only modest advantages in this regard, largely because of early focus on consumer applications that do not demand notably high speeds compared with the CCD's industrial, scientific and medical applications.

Windowing

One unique capability of CMOS technology is the ability to read out a portion of the image sensor. This allows elevated frame or line rates for small regions of interest. This is an enabling capability for CMOS imagers in some applications, such as high-temporal-precision object tracking in a sub-region of an image. CCDs generally have limited abilities in windowing [9].

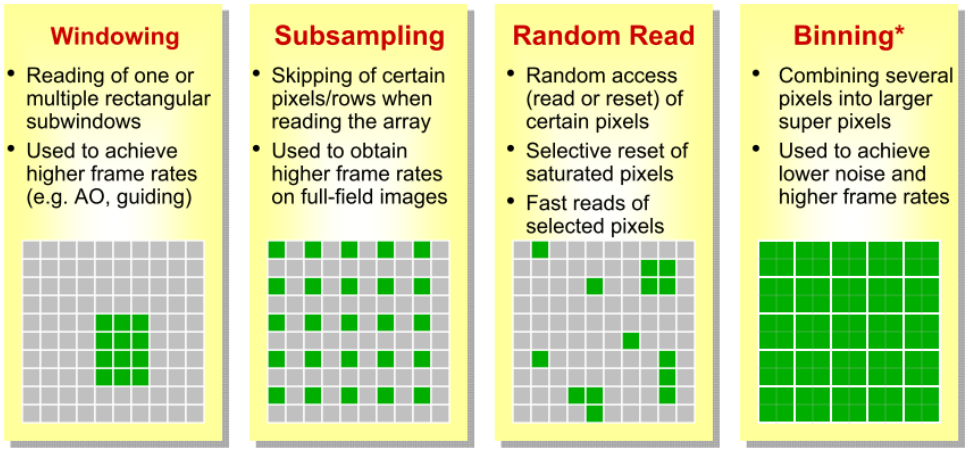


Figure2.11: Different scanning methods are available to reduce the numbers of being read rendering higher framer rate

2.6 Hardware Platforms for Real-Time Image and Video Processing

It is known that in image/video processing systems the most resource demanding operations in terms of required computations and memory bandwidth involve pixel- level data (low-level) operations, as a result considerable research has been devoted to developing hardware architectural features for eliminating bottlenecks within the image/video processing chain, freeing up more time for performing high-level interpretation operations.

From the literature, one can see there are three major hardware architectural features that are essential to any image/video processing system [11]:

- **Single Instruction Multiple Data (SIMD):** The concept of SIMD embodies broadcasting a single instruction to multiple processors, which simultaneously execute the instruction on different portions of data in parallel, thus allowing more computations to be performed in a shorter time [12]. The most common instantiation of the concept of SIMD in today's GPPs, digital signal and media processors, is in the form of the packed data processing extension.
- **Very Long Instruction Word (VLIW):** While SIMD can be used for exploiting data level parallelism (DLP), VLIW can be used for exploiting instruction level parallelism (ILP) [65], and thus for speeding up high-level operations [13]. VLIW furnishes the ability to execute multiple instructions within one processor clock cycle, all running in parallel, hence allowing software-oriented pipelining of instructions by the programmer. At this point it

must me mentioned that the ability to execute more than one instruction per clock cycle is essential for image/video processing applications that require operations in the order of giga operations per second [13].

- **Efficient Memory Subsystem:** Of course, while SIMD and VLIW can help speed up the processing of diverse image/video operations, the time saved through such mechanisms would be completely wasted if there did not exist an efficient way to transfer data throughout the system [14]. Concepts such as direct memory access (DMA) and internal versus external memory are important. DMA allows transferring of data within a system without burdening the CPU with data transfers. DMA is a well-known tool for hiding memory access latencies, especially for image data. Efficient use of any available on-chip memory is also critical since such memory can be accessed at a faster rate than external memory.

An overview of the standard processor architectures and their advantages/disadvantages for real-time image/video processing follows.

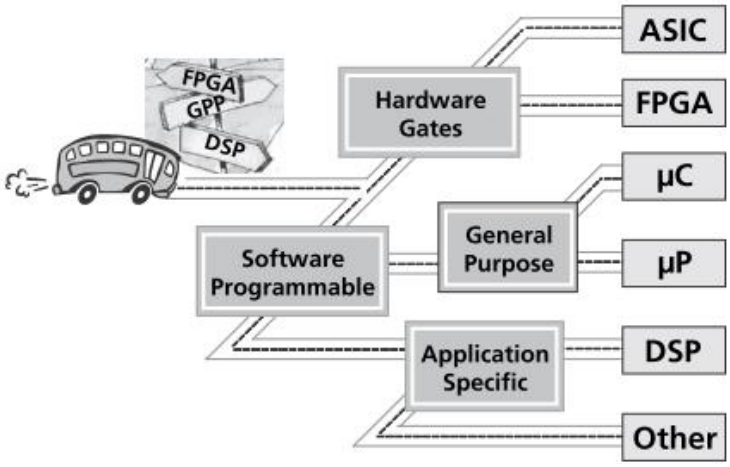


Figure2.12: Different hardware platforms for image/video applications

Digital Signal processors

Digital Signal processors are well known for their high-performance, low-power characteristics and relatively small footprint, which enable them to accelerate computationally intensive tasks on embedded devices. While it may have been true in the past that digital signal processors (DSPs) not suitable for processing image/video data in that they could not meet real-time requirements for video rate processing, this is no longer the case with newly available high-performance DSPs that contain specific architectural enhancements addressing the data/computation throughput barrier. DSPs have been optimized for repetitive computation kernels with special addressing modes for signal processing such as circular or modulo addressing. This helps to accelerate the critical core

routines within inner loops of low- and intermediate-level image/video processing operations. In many DSP implementations, it is observed that a large percentage of the execution time is due to a very low percentage of the code, which simply emphasizes the fact that DSPs are best for accelerating critical loops with few branching and control operations, which are best handled by a GPP [15]. DSPs possess either a fixed-point or a floating-point CPU, depending on the required accuracy for a given application. In most cases, a fixed-point CPU is more than adequate for the computations involved in image/video processing. DSPs also have predictable, deterministic execution times that constitute a critical feature for ensuring that real-time deadlines are met. In addition, DSPs have highly parallel architectures with multiple functional units and VLIW/SIMD features, further proving their suitability for image/video processing. It is critical that DSPs have been designed with high memory bandwidth in mind, on-chip DMA controllers, multilevel caches, buses, and peripherals, allowing efficient movement of data on- and off-chip from memories and other devices. DSPs support the use of real-time operating systems (RTOSs), which again help in guaranteeing that critical system level hard real-time deadlines are met. Of course, DSPs are fully programmable, which adds to their inherent flexibility to changes in algorithm updates. Indeed, DSPs contain specific architectural features that help one to speed up repetitive, compute-intensive signal processing routines, making them a viable option for inclusion in a real-time image/video processing system. That is why DSPs have been used in many real-time image/video processing systems. More recently, DSPs have been included as a core in dual-core processor system-on-chips for consumer electronics devices such as PDAs, cell phones, digital cameras, portable media players, etc.

Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are reconfigurable integrated circuits. They contain arrays of complex logic blocks along with a grid of programmable interconnects that allows them be inter-wired. FPGAs allow the design of fully application-specific custom circuits via a software programming language known as hardware description language (HDL).Current generation FPGAs can be either fully reconfigured or partially reconfigured, with reconfigurable times of less than a 1 ms, making it possible to have a dynamic run-time reconfiguration.
Due to their programmable nature, FPGAs can be programmed to exploit different types of parallelism inherent in an image/video processing algorithm. This in turn leads to highly efficient real-time image/video processing for low-level, intermediate-level, or high-level operations, enabling an entire imaging system to be implemented on a single FPGA.
In general, FPGAs have extremely high memory bandwidth. As a result, one can use custom memory configurations and/or addressing techniques to exploit data locality in high-dimensional data. In many cases, FPGAs have the potential to meet or exceed the performance of a single DSP or multiple DSPs.
FPGAs can be thought of as combining the flexibility of software programmability with the speed of an application-specific circuit (ASIC) within a shorter design cycle or time-to-market. However,

there is a disadvantage associated with FPGAs, that is, their energy or power consumption efficiency. In essence, FPGAs have high computational and memory bandwidth capabilities that are essential to real-time image/video processing systems. Because of such features, FPGAs have already been used to solve many practical real-world, real-time image/video processing problems, from a preprocessing component to the entire processing chain.
FPGAs have also been used in conjunction with DSPs. A current trend in FPGAs is to include a GPP core on the same chip as the FPGA for a customizable system-on-chip (SoC) solution.

Multicore Embedded System-on-Chip

In the consumer electronics market, there has been a drive toward single-chip solutions or SoCs for portable embedded devices, which require high performance computation and memory throughput coupled with low power consumption in order to meet the real-time image/video processing constraints of battery-powered products such as digital cameras, digital video camorders, cell-phone-equipped cameras, etc.
In the consumer electronics market, there has been a drive toward single-chip solutions or SoCs for portable embedded devices, which require high performance computation and memory throughput coupled with low power consumption in order to meet the real-time image/video processing constraints of battery-powered products such as digital cameras, digital video camorders, cell-phone-equipped cameras, etc.
These systems exhibit elegant designs where one can learn how the industry has approached the battery-powered embedded real-time image/video processing problem. For example, consider the TMS320DM320 "digital media processor" manufactured by Texas Instruments [16]. This is a multiprocessor chip with a reduced instruction set (RISC) microprocessor coupled with a low-power fixed-point DSP. The RISC microprocessor serves as the master handling system control, running a RTOS and providing the necessary processing power for complex control-intensive operations. The DSP, acting as a slave to the RISC, is a low-power component for performing computationally intensive signal processing operations. The presence of a memory traffic controller allows achieving a high-throughput access to memory. In this device, the RISC and DSP are accompanied by a set of parameter customizable application-specific processors that provide a "boost," that is to say, they provide the extra computational horsepower that is necessary to perform functions such as real-time LCD preview (Preview Engine) and real-time computation of low-level statistics necessary for autoexposure, autowhite balance, and autofocus (H3A Engine). captured image through the image pipeline and running image/video compression routines.
By examining this architecture, one can see that this SoC has been designed with a DSP plus dedicated hardware accelerators for low-level and intermediate-level operations along with a GPP hardware for more complex high-level operations. This is an illustrative example showing that a complete real-time image/video processing system can be characterized as a heterogeneous architecture with a computation-oriented front end coupled with a general-purpose processing back end.

General-Purpose Processors

There are two types of GPPs on the market today, one geared toward non-embedded applications such as desktop PCs and the other geared toward embedded applications. Today's desktop GPPs are extremely high-performance processors with highly parallel architectures, containing features that help to exploit ILP in control-intensive, high-level image/video operations. SIMD extensions have also been incorporated in their instruction sets allowing such processors to exploit DLP and enabling moderate acceleration of multimedia operations corresponding to low-level and intermediate-level image/video processing operations. GPPs have been outfitted with the multilevel cache feature. This feature provides the potential of having low latency memory accesses for frequently used data. It is worth mentioning that these processors also require an RTOS in order to guarantee a real-time execution. Although GPPs have massive general-purpose processing power, they are extremely high-powered devices requiring hundreds of watts of power. Clearly such processors are not suitable for embedded applications. Despite this fact, advances in desktop GPPs have allowed the standard commercial off-the-shelf desktop PCs to be used for implementing non-embedded real-time image/video processing systems. It should be noted that such industrial inspection systems usually augment the processing power of the desktop GPP with vision accelerator boards. These boards often furnish a dedicated SIMD image/video processor for high-performance real-time processing not normally met by the SIMD extensions to the desktop GPP. Recently, a paradigm shift toward multicore processor designs for desktop PCs has occurred in order to continue making gains in processor performance.
On the embedded front, there are also several GPPs available on the market today with high-performance general-purpose processing capability suitable for exploiting ILP coupled with low power consumption and SIMD-type extensions for moderately accelerating multimedia operations, enabling the exploitation of DLP for low-level and intermediate-level image/video processing operations. Embedded GPPs have been used in multicore embedded SoCs, providing the horsepower to cope with control- and branch-intensive instructions.
Both embedded and desktop GPPs are supported by mature development tools and efficient compilers, allowing quick development cycles. While GPPs are quite powerful, they are neither created nor specialized to accelerate massively data parallel computations.

Graphics Processing Unit

The early 2000s witnessed the introduction of a new type of processor, the graphics processing unit (GPU). The primary function of such processors is for real-time rendering of three-dimensional (3D) computer graphics enabling fast frame rates and higher levels of realism required for state-of-the-art 3D graphics in modern computer games. While the original GPUs were fixed function accelerators, current generation GPUs incorporate more flexibility through ever-

increasing amounts of programmability with programmable vertex and texture/fragment units that are useful for customizing the rendering of 3D computer graphics.
GPUs can also be used for accelerating computations with inherent DLP. In terms of performance, it had been showed that GPUs can provide huge increases in GFLOPS performance and memory throughput over those of a high-performance desktop GPP[17].
Due to their floating-point calculation capabilities, the increased levels of programmability, and the fact that GPUs can be found in almost every desktop PC today, many researchers have been looking into ways to exploit GPUs for applications other than the real-time rendering of 3D computer graphics, an area of research referred to as general-purpose processing on the graphics processing unit (GPGPU). GPUs have already been deployed to solve real-time image/video processing problems including complete computer vision systems [18], medical image reconstruction in magnetic resonance imaging (MRI) and stereo depth map computation [19].
Image processing on a GPU can be performed by downloading an image to the GPU as a texture structure, rendering a rectangle the size of the image, and mapping the image as a texture structure to the rectangle, after which a kernel fragment program can be used to process the image taking advantage of the massive computation power of the GPU.
One key drawback of GPUs has been the data read-back throughput through the Peripheral Component Interconnect (PCI) bus, this is has been mitigated with the introduction of the PCI Express bus standard. One important item to note is that just like desktop GPPs, GPUs are also high-powered devices drawing hundreds of watts of power. Although low-power GPUs for embedded applications are becoming more available, it is currently not known how well these embedded GPUs will fare in GPGPU applications.

3 Related Work

Due to the great extent and variety in smart cameras we will limit the related work to three different smart cameras systems that each one of them has a common aspect with the hardware platform for controlling CMOS image sensors that we proposed.

3.1 Same motivation concerning CMOS image sensors

"Development of a high-resolution, high-speed vision system using CMOS image sensor technology enhanced by intelligent pixel selection technique"

It is often found that only a tiny portion within image frames shot by the camera is needed for final image processing in vision systems. So, in the Mm-Vision System, developers have tried to realize a small-scale and low-cost system, to avoid bottlenecks in communication and processing, by making it possible to transfer, from the entire image area only a local domain image containing necessary information; that is determined via a coordinate transformation circuit and given as a feedback to the system. Three tests for high speed target tracking were done with the Mm-Vision prototype system (CMOS image sensor interfaced to a PCI card).The first is "Window tracking using center of gravity", then it follows "Target tracking using template matching" and "Color tracking". The overall performance of Mm-Vision system can be improved in inverse proportion to the increasing amount of image processing and the size of the required local domain without ever adversely affecting the resolution of the imaging sensor[20].

3.2 Same processing module

"Design and implementation of an automatic traffic sign recognition system on TI OMAP-L138"

This work was developed in the same processing platform as the one used in this thesis. A system that detects and recognizes traffic signs present in an image was developed on TI OMAP-L138. Morphological operators, segmentation and contour detection are used for isolating the Regions of Interest (ROIs) from the input image, while five methods –Hu moment matching, histogram based matching, Histogram of Gradients based matching, Euclidean distance based matching and template matching are used for recognizing the traffic sign in the ROI. A classification system based on the shape of the sign is adopted. The performance of the various recognition methods is evaluated by comparing the number of clock cycles used to run the algorithm on the Texas Instruments TMS320C6748 processor. The use of multiple methods for recognizing the traffic signs allows for customization based on the performance of the methods for different datasets. The experiments show that the developed system is robust and well-suited for real-time applications and achieved recognition and classification accuracies of up to 90% [23].

3.3 Same motivation concerning smart cameras

The SmartCam is a low-power, high-performance embedded vision system mainly consisting of a set of individual components [24] [25]. The prototype is based on an Intel IXDP425 development board equipped with a XScale network processor running at 533 MHz. The board features 256M bytes of RAM and four PCI slots, an on-chip ethernet connection and multiple serial ports amongst others. The sensor module is a Kodak Eastman monochrome CMOS camera which delivers images up to VGA resolution at 30 frames per second and is connected via a FIFO ordered memory. For the main processing task, each PCI slot can host an ATEME Network Video Development Kit (NVDK) board which consists of 264MB of memory and TI TMS320C6416 DSPs running at 1 GHz. Communication channels with other smart cameras, external devices or hosts can be established using wired ethernet, IEEE 802.11 wireless LAN or wireless GPS/GPRS radio. The plausibility of the concept was demonstrated on a vehicle detection and tracking application for tunnel safety [25] [26]. However, the focus of current research is on tracking in multi-camera networks [27] , communication, distributed computing and distributed task allocation.

4 Hardware

This chapter offers an overview of the hardware components used in order to compose the proposed imaging platform.

4.1 Camera Module

The camera module used is a CMOS camera board named LI-CAM-M034. The main electronic components that it incorporates are: the MT9M034 1.2MP CMOS digital image sensor by Aptina and a 24MHz oscillator that works as a clock supply to the image sensor. A detailed description of vital part of the camera board that is the image sensor follows.

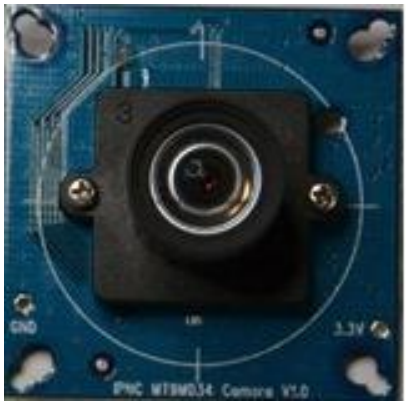


Figure4.1: Front view of the LI-CAM-M034 camera board. On the image sensor a 6mm lens is mounted.

4.1.1 Image Sensor Functional Overview

The MT9M034 is a 1/3 inch CMOS digital image sensor with an active-pixel array of 1280Hx960V. It captures images in either linear or high dynamic range modes with a rolling-shutter readout. It includes sophisticated camera functions such as auto exposure, windowing, both video and single frame modes. It is designed for both low light and high dynamic range scene performance [29]. The following figure depicts the basic structure of the sensor.

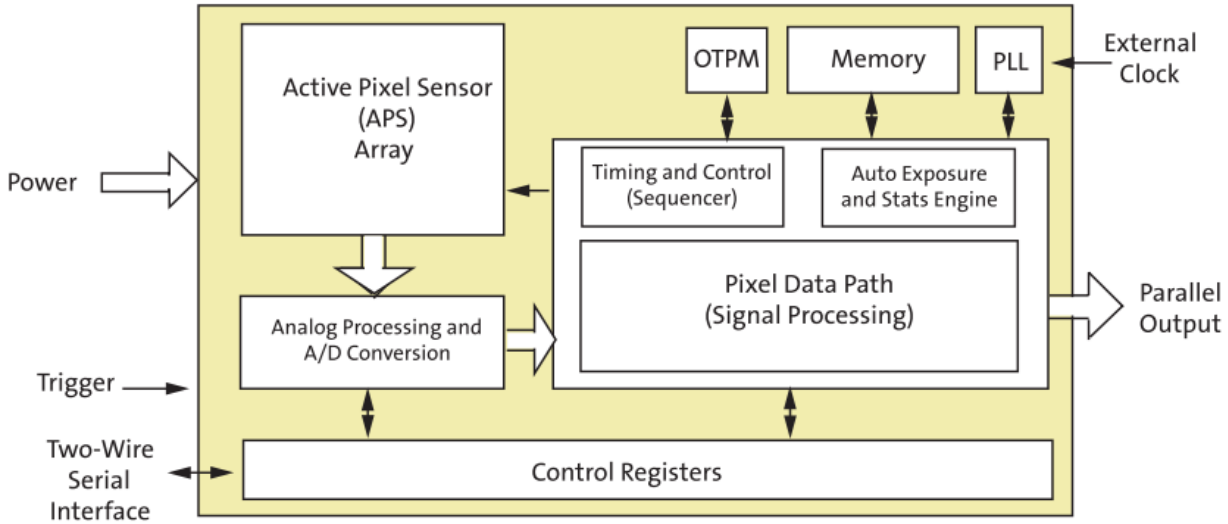


Figure4.2: Block Diagram of MT9M034 CMOS image sensor.

The core of the sensor is the pixel array that contains a 1.2 Mpixel active-pixel array accompanied with numerous light-shielded pixels, which are used to provide data for on-chip black level control algorithms. The active pixel array uses a Bayer color pattern, as shown in Figure4.3.Even-numbered rows contain green and red pixels, odd-numbered rows contain blue and green pixels. Even-numbered columns contain green and blue pixels; odd-numbered columns contain red and green pixels. Continuing with the description of Figure4.2, the timing and control circuitry goes through the rows of the array, resetting and then reading each row in turn. In the time interval between reset and read out of a row, the pixels in each row integrate incident light. Once a row has been read, the data produced from the columns is sequenced through an analog signal chain (providing offset correction and gain) and then through an analog-to-digital converter (ADC). The output from the ADC is a 12 bit-value for each pixel in the array. The last stage before the sensor delivers the actual 12 bit RAW data is a digital processing signal chain, which provides further data path corrections and applies digital gain.

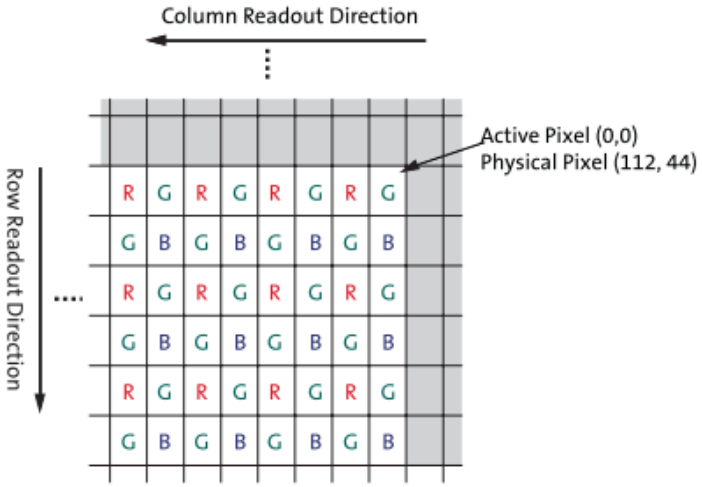


Figure4.3: Color filter pattern (Bayern pattern) of sensor's active pixel array.

The sensor contains a set of control and status registers that can be used to control many parameters of the sensor behavior, including the frame size, exposure and gain settings. These registers can be accessed through the two-wire serial bus, which communicates with the array control, analog signal chain and digital signal chain.
Finally, the sensor chip incorporates a phase-locked loop that can be optionally enabled to generate all internal clocks from a single master input clock running between 6 and 50 MHz. The maximum output pixel rate is 74.25 Mpixel/s, corresponding to a clock rate of 74.25 MHz.
Having referred to sensor's functional units, a coarse description of sensor's functionality follows. When the sensor is imaging, the active surface of the sensor faces the scene as depicted in the following figure.

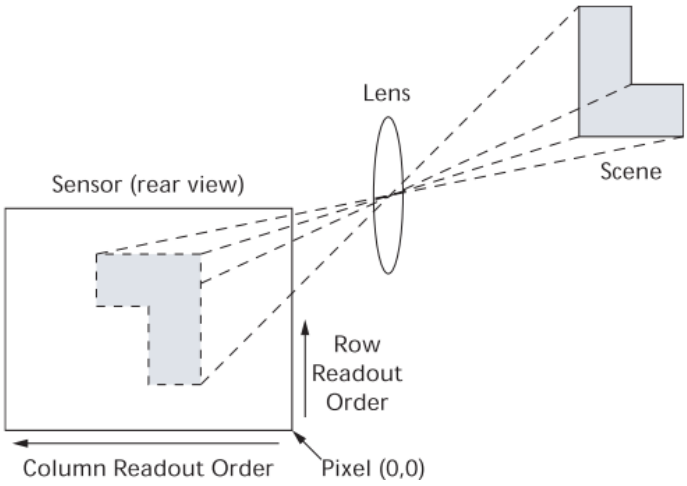


Figure4.4: Imaging a scene

Each scene is exposed using the electronic rolling shutter (ERS), meaning timing and logic control sequences through the rows of the sensor array resetting and then reading each row in turn .The time between row reset and read out is predefined for all rows and during it, the pixels of each row integrate incident light. Consequently, we can understand that row exposure can be controlled by varying the above-mentioned time interval.
The output of the sensor core is a 12-bit parallel pixel data stream qualified by an output data clock (PIXCLK), along with LINE_VALID (LV) and FRAME_VALID (FV) signals .Output data is read out in a progressive scan fashion (just as the way it is integrated) and it contains valid image data surrounded by horizontal and vertical blanking data. Valid image data follows the Bayer pattern; alternate rows are a sequence of either green and red pixels or blue and green pixels. Horizontal and vertical blanking data are zero values.
Macroscopically, the scene is integrated on to the sensor's surface and read out pixel by pixel row-wise starting from sensor's active array top right corner pixel.

Having a basic understanding on how an image is integrated, it would be useful to gain knowledge concerning how the acquired data are outputted to an external receiver.

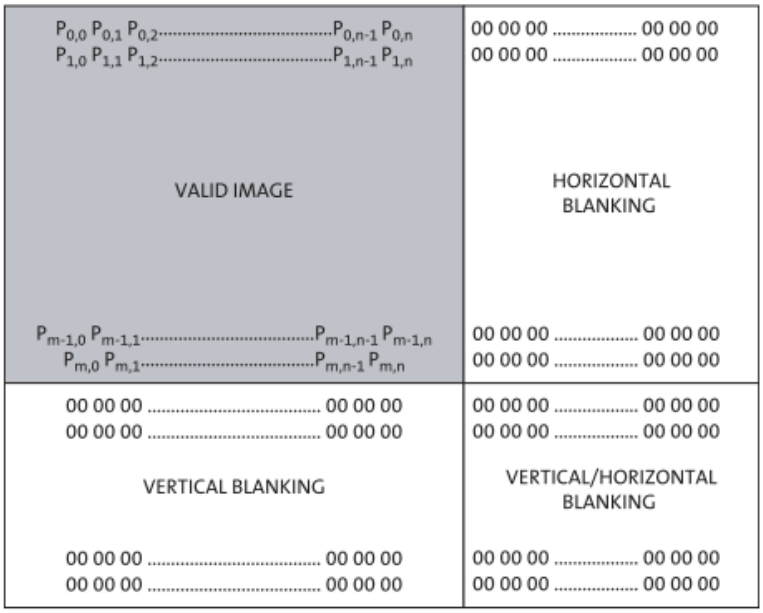


Figure4.5: Spatial representation of image readout

The sensor's output stream is divided into frames, which are further divided into lines, which in turn are divided into pixels. The clocking signals that indicate the boundaries between frames, lines and pixels are FV, LV and PIXCLK respectively. For each PIXCLK cycle, with respect to the falling edge, one 12-bit pixel datum is available to the Dout pins. The FV and LV signals are constructed with respect to the PIXCLK signal and determine the validity of produced pixels. More precisely, when both FV and LV are asserted, the pixel is considered valid data.

PIXCLK cycles that occur when FV is de-asserted are called vertical blanking; PIXCLK cycles that occur when LV is de-asserted are called horizontal blanking. As one can image, the timing of the FV and LV signals is closely related to the row time and frame time respectively. FV will be asserted for an integral number of row times, which will normally be equal to the height of the output image.LV will be asserted during the valid pixels of each row. Usually, LV will be asserted if FV is asserted, although this is configurable.

Apart from its main functionality, the image sensor includes numerous sophisticated functions, some of which are [29]:

- **Readout- Related**
    - **High Dynamic Range Mode (HiDY):** The sensor can optionally operate in HiDY mode, in which it is able to handle 120dB of dynamic range. The sensor sequentially captures three exposures and combines them to render a 20-bit value per pixel which is then optionally compressed back to a 12- or 14-bit.
    - **Mirroring:** Horizontal and Vertical flip of a frame can be performed during readout (on-chip) by the sensor.
    - **Binning:** Binning is the combination of adjacent pixels' charges. The sensor is able to perform binning in three different schemes: the charge of two vertically-neighboring pixels is combined, the charge of two horizontally-neighboring pixels is combined and the charge of two horizontally- neighboring pixels is combined with the charge of their two vertically-neighboring pixels.
- **Control- Related**
    - **Real Time Context Switching:** The sensor has the ability to switch between two full register sets in real time, which means that the new register values are applied to the immediate next exposure and readout time**.**
    - **Automatic Exposure Control:** The integrated Automatic Exposure Control is responsible for ensuring that optimal settings of exposure and gain are computed and updated every other frame.
- **Artifact- Related**
    - **Black Level Correction:** The automatic black level correction is using measurements acquired by a set of optically black lines in the image sensor, in order to provide appropriate offset correction to the output of the image sensor.
    - **Row-wise Noise Correction:** Row-wise noise correction is performed by calculating an average from a set of optically black pixels at the start of each line and then applying each average to all the active pixels of the line.
    - **Column Correction:** The use column parallel readout architecture results in structured fixed pattern noise. The image sensor incorporates column correction circuitry that measures the different offsets added by each column signal path and removes them from the image before output.

4.2 Processing Module

The processing module belongs to the category of heterogeneous multicore processors and is the OMAP-L138 Low Cost Development Kit (OMAP-L138 LCDK). OMAP-L138 LCDK [30]is a scalable platform that eases and accelerates software and hardware development of everyday applications requiring real-time signal processing and control functionality, including industrial control, medical diagnostics and communications.

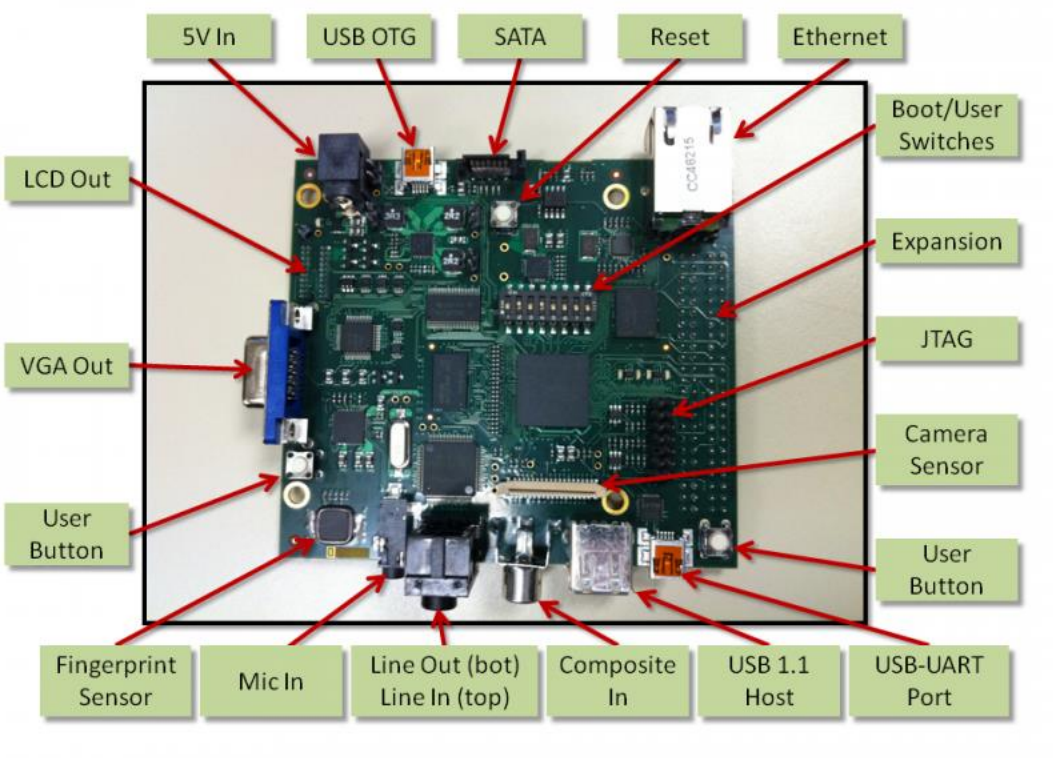


Figure4.6: Physical view of OMAP-L138 LCDK with enumaration of its interfaces.

As it can be inferred by its name, it incorporates the OMAP-L138 processor, a member of the Open Multimedia Applications Platform (OMAP) family. OMAP refers to a series of image/video processors developed by Texas Instruments. These devices generally include a general-purpose ARM architecture processor core plus one or more specialized co-processors. It is worth mentioning that the OMAP L-1x parts have a different technological heritage than the other OMAP parts. Rather than deriving directly from cell phone product lines, they grew from the video-oriented DaVinci product line by removing the video-specific features while using upgraded DaVinci peripherals [31].

As Figure 4.7 depicts the OMAP-L138 Application processor consists of the following primary components:

- ARM subsystem and associated memories (ARM926EJ-S)
- DSP subsystem and associated memories (TMS320C6748 DSP Megamodule)
- System Control
- a set of Input/ Output peripherals
- a DMA subsystem and SDRAM External Memory Interface (EMIF)
- Switch Central Resource
- Join Test Action Group (JTAG) interface

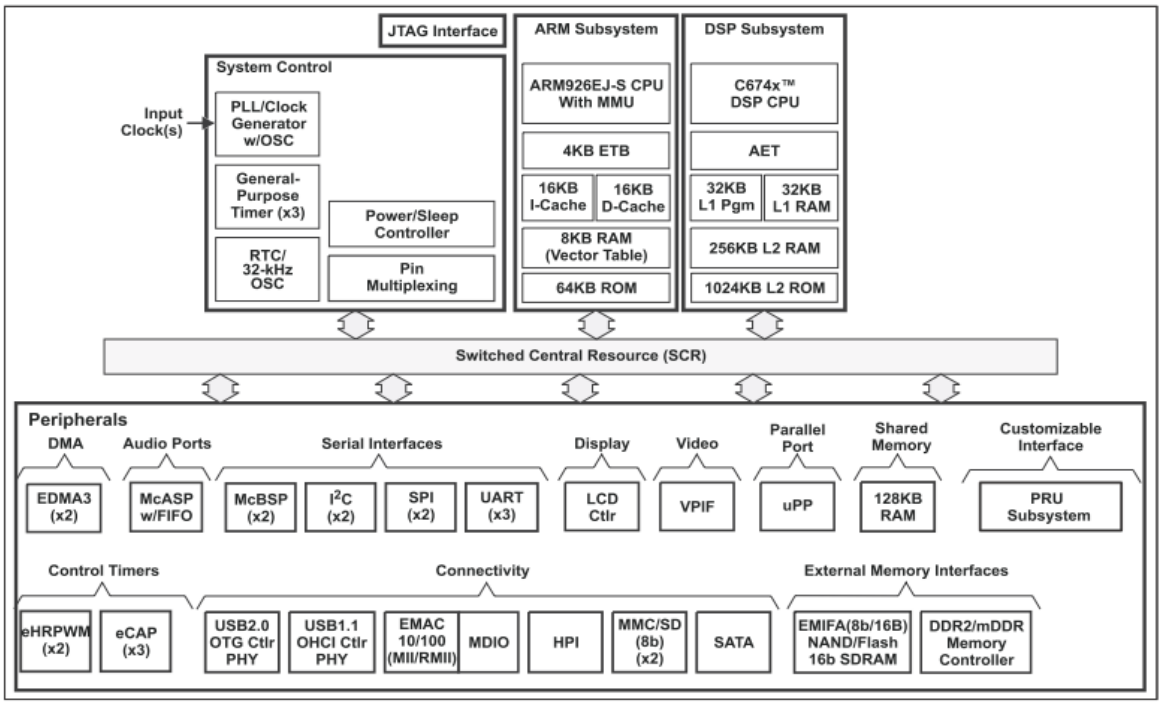


Figure4.7: OMAP-L138 Block diagram

4.2.1 ARM Subsystem

The ARM subsystem constitutes the master core of the OMAP-L138 SoC, this means that by default the device is booted and initialize by the ARM processor which last task of initialization is to "wake-up " the slave core of the OMAP-L138 SoC ,namely the TMS320C674x™ DSP. The ARM subsystem employs ARM926EJ-S processor, a member of the ARM9 family of general-purpose microprocessors. The ARM926EJ-S is suitable for multi-tasking applications where full memory management, high performance, low die size, and low power are important. It supports both the 32-bit ARM and 16-bit Thumb instruction sets, enabling you to trade-off between high performance and high code density. It also includes features for efficient execution of Java byte codes, providing Java performance similar to code derived from Just in Time (JIT) compilation, but without the associated code overhead [32]. As it is illustrated by Figure4.7, ARM926EJ-S has a Harvard cached architecture providing a complete high-performance processor subsystem that includes:

- an ARM9EJ-S integer core
- a Memory Management Unit (MMU)
- separate instruction and data AMBA AHB bus interfaces
- separate instruction and data Thirdly Coupled Memories (TCM) interfaces.

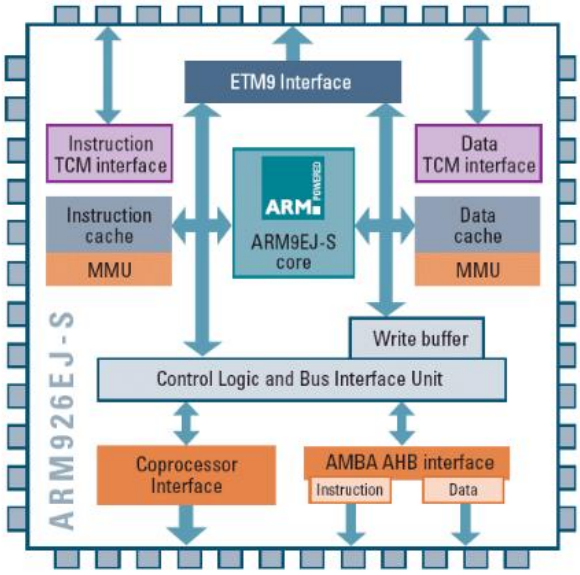


Figure4.8: ARM926EJ-S Block Diagram

4.2.2 DSP Subsystem

The C674x™ Megamodule is OMAP-L138's DSP subsystem .The term megamodule refers to the module's CPU together with hardware providing memory, bandwidth management, interrupt, memory protection and power-down support. The megamodule consists of the following components [33]:

- TMS320C674x™ DSP
- Level 1 program (L1P) memory controller
- Level 1 data (L1D) memory controller
- Level 2 (L2) memory controller
- Internal DMA (IDMA)
- Bandwidth Management (BWM)
- Interrupt Controller (INTC)
- Power-down Controller (PDC)
- Extended Memory Controller (EMC)

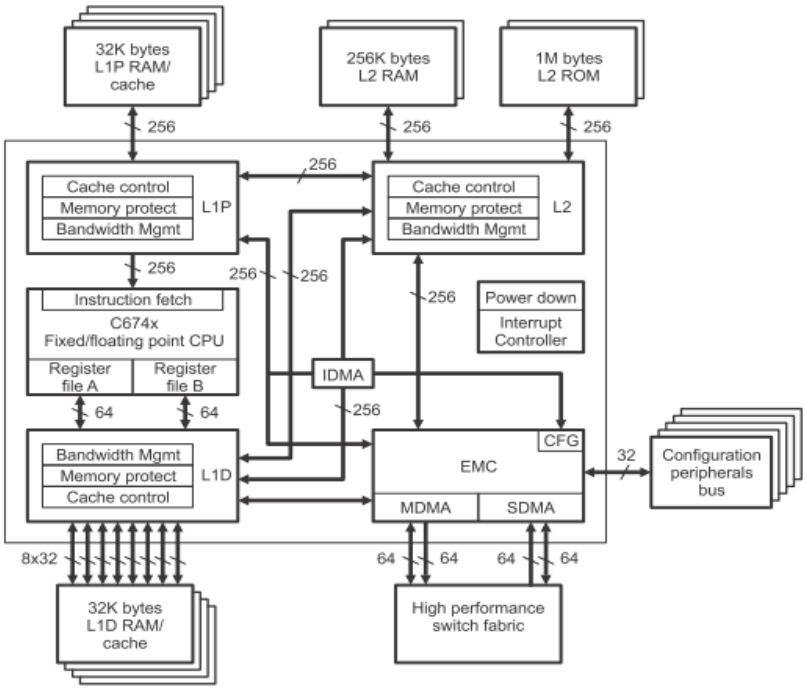


Figure4.9: C674x Megamodule block diagram

The module of C674x™ Megamodule worth commenting from the pre-mentioned list is the processing core, namely the TMS320C674x™ DSP. The TMS320C674x™ DSP is the new generation floating-point DSP that combines the TMS320C67x+™ DSP and the TMS320C64x+™ DSP instruction set architectures into one core [34]. As every member of C6000 devices, it consists of sixty-four general-purpose 32-bit registers and eight functional units (two multipliers and six arithmetic units).It also constitutes an advance VLIW CPU capable of executing up to eight instructions per cycle. Finishing the reference to C674x™ Megamodule, it is also worth mentioning that it employs a two-level cache architecture.

4.2.3 Control System

The processor's Control System major components are the **System Configuration Module** (SYSCFG) and the **Power and Sleep Controllers** (PSC).

The **System Configuration Module** is a **system-level module** containing status and top level control logic required by the device. The system configuration module consists of a set of memory-mapped status and control registers, accessible by the CPU, supporting all of the following system features and functions as well as peripheral-specific operations [35].

- **Device Identification**
- **Device Configuration**
  - Pin Multiplexing Control
  - Device Boot Configuration Status
- **Master Priority Control**
- Controls the system priority for all master peripherals
- **Emulation Control**
  - Emulation suspend control for peripherals that support the feature
- **Special Peripheral Status and Control**
  - Locking of PLL control settings
  - Default burst size configuration for EDMA3 transfer controllers
  - Event source selection for the eCAP peripheral input capture
  - McASP0 AMUTEIN selection and clearing of AMUTE
  - USB PHY Control
  - Clock source selection for EMIFA and DDR2/mDDR
  - HPI Control
- **ARM-DSP Integration**
  - On-chip inter-processor interrupts and status for signaling between ARM and DSP

From the above list, it is worth giving some more input about the following features/functions:

- **ARM-DSP Integration:** The SYSCFG module has a set of registers to facilitate inter-processor communication between ARM and DSP. This is the so called **ARM-DSP Integration feature** and it is generally used to allow the ARM and the DSP to coordinate. For example, the ARM may interrupt the DSP when it is ready to have the DSP process some data buffer in shared memory. Either of the processors can set specific bits in this SYSCFG register, which in turn can interrupt the other processor, if of course the interrupts have been appropriately enabled in the processor's interrupt controller
- **Master Priority Control:** The on-chip peripherals/modules are essentially divided into two broad categories, **masters** and **slaves**. The master peripherals are typically capable of initiating their own read/write data access requests. The master peripherals category includes the ARM, DSP, EDMA3 transfer controllers, and peripherals that do not rely on the CPU or the EDMA3 for initiating the data transfer to/from them. The remaining peripherals are categorized as slave peripherals. Each Switched Central Resource (SCR) interconnection performs prioritization based on priority level of the master that sends the read/write requests. For every peripheral classified as master on the device, there is a priority assigned to it. The priority levels range from zero to seven, with zero corresponding to highest priority. There are default priority levels for every peripheral and any application software is expected to modify these values to obtain the desired performance.
- **Pin Multiplexing Control:** Extensive use of pin multiplexing is used to accommodate the large number of peripheral functions in the smallest possible package. On the device, pin multiplexing can be controlled on a pin by pin basis, using the pin multiplexing registers (PINMUX0-PINMUX19). Each pin that is multiplexed with several different functions has a corresponding 4-bit field in PINMUXn. Pin multiplexing selects which of several peripheral pin functions control the pins I/O buffer output data. Note that the input from each pin is always routed to all of the peripherals that share the pin; the PINMUX registers have no effect on input from a pin.

The **Power and Sleep Controllers** (PSC) are responsible for managing transitions of system power on/off, clock on/off, resets (device level and module level). It is used primarily to provide granular power control for on chip modules (peripherals and CPU). A PSC module consists of a Global PSC (GPSC) and a set of Local PSCs (LPSCs).The GPSC contains memory mapped registers, PSC interrupts, a state machine for each peripheral/module it controls. An LPSC is associated with every module that is controlled by the PSC and provides clock and reset control. Many of the operations of the PSC are transparent to user (software), such as power on and reset control. However, the PSC module(s) also provide you with interface to control several important power, clock and reset operations [35].

4.2.4 DMA subsystem and SDRAM External Memory Interface

The processor's **DMA subsystem** is addressed by the name **Enhanced Direct Memory Access** (EDMA3) controller. Its primary purpose is to service user-programmed data transfers between two memory-mapped slave endpoints on the device. Typical usage includes, but is not limited to:

- servicing software driven paging transfers (for example, from external memory to internal device memory.
- servicing event driven peripherals, such as a serial port
- performing sorting or subframe extraction of various data structures
- offloading data transfers from the main device CPU(s) or DSP(s)

This device has **two** external memory interfaces that provide multiple external memory options accessible by the CPU and master peripherals. The type of memory

- **External Memory Interface (EMIF):**
  - Used to interface "slower" memories
  - 8/16-bit wide asynchronous EMIF module that supports asynchronous devices such as ASRAM,NAND Flash, and NOR Flash
  - 16-bit wide NAND Flash with 4-bit ECC
- **DDR2/mDDR memory controller:**
  - Used to interface "faster" memories
  - 16-bit DDR2 with 128-MB memory address space

4.2.5 Input/ Output Peripherals

Being rich in peripherals, the OMAP-L138 LCDK offers a wide variety of standard interfaces along with expansion headers is available to the end user. The peripherals used in the scope of this thesis and will be discussed are:

- Video Port Interface (VPIF) peripheral
- Liquid Crystal Display Controller (LCDC)
- Inter-Integrated Circuit (I^2C)

VPIF Peripheral

Video Port Interface is the peripheral responsible for video input and output. The VPIF incorporates two receive (CHANNEL0 and CHANNEL1) and two transmit (CHANNEL2 and CHANNEL3) video channels [35].

The VPIF is designed to support the following video formats:

- **ITU-BT.656**
- **ITU-BT.1120**
- **SMTPE 296**
- **RAW data capture**

The above-mentioned video formats can be supported in either interlaced or progressive mode. In progressive mode, all the lines of an incoming frame are captured /transmitted in sequence. This is in contrast to interlaced mode, where only the odd lines, or only the even lines of each incoming frame are captured /transmitted alternately, so that only half the number of an actual image frames are used to produce video.

The VPIF peripheral interact with the processor via five interrupt signals, which are asserted when:

- **FRAME0**: Frame sync or line interval[1] on CHANNEL0 detected.
- **FRAME1:** Frame sync or line interval[1] on CHANNEL1 detected.
- **FRAME2**: Frame sync on CHANNEL2 detected.
- **FRAME3:** Frame sync on CHANNEL3 detected.
- **ERROR:** Error detected. An error event can come up due to internal buffer overflow, or error in the length of synchronization codes.

[1] A line interval can be detected only when RAW capture mode is enabled.

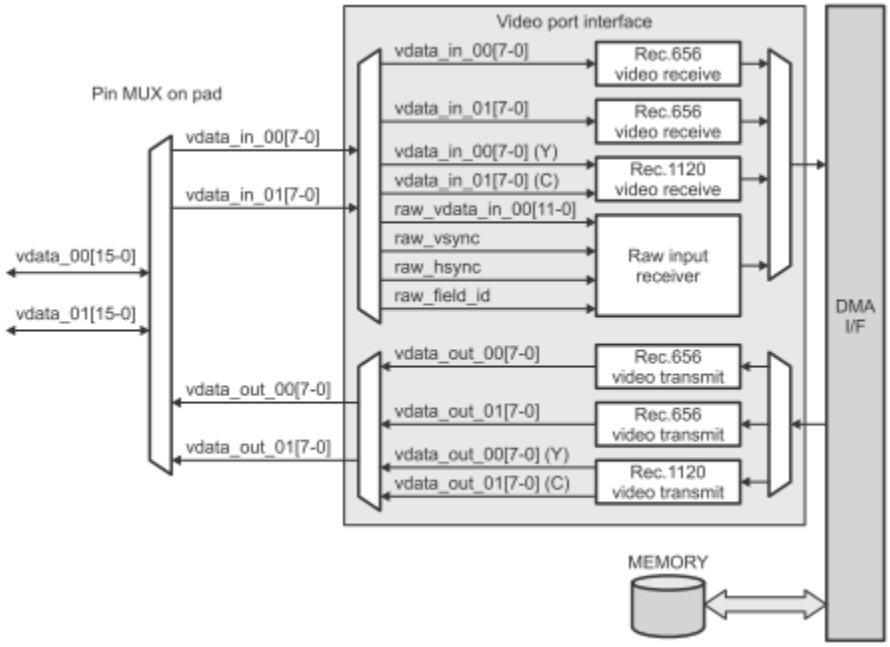


Figure4.10: VPIF peripheral Block Diagram

VPIF **receives** incoming data either through only CHANNEL0 or through CHANNEL0 and CHANNEL1 depending on the incoming data's video format. The acquired data are fed to its internal DMA subsystem, which in turns delivers them to system's memory in bursts. In proportion to the above-mentioned, VPIF **transmits** data either through only CHANNEL2 or through CHANNEL2 and CHANNEL3 depending on the desired output video format.

LCDC controller

Liquid Crystal Display Controller is the peripheral responsible for driving both asynchronous and synchronous LCD interfaces, in order to achieve this it consist of two independent controllers, namely [35]:

- **LCD Interface Display Driver (LIDD):** The LIDD Controller supports the asynchronous LCD interface, providing full-timing programmability of control signals and output data.
- **Raster Controller:** The Raster Controller handles the synchronous LCD interface. It provides timing and data for constant graphics refresh to a display. It supports a wide variety of monochrome and full-color display types and sizes by use of programmable timing controls, a built-in palette, and a gray-scale/serializer.

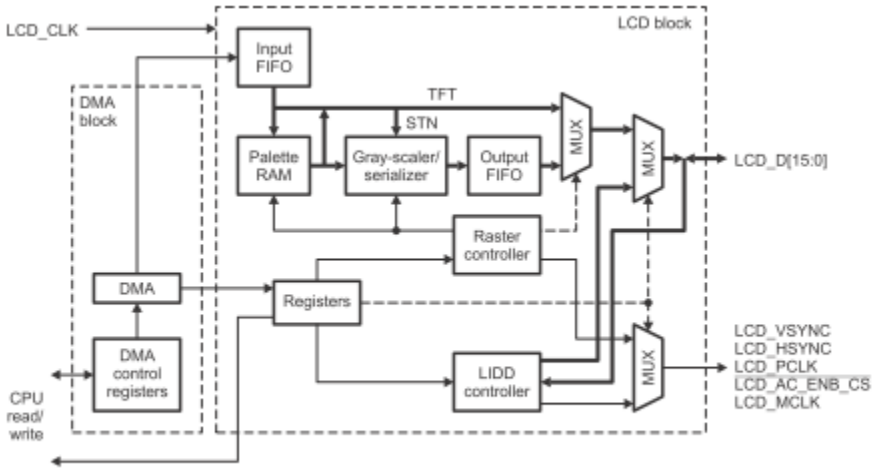


Figure4.11: LCDC Controller Block Diagram

The data to be displayed is stored in frame buffers (contiguous memory blocks in the system). A built-in DMA engine supplies the graphics data from the frame buffers to the Raster and LIDD controllers which, in turn, outputs it to the external LCD device. Apart from graphics data, Raster and LIDD controllers are responsible for generating the appropriate external timing.

The LCDC peripheral is capable of generating event to the processor via seven interrupt signals, which are asserted when:

- **EOF1:** End of frame 1 detected.
- **EOF0**: End of frame 0 detected.
- **PL**: Palette is loaded into memory.
- **FUF**: LCD dither logic does not supplying data to FIFO at a sufficient rate, FIFO has completely emptied and data pin driver logic has attempted to take added data from FIFO.
- **ABC**: When AC-bias transition counter has decremented to zero, indicating that the LCD_AC_O line has transitioned the number of times that is specified by the ACB_I control bit field.
- **SYNC**: Frame synchronization lost has occurred.
- **DONE**: After the last set of pixels is of a frame has been clocked to the LCD and Raster or DMA_to_LIDD has been disabled.

I²C Peripheral

I²C (Inter-Integrated Circuit) is a multimaster **serial** single-ended **bus** used for attaching low-speed peripherals to an electronic device such as a motherboard, an embedded system or a smartphone. In our case the electronic device is the processor or the EDMA3 controller [35].

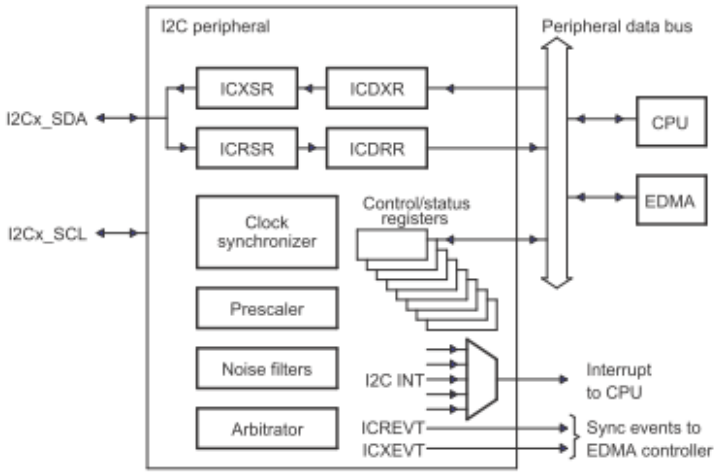


Figure4.12: I²C peripheral Block Diagram

The major components of the I²C peripheral are (see Figure4.12):

- a **serial interface** made up by one data pin (I2Cx_SDA) and one clock pin (I2Cx_SCL)
- **data registers** (ICXSR, ICRSR ,ICDXR and ICDRR) to temporarily hold receive data and transmit data traveling between the I2Cx_SDA pin
- a **data bus** and **interrupt logic** enabling communication between the I²C peripheral and the processor or the EDMA controller.
- a set of **control** and **status registers**
- a **clock synchronizer** to synchronize the I2C input clock (from the processor clock generator) and the clock on the I2Cx_SCL pin and to synchronize data transfers with masters of different clock speeds

The I²C peripheral has four basic operating modes to support data transfers as a master and as a slave, namely:

- Master-transmitter mode
- Master-receiver mode
- Slave-transmitter mode
- Slave-receiver mode

If the I^2C peripheral is a **master**, it begins as a master-transmitter and, typically, transmits an address for a particular slave. When giving data to the slave, the I^2C peripheral must remain a master-transmitter. In order to receive data from a slave, the I^2C peripheral must be changed to the master-receiver mode.
If the I^2C peripheral is a **slave**, it begins as a slave-receiver and, typically, sends acknowledgment when it recognizes its slave address from a master. If the master will be sending data to the I^2C peripheral, the peripheral must remain a slave-receiver. If the master has requested data from the I^2C peripheral, the peripheral must be changed to the slave-transmitter mode.

The I^2C peripheral supports 1-bit to 8-bit data values. The data is transferred serially with the most-significant bit (MSB) first. The number of data values that can be transmitted or received is unrestricted; however, the transmitters and receivers must agree on the number of data values being transferred. Although I^2C can transfer data values which length ranges from 1-bit to 8-bit, it can support slave addresses which length is not restricted by the 8-bit upper bound. The supported addressing modes are:

- **7-bit addressing mode:** a slave address' length is 7 bits, while data values' length ranges from 1-bit to 8-bits
- **10-bit addressing mode:** a slave address' length is 10 bits and it is transmitted as two 8-bit data values, again data values' length ranges from 1-bit to 8-bits
- **free data format mode:** both slave address' and data values' length ranges from 1-bit to 8-bits

The I^2C peripheral is capable of generating event to the processor via the following interrupt signals:

- **Arbitration-lost interrupt (AL):** Generated when the I^2C arbitration procedure is lost or illegal START/STOP conditions occur
- **No-acknowledge interrupt (NACK):** Generated when the master I^2C does not receive any acknowledge from the receiver
- **Registers-ready-for-access interrupt** (ARDY)**:** Generated by the I^2C when the previously programmed address, data and command have been performed and the status bits have been updated. This interrupt is used to let the controlling processor know that the I^2C registers are ready to be accessed.
- **Receive interrupt/status:** Generated when the received data in the receive-shift register (ICRSR) has been copied into (ICRINT and ICRRDY) the ICDRR.
- **Transmit interrupt/status:** Generated when the transmitted data has been copied from ICDXR to the transmit-shift (ICXINT and ICXRDY) register (ICXSR) and shifted out on the I2Cx_SDA pin.
- **Stop-Condition-Detection interrupt** (SCD)**:** Generated when a STOP condition has been detected
- **Address-as-Slave interrupt (AAS)**: Generated when the I^2C has recognized its own slave address or an address of all zeros.

4.2.6 Switch Central Resource

The DSP, the ARM, the Programmable Real-Time Unit (PRU) subsystem, the EDMA3 transfer controllers, and the device peripherals are interconnected through a **switch fabric architecture**. The switch fabric is composed of multiple Switched Central Resources (SCRs) and multiple bridges. The SCRs establish low-latency connectivity between master peripherals and slave peripherals. Additionally, the SCRs provide address decoding, priority-based arbitration and routing, attributes that enables it to facilitate efficient concurrent data movement between master and slave peripherals [36].

4.2.7 JTAG interface

JTAG is the common name for the **Standard Test Access Port and Boundary-Scan Architecture**. JTAG is used as the primary means of accessing sub-blocks of integrated circuits, making it an essential mechanism for debugging embedded systems which may not have any other debug-capable communications channel. An in-circuit **emulator** (or, more correctly, a "JTAG adapter") uses JTAG as the transport mechanism to access (through a small number of dedicated pins) on-chip debug modules inside the target CPU. Those modules let software developers debug the software of an embedded system directly, through mechanism such as single-stepping, and breakpointing [37].

The main two components of the platform were described with respect to its structure and its functionality. So, the hardware platform is set up waiting for the software to enliven it. A picture of the hardware platform (OMAP-L138 LCDK + LI-CAM-M034) connected via a debugger (XDS100v2 USB JTAG emulator) to a host laptop and via VGA to a 17’ monitor follows (Figure4.13):

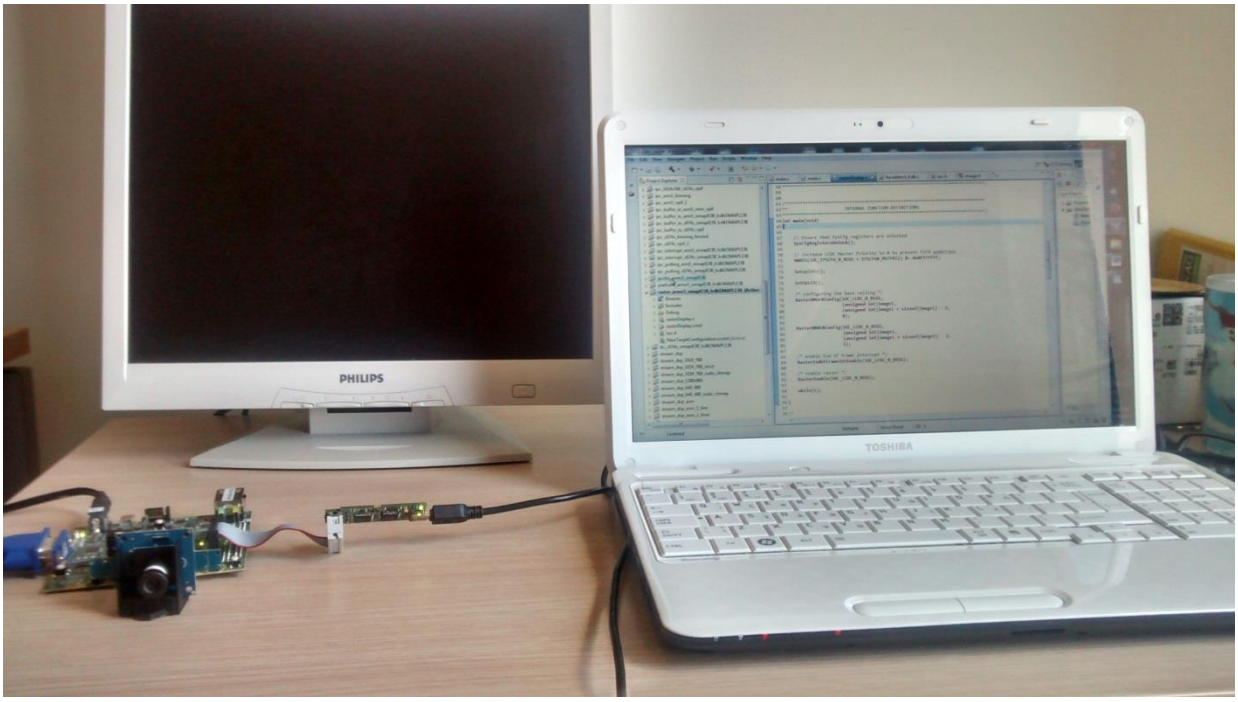


Figure4.23: The developed hardware platform connected to a host PC via a debugger and to a monitor.

5 Software

5.1 Software Infrastructure Options

The first step is to determine the base software layer that will coat the hardware and serve as an infrastructure to the applications that will be developed.

There are several options for the base software layer [40]:

- **Real Time Operating System (RTOS)**: The fact that differentiates a generic operating system from a real time one is that the later is designed to serve real-time application requests. As a result, a RTOS is characterized by predictable time response and deterministic behavior. These properties are attributed to features such as sophisticated scheduling, minimal interrupt latency and minimal thread switching latency. Developing on a RTOS enables you to conveniently design a system with deterministic behavior; not that without it you are not able to do so.
- **Generic Operating System (OS)**: An operating system is software that manages computer hardware resources and provides common services for computer programs. Developing on a generic OS can jump start your development by providing a predefined code structure (and typically existent peripheral drivers) which will allow you to focus directly on your end application development. While this may sound advantageous, it is important to understand that the benefits that one gains with a traditional operating system can quickly be out-shadowed by non-optimized code that can quickly consume a lot of unnecessary processing power and/or available on-chip system memory resulting in a higher overall system cost.

- **No operating System:** Software development on a device without the use of any operating system is typically done when a very minimal code footprint is needed, or when an operating system may add unnecessary complexity that is not needed. Development with no operating systems is typically discouraged because it requires an application to rely solely on interrupts for scheduling. While this approach does work, it quickly becomes a heavy burden on the developer: as the application scales to meet additional demands difficulty in composing code to manage all events scales analogously.

Specifically, for OMAP-L138/C6748 products TI offers the following software development packages:

- **Real Time Operating System**
  - DSP/BIOS Platform Support Package for OMAP-L138/C6748
- **Generic Operating System**
  - Linux Software Development Kit for OMAP-L138
  - WindowsCE(WinCE) Software Development Kit for OMAP-L138
  - QNX Software Development Kit for OMAP-L138
  - Mentor Embedded Nucleus RTOS Development Kit
- **No Operating System**
  - QuickStartOMAPL1x rCSL Register Level Chip Support Library Example Package
  - StarterWare Device Abstraction Layer
  - EVM BSL Board Support Library
- **Optimized DSP Software libraries**
  - DSP Software Libraries

The most appealing choices were to either use a Real Time Operating System (DSP/BIOS Platform Support Package) or no operating system at all (StarterWare development package). We ended up with the second choice. The main reason for going with the StarterWare instead DSP/BIOS is that StarterWare not being an OS is more controllable and neat, fact that was very important since the project started from scratch. Based on StarterWare an event-driven main control loop [41] was developed, which enabled us to enjoy RTOS-like features while having the luxury not to worry about protecting data with semaphores/mutexes. By the term RTOS-like features we refer to abstracting out algorithmic details and creating standard programming interfaces (task-like structure) and to the fact that functions (tasks) can be event-driven (i.e. triggered by hardware interrupts) and can exchange messages (e.g. inter-processor communication).

It is worth mentioning that both StarterWare and DSP/BIOS are equally maintained for OMAP-L138, so these software packages offered the same driver support libraries and that both being processor/platform specific they suffer the same portability issues.

5.2 StarterWare Software Development Package

StarterWare is a free software development package that provides no-OS platform support for OMAPL138. StarterWare includes Device Abstraction Layer (DAL) libraries and example applications that demonstrate the capabilities of the peripherals OMAPL138 on both the ARM and DSP cores [42]. This package includes the following components (see Figure5.1):

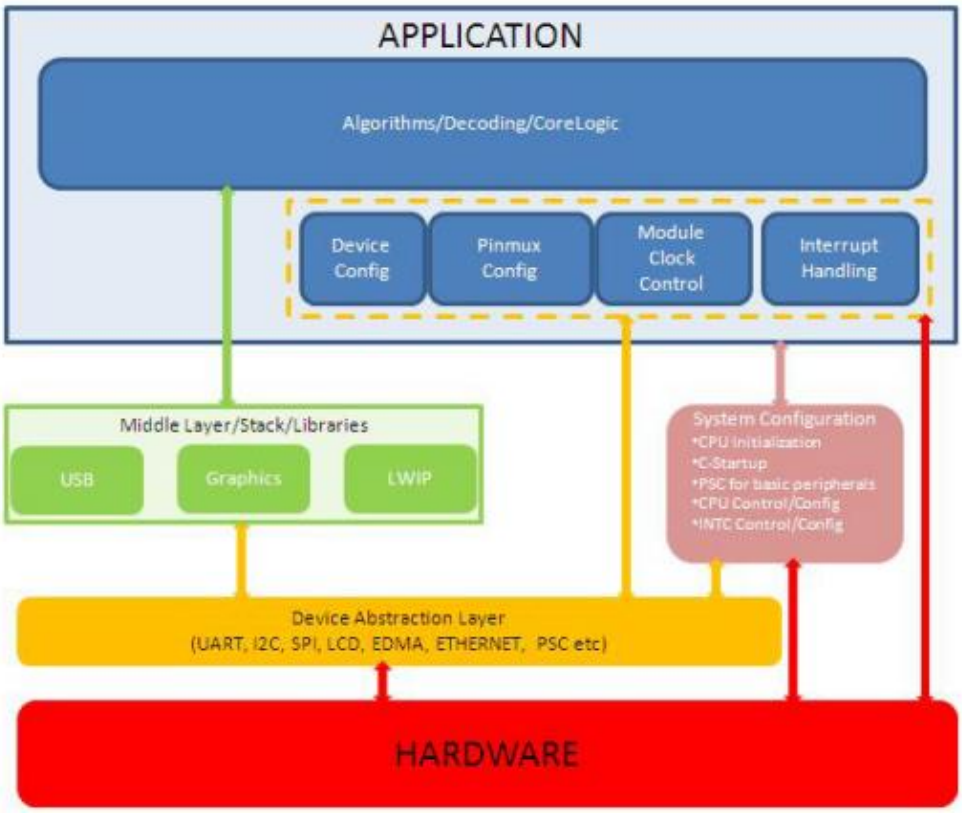


Figure5.3: The StarterWare development package

- **Device Abstraction Layer Library:** Device abstraction layer (i.e. drivers) for supported peripherals.
- **Example Applications:** Sample applications which demonstrate the use of peripheral drivers and other libraries.
- **System Configuration Library:** Fundamental APIs that enable interrupt and cache usage.

- **Platform Library:** Platform-specific initialization code. This code sets board-level features like pin multiplexing, IO expanders, GPIOs and other similar features that are commonly required by applications.
- **Graphics Library:** Lightweight 2D graphics library that renders primitives (lines, circle, etc.), fonts, and user interface widgets.
- **USB Stack Library:** Implements host and device support for common USB classes.
- **IPC Lite:** Lightweight inter-processor communication (IPC) module with very small code size and low latency.
- **lwIP:** Lightweight, open source network stack for use with EMAC.
- **FatFs:** Lightweight, open source file system for use with external memory devices.

At this point, we have a hardware platform accompanied with a basic software infrastructure on which we can build any application we wish.

5.3 Video Streaming Breakdown

Using and modifying the example application offered by StarterWare in which frames from a camera can be displayed to a LCD device, we built a simple video streaming application, which will later serve as the base of the applications that we intend to develop.
Conceptually, video streaming consists of two main parts: **video capture** and **video display**.

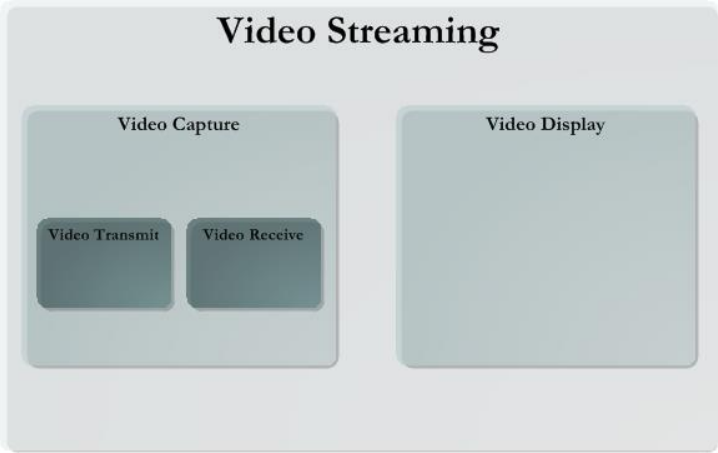


Figure5.4: The Video Streaming breakdown

5.3.1 Video Capture

By the term video **capture**, it is implied that video is transmitted and received. In this case, video is transmitted by the image sensor and received by the OMAP-L138 processor. Both the video **transmitter** and the **video receiver** should be configured analogously, so the following part will be devoted to the description of their configuration.

The video transmitter

As it had been stated in the imaging parameters of the sensor can be controlled via a set of registers accessed by an I^2C bus. When two devices are connected via an I^2C bus, their communication is based on a **master-slave** model. Since the image sensor is the device to be controlled we can understand that it constitutes the slave. The device responsible for controlling the image sensor is the processor and it will do it via the I^2C peripheral, which makes the I^2C peripheral the master.
The communication between master and slave can be accomplished via a clock and a data signal. More particularly, the master generates a clock (**SCLK**) that is an input to the sensor and is used to synchronize transfers. Data is transferred between the master and the slave on a bidirectional signal named **SDATA**. SDATA is by default pulled up by a resistor. Either the slave or master device can drive SDATA LOW; the interface protocol determines which device is allowed to drive SDATA at any given time.

In order to use the I^2C peripheral as a master, a set of initialization steps should take place:

- appropriate Pin Multiplexing for I^2C
- put I^2C in reset
- set I^2C to operate as a master
- set the speed of clock on SCLK pin
- let the master know the slave address
- set up interrupts from I^2C to processor
- enable the I^2C bus

The following table describes the key configuration parameters for configuring the I^2C as a master:

Register	Register Field	Value
ICMDR	No-acknowledge (NACK) mode bit (only applicable when the I2C is a receiver)	The I2C (as a receiver) sends a NACK bit to the transmitter during the next acknowledge cycle on the bus
	START condition bit (only applicable when the I2C is a master)	STT is set to 1 causing the I2C (in master mode) to generate a START condition on the I2C-bus.
	STOP condition bit (only applicable when the I2C is a master).	STP has been set to generate a STOP condition when the internal data counter of the I2C counts down to 0.
	Master mode bit	Master mode
	Transmitter mode bit	Transmitter mode
	Expanded address enable bit	7-bit addressing mode (normal address mode)

Table1: I²C key register settings

At this point we should mention that the image sensor acts as a slave is by default and nothing needs to be done.

Any configuration in image sensor's functionality is a write access to the appropriate register. This means that the master must tell the slave which register is to be accessed and which value it will be assigned to that register. A register's address is a 16-bit value and also 16-bit values are expected to be assigned to it. But, an I^2C bus can transfer data with length up to 8-bits , as a result, in order to specify one register address or one register value the master will transmit two 8-bit messages.

Data transfers on the two-wire serial interface are performed by a sequence of low-level protocol elements:

- **a (repeated) start condition :** a HIGH to LOW transition on SDATA pin while SCLK is HIGH; a master drives this condition to indicate the start of a data transfer.
- **a slave address/data direction byte:** This byte represents two kind of information; the slave device address (bits[7:1]) and the data transfer direction (bit[0]).The sensor slave address is 0x90 and data transfer direction bit has the valued '0' when a WRITE operation is performed and the value '1' when a READ operation is performed. Consequently, sensor's write address is 0x90 and read address is 0x91.

- **a (no) acknowledge bit:** Each 8-bit data transfer is followed by an acknowledge bit or a no-acknowledge bit in the SCLK clock period following the data transfer.

    - An **acknowledge** bit is generated by the transmitter (which is the master when writing, or the slave when reading) when SDATA is released.
    - An **acknowledge** bit is generated by the receiver by driving SDATA LOW.
    - A **no-acknowledge** bit is generated when the receiver does not drive SDATA LOW during the SCLK clock period following a data transfer. A no-acknowledge bit is used to terminate a read sequence.
- **a message byte:** This byte carries the actual data to be transferred. Data to be transferred can be: a register address, a register value to be written or read.
- **a stop condition:** a LOW to HIGH transition on SDATA pin while SCLK is HIGH; a master drives this condition to indicate the end of a data transfer.

A write operation to a sensor's register (see Figure5.3) begins with the master generating a start condition. Then the master sends the slave address/data direction byte which signals a write operation in the predefined slave .Then, the master sends the higher 8 bits of the register address that is to be written followed by lower 8 bits bytes of that register. Afterwards, the master sends the write data, which in our case are two 8-bit messages representing the value to be written to the register. Throughout, this communication the slave just acknowledges each message. Finally, the write access is terminated by the master generating a stop condition [29].

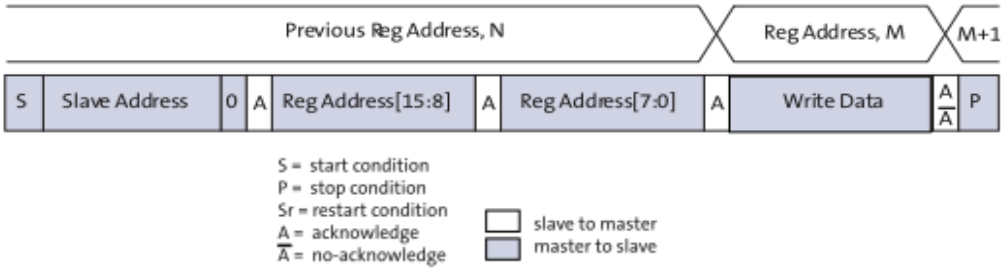


Figure5.5: A write operation; a value is assigned to a random register.

A read operation from a sensor register starts with a dummy write operation to the 16-bit address that is to be read. The master terminates the write operation by generating a restart condition. The master then sends the 8-bit read slave address/data direction byte to signal a read operation and clocks out the read data (two 8-bit messages representing the register's value). The master terminates the read operation by generating a no-acknowledge bit followed by a stop condition [29].

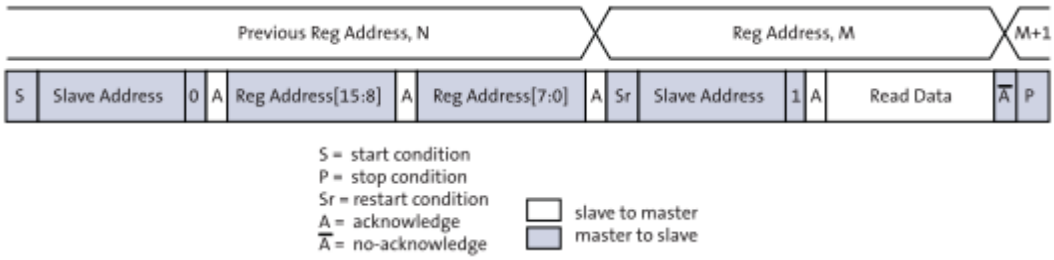


Figure5.6: A read operation; a value is assigned to a random register.

The sensor's configuration which will be done via I^2C write operations entails the following:

Clocking

The image sensor is fed with an external clock that has to be transformed in order to provide the desired master clock frequency in which the sensor will operate. The transformation is done via a prescaler (N), followed by a multiplier (M) which in turns is followed by two dividers (P1, P2). The following equation gives the relationship between master clock frequency (f_{PIXCLK}) and external clock frequency (f_{EXTCLK}).

$$f_{PIXCLK} = \frac{(f_{EXTCLK} * M)}{(N * P1 * P2)}$$

$$where$$

$$M = Multiplier\ and\ \ 32 \leq M \leq 255$$
$$N = Prescaler\ \ and\ \ 1 \leq N \leq 63$$
$$P1 = Divider\_1\ \ and\ \ 1 \leq P1 \leq 16$$
$$P2 = Divider\_2\ \ and\ \ 4 \leq P2 \leq 16$$

An assignment of a desired value to any of the parameters (N, M, P1 and P2) associated with the external clock transformation is a write operation to the appropriate register. In our case, the image sensor's master clock frequency is always set to the maximum value: 74.25 Mp/s.

Sequencer Set up

The timing and control circuitry, also called a sequencer, is essentially a state machine [Mt9d112] providing timing control to the sensor active array. It can also perform write/read operations to a RAM memory when there is not an on-going readout process. A primary sensor initialization step requires a set of given values associated with the linear and high dynamic range operating modes to

be written in the sequencer RAM. This means that repeated write accesses are performed in the sequencer RAM. When these write operations are finished a readout process can be initiated. Write accesses again are performed via proper assignments to proper registers.

Linear mode
As it had been mentioned the image sensor supports two operational modes: linear and high dynamic range. The term linear mode refers to the relationship between the integrated light and the produced charge. In the scope of this thesis, sensor is always set to operate in linear mode. Configuring the sensor in linear mode mainly includes appropriate ADC-DAC settings for improved noise performance and operations related to the column correction algorithm. Again all these are write operations to the image sensor's registers.

Exposure
The sensor's exposure is affected by **integration time** and **gain**, by adapting these two factors one can achieve the appropriate exposure for a scene. The sensor offers both **manual** and **automatic** control of exposure, meaning that integration time and gain can be controlled either manually by the user or automatically by the image sensor's on-chip Automatic Exposure Control (AEC) algorithm.

Integration time is the summation of **coarse integration time** and **fine integration time**:

$$t_{INT} = t_{INT_{coarse}} + t_{INT_{fine}}$$

Coarse integration time is measured in terms of the time interval that a sensor row is integrating light and equals:

$$t_{INT_{coarse}} = number\ of\ frame\ lines * line\ time$$

When AEC is enabled, the number of lines may vary from frame to frame within controlled upper-lower limits. If AEC is disabled, the number of lines equals the value in the corresponding register. Typically, the value of coarse integration time is limited to the number of lines per frame (which includes vertical blanking lines), such that the frame rate is not affected by the integration time.

Fine Integration Time is measured in terms of the time interval that a pixel is integrated and equals:

$$t_{INT_{fine}} = number\ of\ pixels * pixel\ time$$

The number of fine shutter width pixels is independent of AEC mode (enabled or disabled). Maximum value for $t_{INT_{fine}}$ is line length measured in pixel clocks minus 750.

Gain
The sensor's gain can be of three types:

- **Analog Gain:** The sensor has a column parallel architecture and therefore has an analog gain stage per column. There are 2 stages of analog gain; the first stage can be set by the

appropriate register to 1x, 2x, 4x or 8x. The second stage is capable of setting an additional 1x or 1.25x gain by the appropriate. In the end, the maximum possible analog gain to be set to 10x.

- **Digital Gain:** Digital gain can be controlled globally by the corresponding registers. There are also registers that allow individual control over each Bayer color (GreenR, GreenB, Red, Blue). The format for digital gain setting is xxx.yyyyy , where the step size for yyyyy is 0.03125 and the step size for xxx is 1.

$$Digital_{gain} = (xxx * 1) + (yyyyy * 0.03125)$$

- **Dual Conversion Gain (DCG):** In order for higher inter-scene dynamic range to be achieved, each pixel is designed to handle both high and low light conditions. When the sensor is imaging in high-light conditions, the DCG switch is turned on, enabling the pixel to handle more charge via an assistant capacity. In low-light conditions, the DCG is turned off, disconnecting the assistant capacity, resulting in higher sensitivity [43].

To sum up about exposure control: when the sensor operates in manual exposure, it sets the integration time according to the values in coarse and fine integration time registers and the gain according to the values of the gain registers. When AEC is enabled, it measures current scene luminosity by accumulating a histogram of pixel values while reading out a frame. It then compares the current luminosity to the desired output luminosity. Finally, the appropriate adjustments are made to the exposure time and gain. AEC can be enabled or disabled by a proper write access to the corresponding register.

Frame rate

Frame rate is affected by the **frame size** and the **integration time**. As frame size or integration time increases frame rate drops analogously. Generally, it is not desirable to allow the frame rate to be reduced due to long integration times, so we will focus on how the frame rate is adjusted according to the frame size. The equation that represents the relationship between frame size and frame rate follows:

$$frame\ rate = \frac{f_{PIXCLK} * 10^6}{line\ length * frame\ lines}$$

The length of a line and the lines of a frame are defined via the corresponding registers.

Frame size

The size of a frame can be configured via a set of two coordinates; one designating the top left corner (x_addr_start, y_addr_start) and the other the bottom right corner (x_addr_end, y_addr_end).These coordinates are represented by four registers. The following holds:

$$frame\ width = \text{x\_addr\_end} - \text{x\_addr\_start} + 1$$
$$frame\ height = \text{y\_addr\_end} - \text{y\_addr\_start} + 1$$

There are certain limitations concerning the values of the above mentioned coordinates as well as the size of a frame which are summarized in the next table.

Parameter	Constraints
x_addr_start[0]	0
x_addr_end[0]	1
x_output_size	min=4 max=1280
x_output_size[1:0]	must be zero
y_addr_start[0]	0
y_addr_end[0]	1
y_output_size	min=2 max=960
y_output_size[0]	0

Table2: Programming rules concerning frame size

The transmitter or the image sensor is all set up, ready to output frames, so the next step is to configure the receiver.

The video receiver

As it had been stated, the receiver is the OMAP-L138 processor, this is not entirely correct; the image sensor does not communicate directly with the processor instead it communicates with the processor via the peripheral named: Video Port Interface (VPIF).

Since our intention is to interface a progressive scan image sensor with 12-bit RAW output, VPIF should be configured to receive 12-bit RAW data in a progressive fashion. In progressive RAW capture mode, RAW data is captured line by line via the VPIF with respect to the active periods of horizontal and vertical timing signals. In order to use VPIF's peripheral receive channels, a set of initialization steps should take place:

- enable VPIF in the LPSC
- appropriate Pin Multiplexing for VPIF
- disable capture VPIF's channels
- configure VPIF's receive channels
- configure VPIF's internal DMA engine

- set up interrupts from VPIF to processor
- enable the receiving channels

When RAW data capture mode is selected the VPIF's input channels work as a pair, thus they must have the same configuration. The following table describes the configuration parameters for CHANNEL0 and CHANNEL1 since they constitute the key configuration parameters of VPIF.

Register	Register Field	Value
C0CTRL\C1CTRL	Clock Edge Control	Data captured on falling edge of input clock
	Data Width (in bits)	12 bits/pixel
	Line Interrupt Interval (CCD/CMOS)	One interrupt per line
	Memory Storage Mode of input picture	Frame –based storage
	Output Display Format	Progressive
	Channel 0 frame interrupt to CPU	Top field V-sync only
	Channel 0 filed identification	Top field
	Channel 0 input data format	Channel 0 Y/C non-multiplexed mode
	Data Capture Mode	CCD/CMOS data capture mode
	Channel 0 enable	Channel is enabled

Table3: VPIF peripheral key register settings

At this point, the receiver is ready to accept the sensor's output. Going back to our primary goal, namely video steaming, we have achieved the first half of it; that is, video capturing.

5.3.2 Video Display

The video display part of the streaming procedure constitutes the process, in which video data is retrieved from memory and displayed to a visual display unit. In terms of this thesis, the memory is the DDR2 memory of OMAP-L138 LCDK from which Liquid Crystal Display Controller (LCDC) loads the video data and feeds them via the VGA connector to a computer monitor. In order to use the raster part of the LCDC controller, a set of initialization steps should take place:

- enable LCDC in the LPSC
- appropriate Pin Multiplexing for LCDC
- disable Raster controller
- configure LCDC's Raster controller
- configure LCDC internal DMA engine
- set up interrupts from LCDC to processor
- enable Raster Controller

The following table describes the key configuration parameters for raster controller of LCDC controller:

Register	Register Field	Value
RASTER_CTRL	12-Bir-Per-Pixel (5-6-5) Mode	Disabled
	TFT Alternative Signal Mapping	Output pixel data for 1, 2, 4, and 8 BPP will be converted to 5-6-5 format and transferred via LCD_D[15:0]
	Palette Loading Mode	Palette and Data
	FIFO DMA Request Delay	10 (Delay Time = [(LCD Pixel Clock) × FIFO_DMA_DELAY])
	Raster Data Order	Frame buffer data is ordered from least-to-most significant bit/nibble/byte/word/d-word
	TFT or STN Mode	Enable Color display operation
	LCD Monochrome or Color	Enable Color display operation
	LCD Raster Controller Enable	Enable the LCD Raster Controller

Table4: LCDC controller key register settings

5.3.3 In-between Video Capture and Video Display

Through the description of video capture and video display it has been purposely concealed the fact that there is a mismatch between the format of the data produced by the image sensor and the data displayed by the LCD device. The sensor produces 12-bit RAW data values while the raster controller feeds the display device with data in RGB565 colorspace. The first step to compromise the gap between data produced by the sensor and data displayed by the monitor, is to convert the

12-bit RAW data to RGB data via **demosaicing**. The next step is to convert the RGB data derived from demosaicing to RGB565 colorspace via **tone mapping**.

Demosaicing

Monochrome imaging requires light intensity values to be measured; color imaging is very much the same, except that the light intensity needs to be measured in different color bands. The

required multi-color band light sampling is addressed via the introduction of **color filter array** (CFA) in sensors designs. A CFA is an array of alternating color filters that samples only one color band at each pixel location. The most popular CFA pattern is the **Bayer pattern**, which features blue and red filters at alternating pixel locations in the horizontal and vertical directions, and green filters organized in the quincunx pattern at the remaining locations [46].This pattern results in half of the image resolution being dedicated to accurate measurement of the green color band. The peak sensitivity of the human visual system lies in the medium wavelengths, justifying the extra green sampling [47].
The introduction of the color filter array results to a mosaic image; at each pixel there is only one spectral measurement which means that the other two spectral measurements have to be estimated through the measurements of neighboring pixels. This estimation is essentially the CFA demosaicing. Through this process three fully populated color planes are created from the CFA data (Figure5.5). Several algorithms exist for this purpose, ranging from simple linear interpolators to high-end nonlinear interpolators that exploit as much spatial and spectral information as possible.

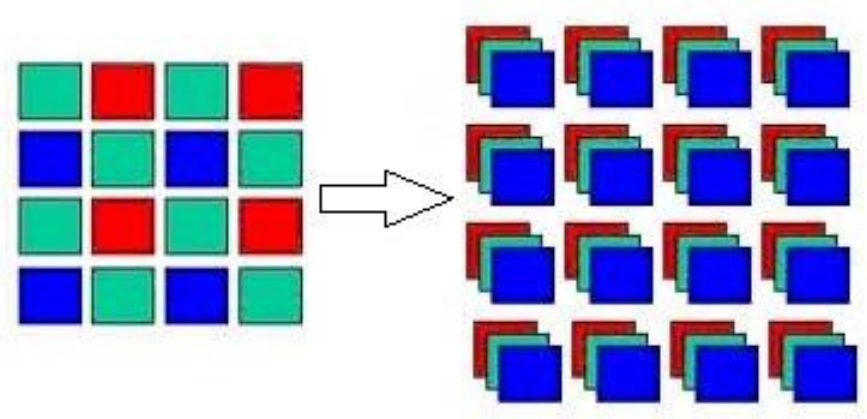


Figure5.7: Red, Green and Blue colors panes extraction from CFA pattern.

As it is inferred by the term "estimation", demosaicing can introduce estimation errors in the resulting three-color pane picture. This happens because sampling a scene using an image sensor

with a Bayer pattern CFA measures only 33% of the information of the original scene. As a result the following artifacts are introduced (Figure 5.6) [44]:

- **false coloring**: This artifact typically manifests itself along edges, where abrupt or unnatural shifts in color occur as a result of misinterpolating across, rather than along, an edge.
- **zippering:** Zipper effect also occurs primarily along edges. Simply put, zippering is another name for edge blurring that occurs in an on/off pattern along an edge
- **blurring:** Blurring is detected in non-uniform color areas where reduced resolution (due to demosaicing) results in loss of detail and sharpness
- **aliasing:** Images with small-scale detail near the resolution limit of the digital sensor can sometimes trick the demosaicing algorithm—producing an unrealistic looking result. This unrealistic result is produced due to aliasing between the green, red and blue spectrums. The most common artifact is moiré, which may appear as repeating patterns, color artifacts or pixels arranged in an unrealistic maze-like pattern [48] [44].

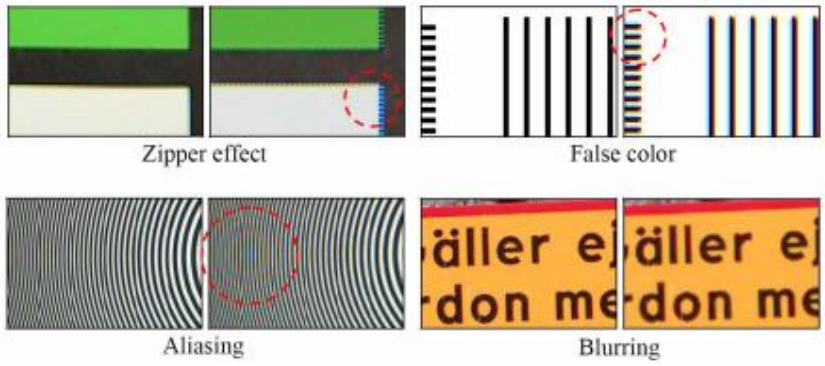


Figure5.8: The most common demosaicing artifacts

A simple and relatively good performing demosaicing algorithm is based on linear interpolation. More particularly, a 3x3 neighborhood is taken from the CFA, and missing pixel values are estimated by the averaging of nearby values. As one can see form the following figure (Figure 5.7) there four possible cases [49]:

- in the first case (Figure5.7 (a)) we have a green near red pixel (**Gr**) so the value of the green color exist and we need to calculate the red and blue color values for that Gr pixel.

$$R_{Gr} = \frac{R_l + R\_r}{2} \qquad B_{Gr} = \frac{B_u + B_d}{2}$$

- in the second case (Figure5.7 (b)) we have a green near blue pixel (**Gb**) so the value of the green color again exist and we need to calculate the red and blue color values for that Gb pixel.

$$R_{Gb} = \frac{R_u + R_d}{2} \qquad B_{Gb} = \frac{B_l + B_r}{2}$$

- in the third case (Figure5.7 (c)) we have a red pixel (**R**) so the value of the red color exist and we need to calculate the green and blue color values for that R pixel.

$$G_R = \frac{G_u + G_d + G_r + G_l}{4} \qquad B_R = \frac{B_{ul} + B_{ur} + B_{dl} + B_{dr}}{4}$$

- in the fourth case (Figure5.7 (d)) we have a blue pixel (**B**) so the value of the blue color exist and we need to calculate the green and red color values for that B pixel.

$$G_B = \frac{G_u + G_d + G_r + G_l}{4} \qquad R_B = \frac{R_{ul} + R_{ur} + R_{dl} + R_{dr}}{4}$$

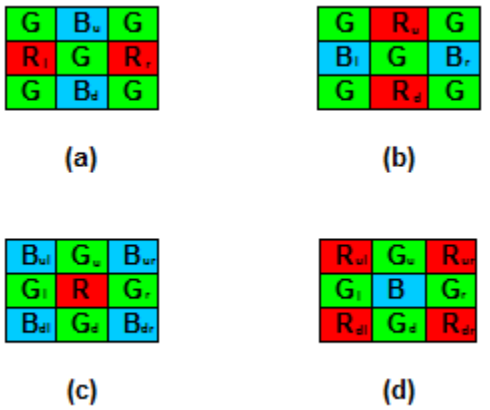


Figure5.9: The four possible cases when interpolating Red and Blue color panes.

One benefit of this method is that it can be performed by a convolution with the appropriate kernels. Two different kernels are required: one for estimating the missing green values (K_G) and one for estimating missing red/blue values ($K_{R/B}$):

$$K_G = \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \qquad K_{R/B} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & 1 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}$$

This interpolation method performs well in smooth areas where the color changes slowly from one to the next. However, when performed along edges where color changes occur abruptly, **false color** and **zipper** artifacts are introduced, resulting in a poor quality.

Tone mapping

Tone mapping is the process of converting the tonal values of an image from a high range to a lower one [50].It constitutes a necessary process when the image reproduced has a higher dynamic range than the reproducing media and this is the case with the RAW data of digital cameras. For example, standard 12-bit image sensors may be able to capture a tonal range of 1,000:1 and this is much more than typical monitors (standard display devices have a dynamic range of about 100:1) or printers can reproduce.
In our case, a frame is reproduced by a 12-bit image sensor and after demosaicing it is represented by three (one for each color channel) 12-bit values. We can say, informally, that the frame is in RGB121212 colorspace, but in order to be displayed via the raster controller it should be in RGB565 colorspace. So, we had to employ a mapping between these two colorspaces. In general, two kinds of tone mapping is used: **linear** and **non-linear** [51]. Linear tone mapping means that higher dynamic range tones are mapped to lower dynamic range tones in a linear fashion. Non-linear (usually logarithmic) mapping is defined analogously (Figure 5.8). Employing a linear tone mapping means that the dark end has very coarse increments and the bright end has very fine increments. This is not compatible with the perceptual sensitivity of the human eye, which is more close to non-linear tone mapping. On the other hand, linear mapping is computationally cheaper, so we used it instead.

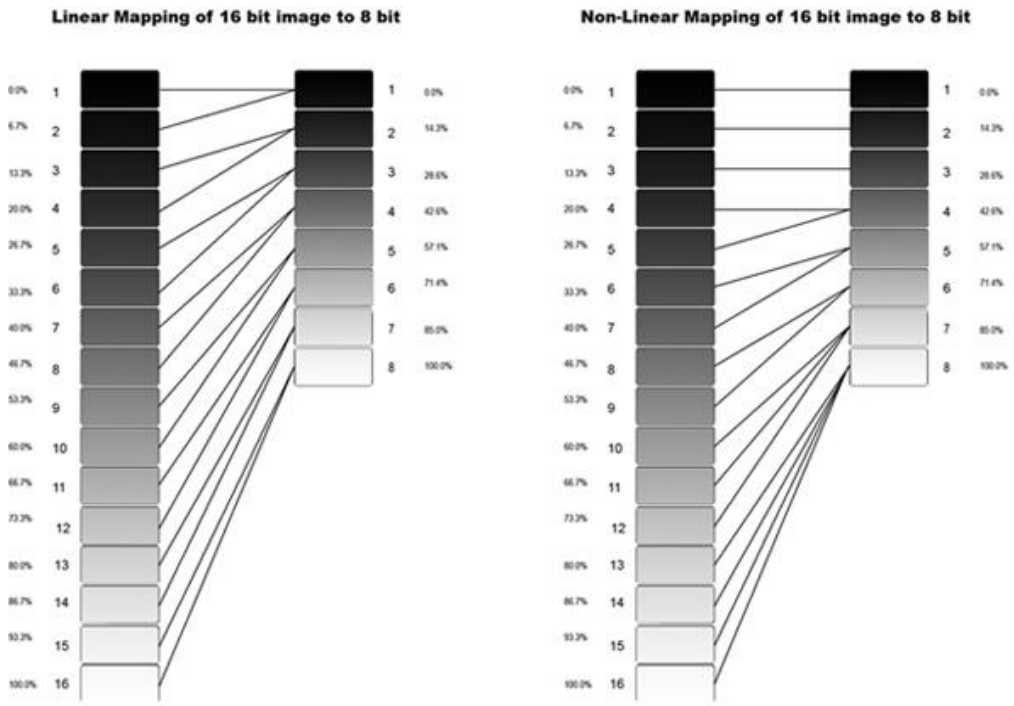


Figure5.10: Linear and logarithmic tone mapping

Before proceeding with linear tone mapping, it is important to mention that **bit depth** and **dynamic range** are separate concepts and there is no direct one to one relationship between them. The bit depth of a capturing or displaying device gives you an indication of its dynamic range capacity, i.e. the highest dynamic range that the device would be capable of reproducing if all other constraints are eliminated. For instance, a bit-depth of 12 for an image sensor tells you that the maximum dynamic range of the sensor is 4096:1, but the captured dynamic range is likely to be much less once noise is taken into account (most 12-bit sensors have on average a dynamic range around 1,000:1 only).This is the case for the image sensor used in this thesis; Produced pixel values should be ranging in the interval [0, 4095], whereas they actually range from 0 to 1023.

Continuing with tone mapping, two linear functions that express the relationship between a 12-bit color value and a 5-bit or 6-bit color value have to be defined. Taking into account the above mentioned about bit-depth and dynamic range and that 5-bit values range from 0 to 31 and 6-bit values from 0 to 63, we define the following equations which, essentially, perform a scaling:

$$R_5/B_5 = R_{12}/B_{12} * \frac{31}{1023} \qquad G_6 = G_{12} * \frac{63}{1023}$$

Summing up, we have explained the two steps required for displaying the captured data. Since we want to traverse our frame the least possible times when we process it, we combined these two steps (demosaicing and tone mapping) in one operation by adjusting the kernels described. More specifically, the new kernels that perform linear interpolation democaising and linear tone mapping from RAW RGB data to RGB 565 are:

$$K_{G_6} = \begin{bmatrix} 0 & \frac{63}{2046} & 0 \\ \frac{63}{2046} & \frac{63}{2046} & \frac{63}{2046} \\ 0 & \frac{63}{2046} & 0 \end{bmatrix} \qquad K_{R_5/B_5} = \begin{bmatrix} \frac{31}{4092} & \frac{31}{4092} & \frac{31}{4092} \\ \frac{31}{4092} & \frac{31}{4092} & \frac{31}{4092} \\ \frac{31}{4092} & \frac{31}{4092} & \frac{31}{4092} \end{bmatrix}$$

We will not go into implementation specific details since the demosaicing approach will be changed. Closing the part of video streaming implementation, which was described via the video capture and video display parts, the following figure is included.

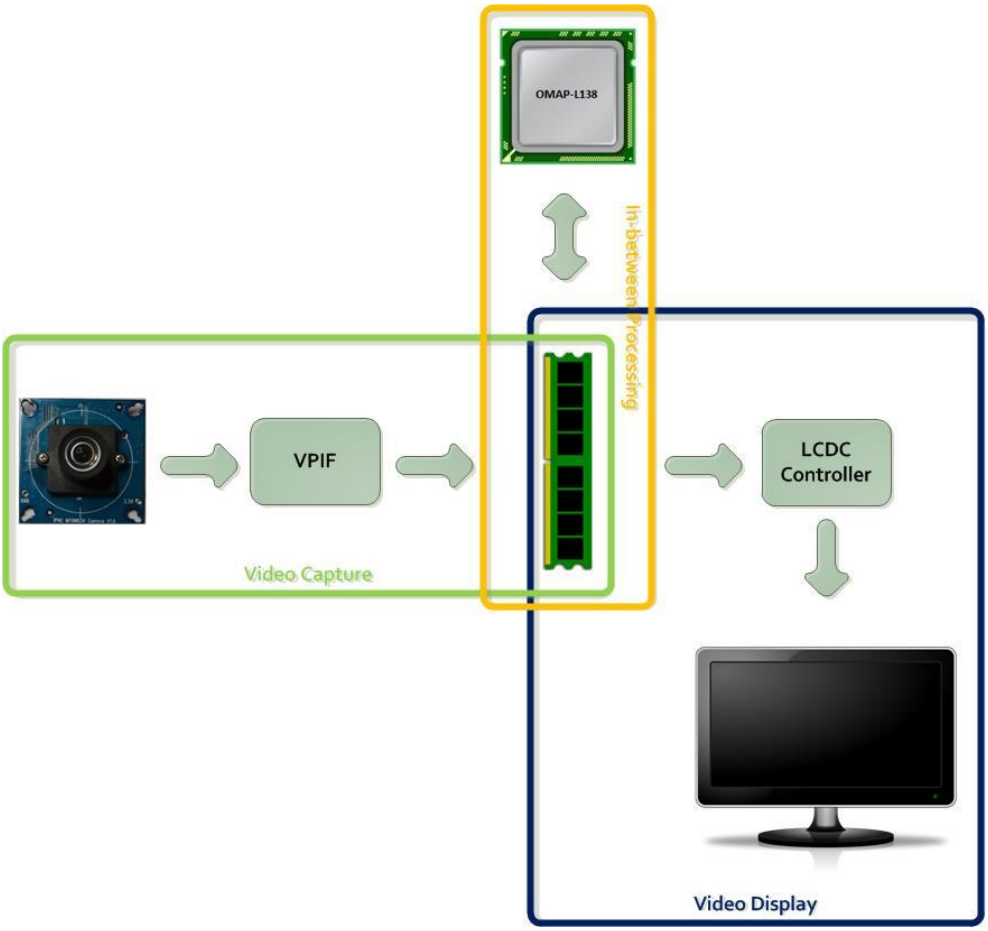


Figure5.11: The Video Streaming Breakdown

Up until now we have only described video streaming in terms of its modules and their configuration. So, a flowchart describing the flow of the video streaming application follows:
As the flowchart suggest the video streaming starts with the initialization and configuration of the image sensor, this includes, primarily detection of the image sensor (through reading its chip version), soft resetting and then configuration of sensor's imaging parameters. Afterwards, the initialization and configuration of VPIF peripheral and LCDC controller follows; including the peripheral initialization steps mentioned and proper configuration through assignment of proper value to the peripheral's registers.
Then the program enters an infinite loop in which program control waits for a frame interrupt from VPIF peripheral. It is important to mention that when a frame interrupt is generated it means that the n-th frame has arrived and begins to be stored in memory while the (n-1)-th frame is already stored in memory awaiting to be processed. When a frame interrupt arrives, before processing it, it is checked if the raster controller has completely transferred to the monitor the previous frame data to be displayed.

At this point we have to mention that data to be displayed are copied in two raster buffers in order to eliminate jitter effects. If the raster controller has finished transferring the data to the monitor a frame can be processed. Each frame processing includes two steps. The first processing step it to cast sensor RAW data to 16-bit value. This is essential due to 12-bit sensor RAW data memory stuffing manner that VPIF peripheral employs; each pixel is padded with four zeros to form a 16-bit value, which occupies two 8-bit memory slots (see Figure 4.10)

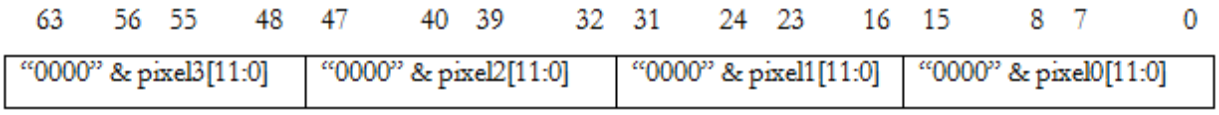


Figure5.12: Stuffing 12-bit RAW data to memory

The second processing step includes demosaicaing and tone mapping as described in pages 65-68. Cache invalidation and write back follows; an essential step in order to make the processed and ready to be displayed frame available in system's external DDR2 memory, so that raster controller can access it and feed it to the display. Then, the program control returns to the infinite loop in order to stream another frame. Closing, we should mention that a double buffering scheme (ping-pong) is employed for effectiveness; while a frame buffer is being filled by sensor's RAW data the other one is being processed, fact that results in concurrent data movement and processing and thus computational effectiveness.

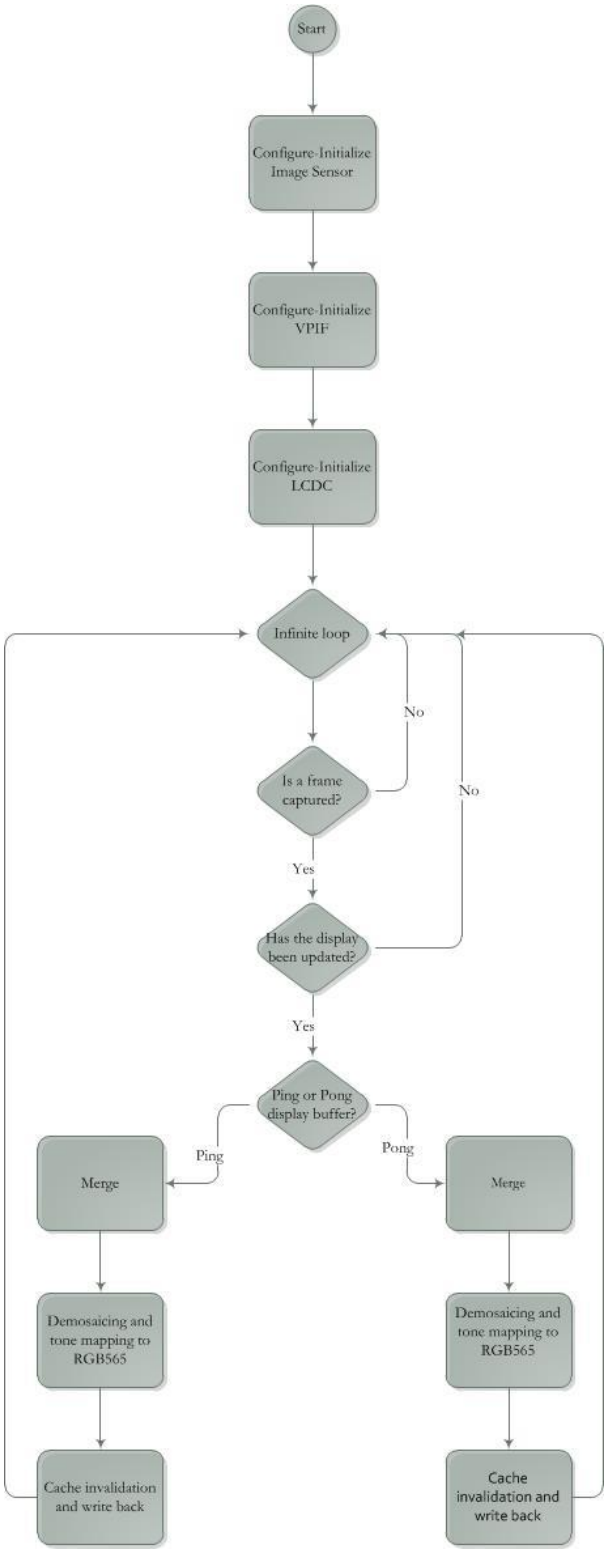


Figure5.13: The Video Streaming Flowchart

5.4 Performance analysis

The most critical and tricky part in a real- time system is performance or in other words how "well" deadlines are met. The term "real-time" is elusive and is often used to describe a wide variety of image/video processing systems and algorithms. From the literature, it can be derived that there are three main concepts of "real-time":

- **real-time in the perceptual sense:** Real-time in the perceptual sense is used mainly to describe the interaction between a human and a computer device for a near instantaneous response of the device to an input by a human user. This concept connotes the idea of a maximum tolerable delay based on human perception of delay, which is essentially some sort of application-dependent bounded response time.
- **real-time in the software engineering sense**: Real-time in the software engineering sense is also based on the concept of a bounded response time as in the perceptual sense. In such a sense logical correctness of the outputs is based both on their correctness and their timeliness [52].
- **real-time in the signal processing sense**: Real-time in the signal processing sense is based on the idea of completing processing in the time available between successive input samples [53].

Being part of the second concept, our application must render both correct and in-time data. Current implementation of video streaming could not keep up with this concept, fact that became obvious when moving the camera board in different directions. Execution time measurements showed that we managed to stream only **8.5 frames per second (fps)** of 640x480 resolution (VGA). As one can imagine the bulky(in terms of processing) part of video streaming is the linear interpolation demosaicing and linear tone mapping routine, which was replaced by a nearest neighbor demosaicing and linear tone mapping routine.
Nearest neighbor demosaicing is the simplest demosaicing algorithm. Using a 2x2 neighborhood (Figure 5.12) from the Bayer pattern CFA, it interpolates missing color pane values (Red ,Green and Blue) by simply adopting the nearest sampled value [54].The blue and red values in this 2x2 neighborhood are used at the three remaining locations. The sampled green values can be moved in either a vertical or horizontal direction to render green color pane information in the red and blue pixels. We have to note that with the advantage of simplicity comes the price of "stronger" demosaicing artifacts, especially along edges.

Figure5.14: The 2x2 neighborhood of nearest neighbor demosaincing.

Now it is time to go into implementation details about demosaicing and tone mapping. The nearest neighbor demosaicing and linear tone mapping routine (just as the linear interpolation demosaicing and linear tone mapping routine) is a C-callable function written in linear assembly.
Linear assembly looks much like assembly language code, but it allows for symbolic names and does not require the programmer schedule events and manage CPU core registers on the DSP. Its advantage over C code is that it uses the DSP more efficiently, and its advantage over assembly code is that it requires less time to program with.

The main concepts on which this function was written are [55]:

- **SIMD:** Working on more than one data value per instruction; in other words packed data processing.
- **Restrict keyword:** the aggressiveness of optimization used by a compiler can be limited since the compiler worries about memory aliasing (single memory object accessed in multiple ways), so in order to ensure the compiler that no bad alias exists we use the restrict keyword as a prefix to the desired pointer. A common example is to use restrict keyword to the pointer of a routine's output data to reassure the compiler that input and output data do not overlap in memory and that it can freely perform software pipelining.
- **Loop unrolling:** Unrolling loops involves replacing iterations of the loop with additional copies of the loop itself. This technique leads to faster, larger code.

Briefly, the concept of the nearest neighbor and demosaicing serial assembly function is as follows: Data (being aligned) is loaded from memory in packets of two 2x2 neighborhoods (see Figure5.13). By processing eight pixels at a time the number of iterations for a given frame size is eight times less, yielding less loops overheads.

Figure5.15: The block.of data loaded per iteration

Each pixel value before contributing its color pane information to its neighbor or to its own is toned mapped as described in pg 68. For example, the first green pixel in a 2x2 neighborhood will scale itself according to G_6 equation and will receive the scaled values of its red and green neighbors according to R_5/B_5 equation. These three scaled values will then be combined by two OR operations to yield a 16-bit value to be displayed of the following form:

$$Pixel\ to\ be\ displayed = R_1R_2R_3R_4R_5G_1G_2G_3G_4G_5G_6\ B_1B_2B_3B_4$$

The usage of nearest neighbor algorithm boosted the performance of video streaming to **20 frames per second (fps)** of 640x480 resolution, but there is still room for improvement.

What we have not specified during the analysis of the video streaming application is on which core(s) it is executed. So far, our implementation is running on the DSP core of the processor SoC. So, in pursuit of better performance, one can employ a multicore processing scheme, so that both the ARM and DSP cores contribute in the execution of video streaming.

In order to achieve this, the lightweight **inter-processor communication** (IPC) module of StarterWare was used. By using this module both cores have full access to the device memory map; this means that any core can read from and write to any memory. In addition, there is support for direct event signaling between cores for notification .The underlying hardware mechanism that realizes the communication between the two cores is very simple; Five bits (SYSCFG_ CHIPSIG [0-4]) located in the SYSCFG system configuration module enables signaling between the DSP and ARM. Writing one to any of these bits will generate a signal or event. These events are fed to the respective interrupt controller (INTC) to get mapped to the core interrupt inputs. To pass data, any of the 128KB internal or 512MB external memory areas can be used as shared memory [42].

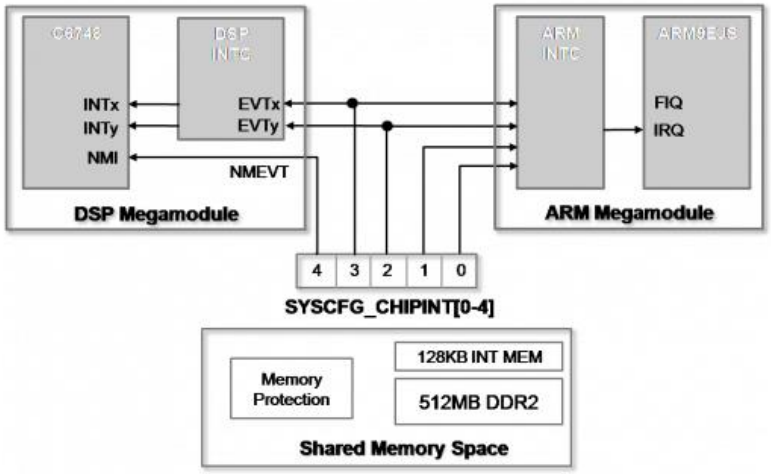


Figure5.16: The mechanism of inter-processor communication

The video streaming application was mapped to the two cores as the following flowchart suggests (Figure 5.16). More particularly, the application starts with the ARM processor initializing and configuring the inter-processor communication by setting appropriate values to an IPC structure, registering the interrupt to be generated form DSP and finally synchronize itself with the DSP. The ARM then starts the notify mechanism with the DSP, initializes and configures the LCDC and enters an infinite loop waiting for a DSP event (a frame is ready to be displayed). DSP on its behalf starts with initializing and configuring the image sensor and the VPIF peripheral. Afterwards, DSP also starts the notify mechanism, it then enters an infinite loop and waits until a frame is captured.

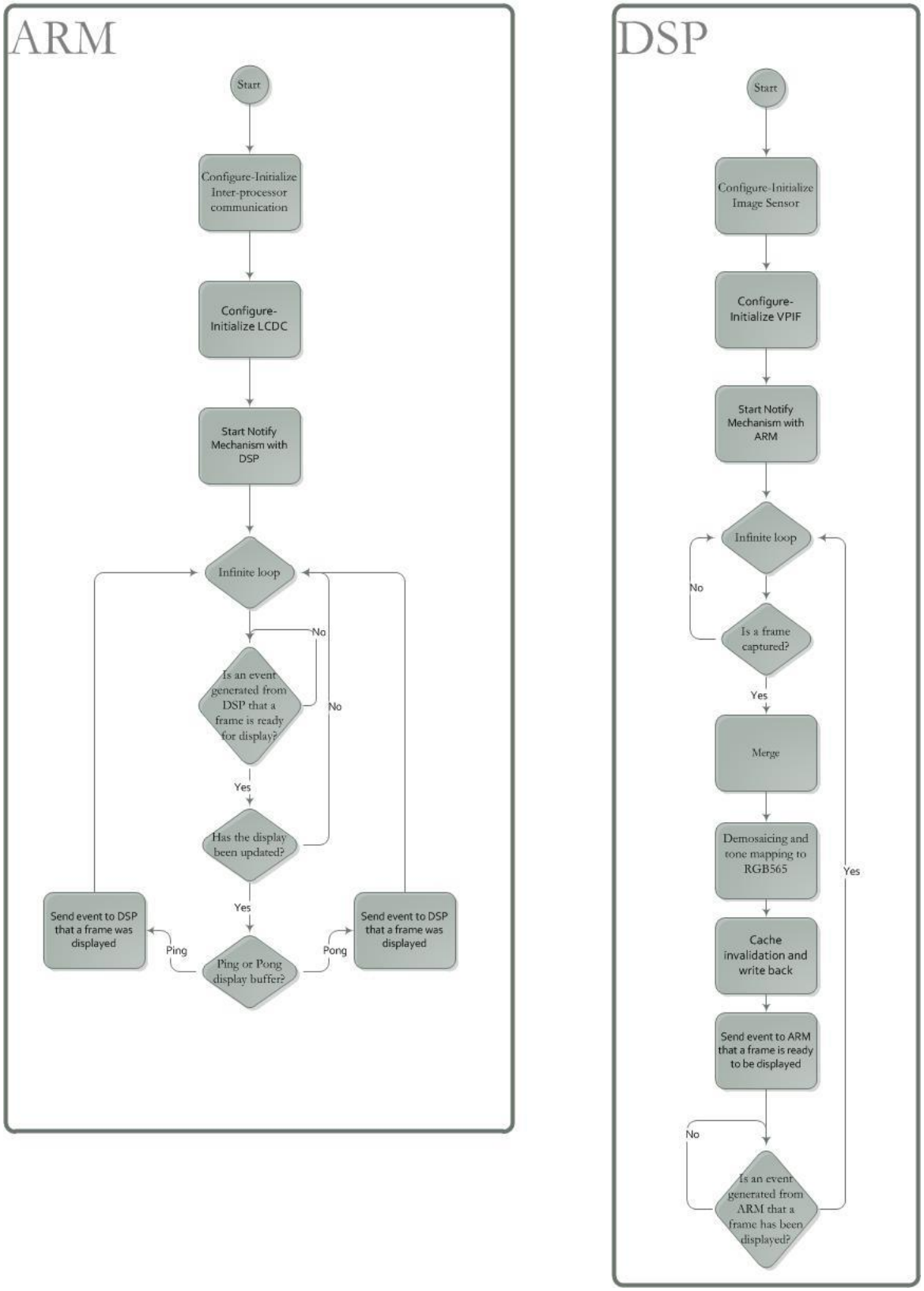


Figure5.17: The Video Streaming flowchart when executed by both ARM and DSP.

When this happens, DSP process the frame and then notifies the ARM with an interrupt that a frame is ready to be displayed. Then DSP waits an event from ARM informing it that the frame has been displayed. ARM understands from that signal that a frame is ready to be displayed. The ARM displays it and notifies DSP that himself had displayed the frame and that DSP is able to capture a new one.

A table with the frame rates achieved in the video streaming application follows. The frame rates for both demosaicing approaches are calculated; the best performing demosaicing approach is then evaluating while being execute in both cores (not just DSP).

Sensor configuration		**Linear demosaicing- Linear Tone mapping@DSP**	**Nearest Neighbor demosaicing - Linear Tone mapping@DSP**	**Nearest Neighbor demosaicing - Linear Tone mapping@DSP+ARM**
640*480@55 fps	Frames captured	6461	2273	3695
	Frames processed	1000	1000	1000
	Time (sec)	118	50	37
	Frames per second	8.4	20	26.3

Table5: Performance in terms of frames per second for three different processing schemes

At first we notice that nearest-neighbor demosaicing is approximately two times faster than linear-interpolation demosaicing. This happens due to the fact that nearest-neighbor demosaicing is a uniform repeated operation, enabling the compiler to perform better software pipelining. On the other hand, linear-interpolation demosaicing is not so uniform; two different masks are applied meaning that a condition should be evaluated each time a mask is to be applied, fact that results in a not so aggressive pipelining. In addition, the kernels are applied to overlapping areas of the frame meaning that pixels of a certain area have to be accessed more than one time.
Just, to get an impression of the performance in sensor's full resolution (1280x960@45fps) we count the frame rates for the nearest-neighbor demosaicing executed by DSP; the frame rate achieved was 6.6 fps. This result was expected since the size of the frame has become four times bigger comparing to the frame size of 640x480, so a frame rate about 25% of the one achieved for the 640*480 resolution is reasonable. Afterwards, the frame rate for the nearest-neighbor demosaicing executed by both DSP and ARM was calculated; its value was 7.4 fps, constituting an also expected result.

Concluding, the processing scheme of nearest-neighbor demosaicing and linear tone mapping executed by both cores offers the best performance and as result the platform smart application will be built around it.

6 Developed Applications

Having gained a main functionality, it's time to outsmart the image sensor. This is done via a set if four different applications: "De-noising during Acquisition", "Ultra-fast Spectrometer", "Human Vision Emulation" and "Pseudocolor Mapper".

6.1 De-noising during Acquisition

Images taken with both CCD and CMOS images will pick up noise from a variety of sources. Noise can compromise the level of detail in photos and generally degrade the image quality. As it had been mentioned in Chapter 2 CMOS image sensors are, in general, more susceptible to noise than CCDs. The good news is that noise can be reduced post acquisition with several techniques. The problem is that most techniques to reduce or remove noise always end up softening the image as well. Some softening may be acceptable for images consisting primarily of smooth-uniform areas, but areas with fine details within the frame can suffer severely even with conservative attempts to reduce noise.
Image averaging based noise removal has the power to reduce noise without compromising detail, because it actually increases the signal to noise ratio (SNR) of your image. An added bonus is that averaging may also increase the bit depth of your image- beyond what would be possible with a single image. Image averaging works on the assumption that the noise in your image is truly random. This way, random fluctuations above and below actual image data will gradually even out

as one averages more and more images. If you were to take two shots of a smooth gray patch, using the same camera settings and under identical conditions (temperature, lighting, etc.), then you would obtain intensity values similar to those shown by the red and blue line on the next figure (Figure6.1) [56].

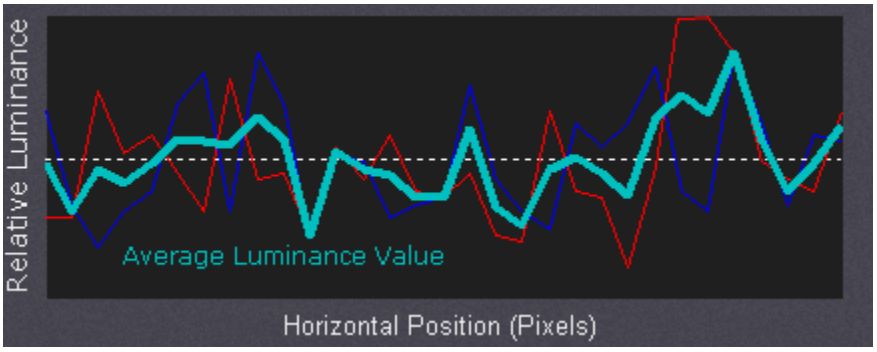
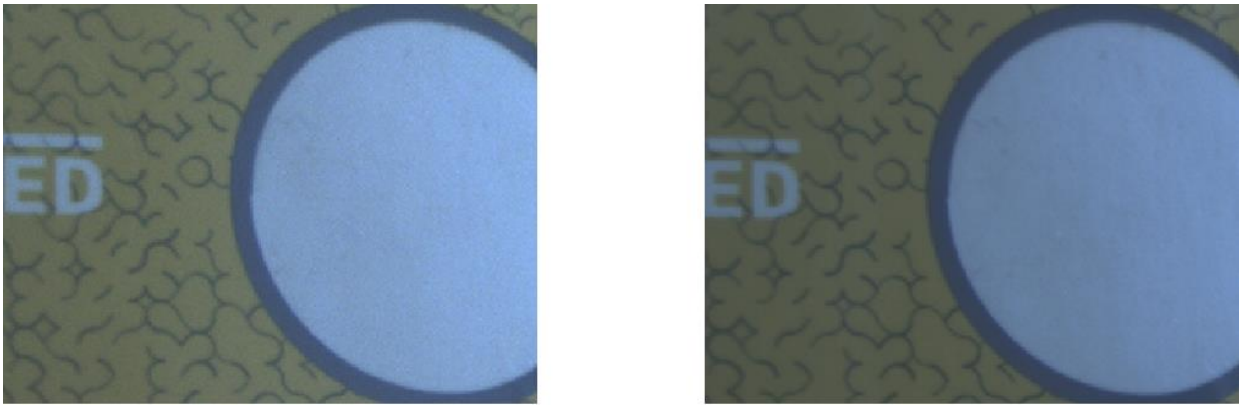


Figure6.1: Intensity values of two identical images (red-blue lines) and the intensity of the average produced by these intensities (light-blue line).

The dashed horizontal line represents the average, or what this plot look like if there were zero noise. If we were to take the average point by point between the red and the blue line, then the intensity variation would be reduced as it showed by the light blue line.
Even though the average of the two still fluctuates above and below the mean, the maximum deviation is greatly reduced. In general, magnitude of noise fluctuation drops by the square root of the number of images averaged, so you need to average four images in order to cut the magnitude in half.
Based on these, an application was developed that performs averaging during acquisition. The number of frames to be averaged had been chosen to be eight, thus the magnitude of noise fluctuation drops tree times approximately.



(a) (b)

Figure6.2: The left image (a) is derived from one frame whereas the right (b) is averaged from eight frames yielding less noise (especially visible in the white circle)

The flowchart of the application is the same as the one in pg. 77 only that the DSP proceeds in demosaicing only when eight frames are captured and averaged. We have to note that the averaging of the intensity values of the eight frames is incorporated to the merged function. More particularly, the intensity values of the first frame are just merged (as described in pg. 72).The intensity values of the subsequence six frames are merged and added to the intensity values of the previous captured frame each time. Finally, the intensity values of the eighth frame are merged and added to the corresponding intensity values of the previous seventh frame and simultaneously the average of the added values is calculated.
As one can infers, averaging eight frames to yield just one frame restrict the frame rate severely. More particularly, let's assume that the sensor is configured to render frame with VGA resolution (640x480) taking into account the pixel clock frequency and making certain calculations this means that the maximum frame rate that can be achieved is 88 fps. As a result, the frames derived from averaging targeted for processed would be 11 per second so without even the overhead of processing we already have a low frame rate. To conclude "De-noising on capture would be real time if the size of the frame to be averaged could output by the sensor in a frame rate approximately 30*8 =240 fps. Performing appropriate calculations the frame size able to be rendered in 240 fps by the sensor is 240x160.

6.2 Ultra-Fast Spectrometer

Spectrometers are used to measure the properties of light for a variety of applications including environmental or chemical analysis, fluorescence, or Raman.
The basic function of a spectrometer is to take in light, break it into its spectral components, digitize the signal as a function of wavelength, and read it out and display it through a computer (see Figure 6.3).

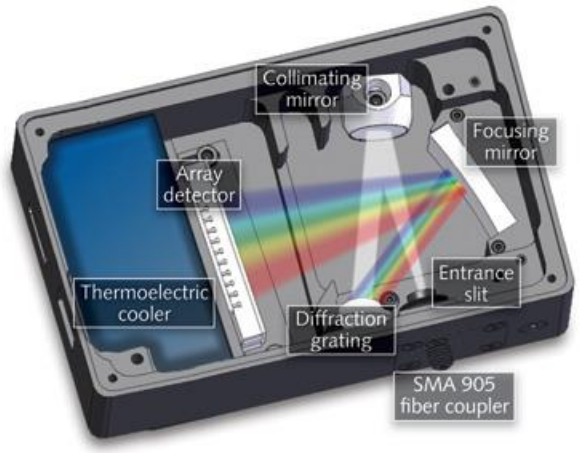


Figure6.3: The block diagram of a spectrometer

The first step in this process is to direct light through a fiber optic cable into the spectrometer through a narrow aperture known as an entrance slit. In most spectrometers, the divergent light is then collimated by a concave mirror and directed onto a grating. The grating then disperses the spectral components of the light at slightly varying angles, which is then focused by a second concave mirror and imaged onto the detector. As the incident light strikes the individual pixels across the detector, each pixel represents the intensity value of a portion of the spectrum.
The developed platform can be used to develop a reconfigurable ultra-fast spectrometer by taking the place of the **array detector** in the above-mention spectrometer set up. The term **reconfigurable** is used since the programmer has full freedom in choosing which row/rows of the image sensor's array will be read and of what width it/they will be. Additionally, the programmer can perform pixel **binning**, an on-sensor chip operation that by combining adjacent pixel values yields higher signal-to- noise ratio. Binning is important to a spectrometer for two reasons: primarily because it reduces noise and as a result achieves higher sensitivity and secondly because technically it's difficult to direct light to a micro sized area (pixel).So one could use multiple pixels to direct light more easily in them with the additional benefit of higher sensitivity. More precisely, in our case, the programmer can read two rows and combine their values by row-wise binning or read one row and combine its values by column-wise binning. An example follows to illustrate this point: a window of size 1280x2 can be selected from the sensor and row-wise binning can be performed, this will result in a 1280x1 window of intensity values. Analogously, a 1280x1 window can be selected, column-wise binning can be performed in it yielding a 640x1 window of intensity values. The term **ultra- fast** is justified by the high frame rates achieved (see Figure 6.4).

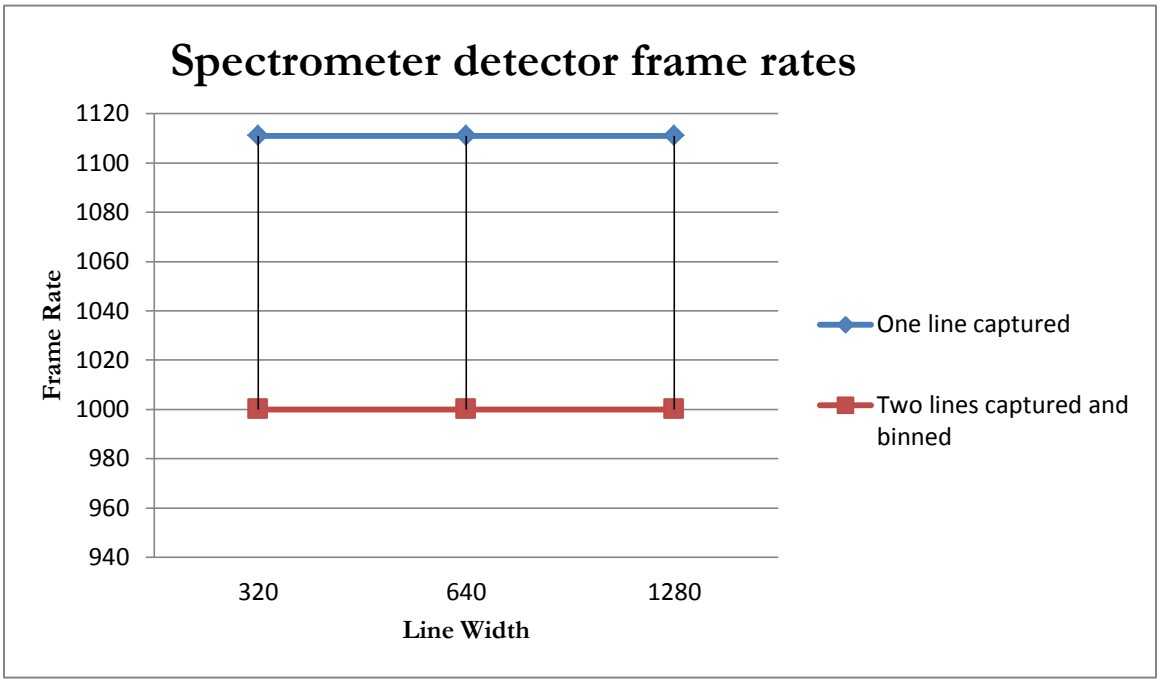


Figure6.4: Frame rates for different line widths achieved

An interesting fact to note about Figure6.4 is that the frame rate is independent to the line width. Fact that is justifiable since a CMOS image sensor is accessed pretty much like a RAM. In other words, sequential access introduces no additional latency.
Finally, we have to point out that the introduction of a two-dimension imaging array instead of a single- dimension is by itself advantageous since its renders two-dimensional spectral measurements.
At this point we have to mention that this application was developed modifying the video streaming application. The demoisaicing part and video display part of video streaming were removed since spectrometry does not to interpolate pixel values to generate an image to be displayed. It requires a processing (calibration) related to the structure of the system in order intensity data to be plotted as a function of wavelength, which was not implemented since the whole spectrometer set up was not developed. The flowchart of the application follows:

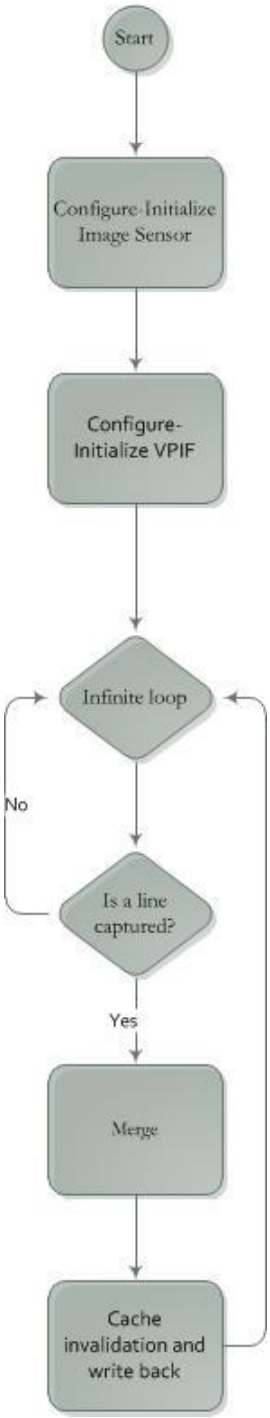


Figure6.5: The ultra-fast spectrometer flowchart.

6.3 Human Vision Emulation

The human visual system can be regarded as consisting of two parts. The eyes act as image receptors which capture light and convert it into signals that are then transmitted to image processing centers in the brain. In accordance, the **image sensor** can be regarded as the **human eye** and the brain processing centers as the processing module. The human eye senses light through a thin tissue placed at its back end; the retina. The retina acts pretty much like a film and it consists of two main categories of cells: nerve cells and light-sensitive cells. There are two general classes of light sensitive cells; rods and cones. Rod cells are very sensitive and provide visual capability at very low light levels. Cone cells perform best at normal light levels.
An interesting fact about the retina is that it is not uniformly light-sensitive; this is attributed to the fact that the distribution of rods and cones is not uniform across the retina. This is the very property of human eye that our developed platform tries to mimic.
More analytically concerning the distribution of rods and cones; the cones are concentrated towards while the rods away from the center of the retina .The central area of the retina is called **macula** (Figure 6.6) and it is responsible for detailed vision such as reading or seeing details straight ahead. The remaining area of the retina outside of the macula is responsible for peripheral vision. So one could say that the macula is responsible for "what is it" vision, while re rest of the retina is responsible for "where is it" vision [58].

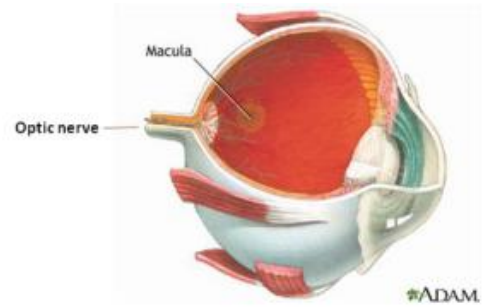


Figure6.6: Central area of the retina of the human eye depicted, namely the macula.

Summing up, the human eye being packed with **rod** cells in its center yields a **fine** in terms of resolution area. Simultaneously, around this area the human eye consist mainly of **cone** cells offering a **coarse** in terms of resolution area. Exploiting two of the image sensor's on-chip features mentioned in (pg. 37) we are able to offer an overview of a scene in coarse resolution accompanied with an area of interest in fine resolution. The two features are **Binning** and **Real Time Context Switching**. The main concept of the application is that the sensor can toggle via context switching between a binned overview frame and a full resolution (non-binned) region of interest (ROI). The application was based on the video streaming application in which the DSP part was modified, thus in the following flowchart we present only the DSP part of the application.

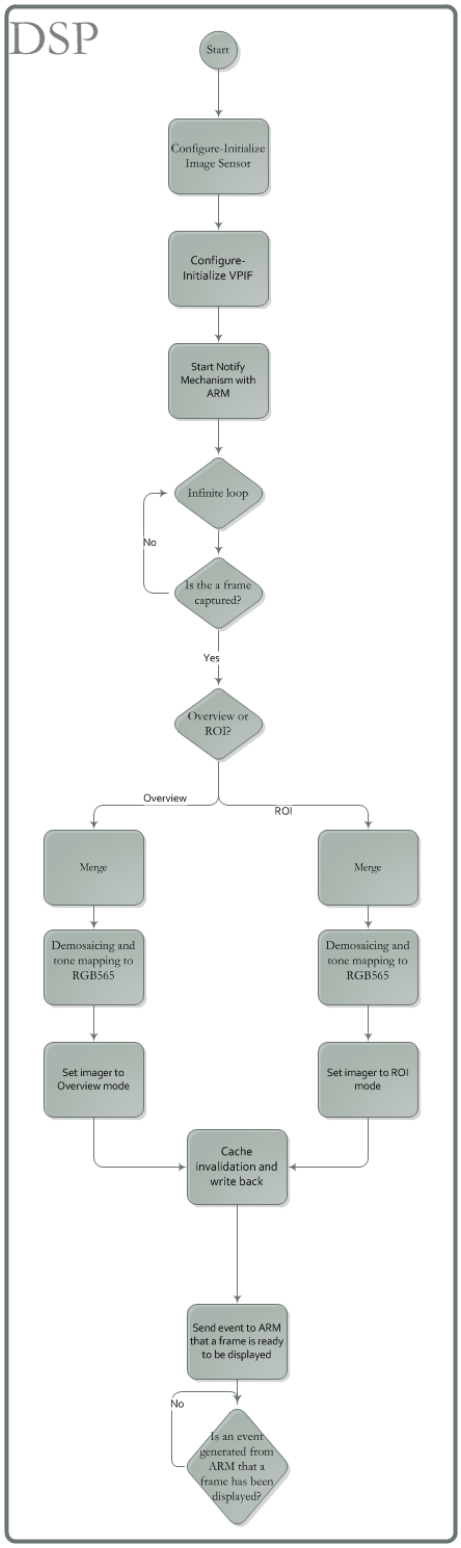


Figure6.7: Human Vision Emulation flowchart.

A chart with different scenarios of overviews accompanied with areas of interest follows:

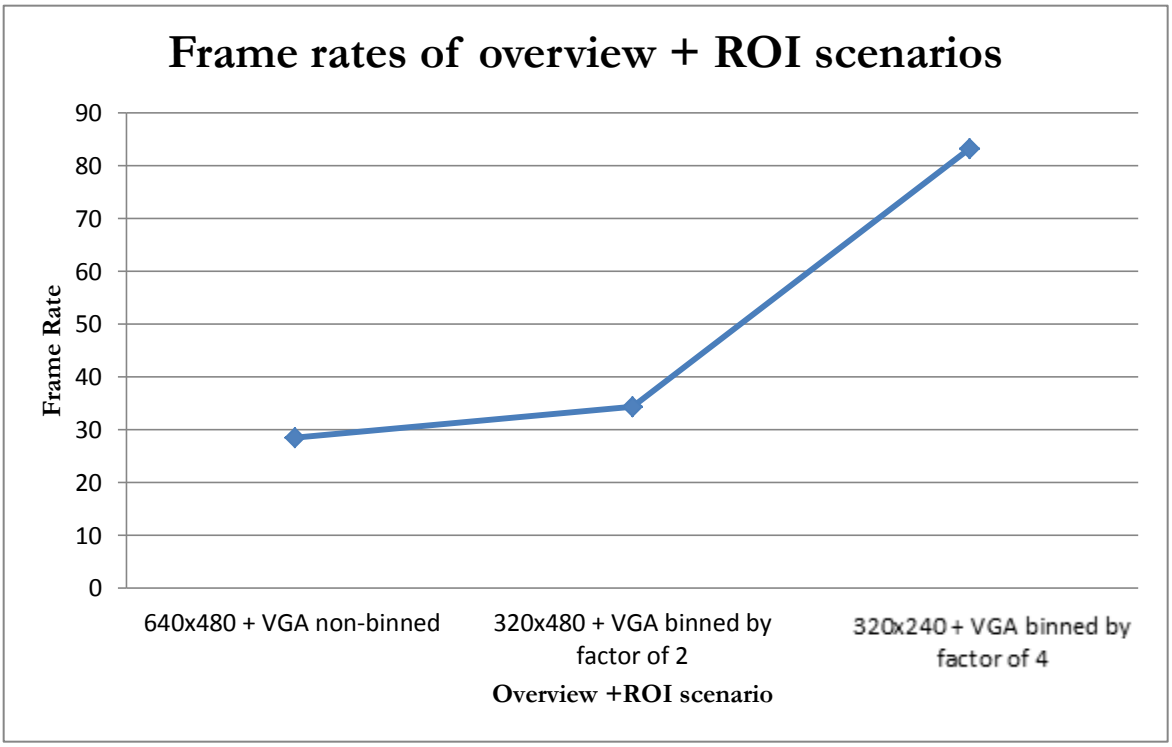
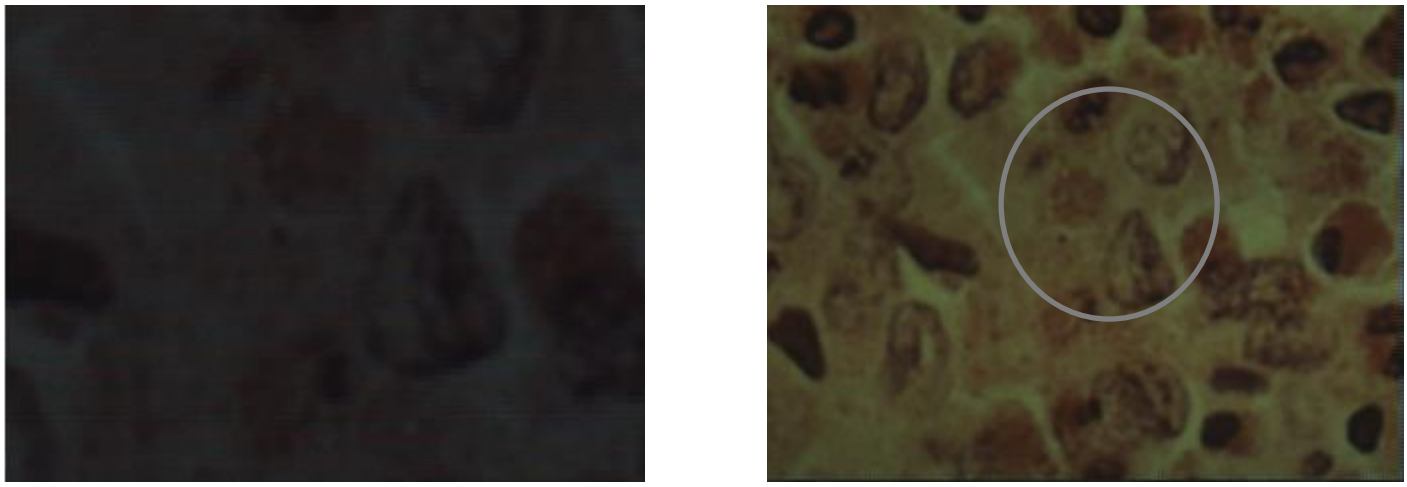


Figure6.8: Frame rate increases with binning factor.

The primary benefit of binning is higher signal-to-noise ratio (SNR) due to reduced read noise. However, in our case we are interested in the fact that binning reduces resolution while preserves field of view, resulting in rendering the same information concerning a scene in lower resolution. As the above diagram suggests, a region of interest of 640x480 (in full resolution- fine resolution) accompanied by an overview of 640x480 in full resolution can be obtained in frame rates around 28 fps. When a region of interest of 320x240 (in full resolution-fine resolution) is accompanied with an overview of 640x480 binned by factor of four (coarse resolution) the frame rate increases to 84 fps approximately. So, it is derived that binning can be a mean of achieving high frame rates when working with large imaging arrays; a region of interest is in full resolution while the overview of the imaging array is in coarse resolution (binned). What is displayed by our platform follows;

the area in the grey circle constitutes the region of interest on the left which is in four times bigger resolution than the overview on the right.



(a) (b)

Figure6.9: On the right (b) an overview (640x480 binned by factor of four) is offered and on the left there is an area of interest (320x240 full resolution).

It is worth noticing the difference in exposure of the overview and the ROI; this is imposed by the fact that we want to obtain both the overview and ROI with the same frame rate. Frame rate is analogous to frame size and exposure time as a result the frame rate and exposure imposed by the overview is also used for the ROI even if it could be retrieved in a higher frame rate.

6.4 Pseudocolor Mapper

False-color image sacrifices natural color rendition in order to ease the detection of features that are not readily discernible otherwise. As the human eye uses three "spectral bands" these three spectral bands are commonly combined into a false-color image. In our case, we combine the Red and Green spectral bands. More particularly, we calculated the Red to Green channel ratio in order to obtain an index for locating veins in human body.
The Red to Green ratio values are represented by the colormap "jet" in which higher Red to Green ratios (more red) are mapped towards red and lower Red to Green ratios (less red) towards blue.

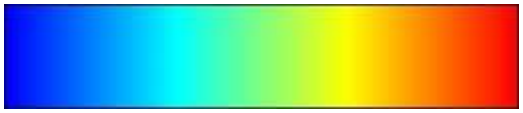


Figure6.10: The jet colormap.

As all of the previous mentioned applications,"Pseudocolor Mapper" is also based on the video streaming application in which the DSP part was appropriately modified. After merging now we do not need demosaicing instead we calculate the Red to Green ratio for every 2x2 neighborhood as it was defined for demosaicing.

Figure6.11: The 2x2 neighborhood for Red to Green Ratio calculation.

For example, the Red to Green ratio for the first row of pixels is the same and it is calculated by dividing the intensity value of the red pixel by the intensity value of the first-row-green pixel. The ratio for the second row is also the same and its calculated by dividing the intensity value of the red pixel by the intensity value of the second-row-green pixel. We employed the same block processing scheme as mentioned in pg.75.
A colormap is an m-by-3 matrix of real numbers between 0.0 and 1.0. Each row is an RGB vector that defines one color. Due to the fact that we need the resulted ratios had to range between 0.0 and 1.0, we had to perform normalization since ratios are not "naturally" bounded. In order to achieve this, we had to find the higher ratio each time and divide all the Red to Green ratios by it. After that we were ready to map the resulted ratio to the appropriate row of the jet colormap. At this point we have to mention that we used a colormap of 128 rows, thus 128 different colors, which was implemented as a look up table. We also have to note that our colormap is not 128x3 but 128x1 since Red, Green and Blue values are combined to a single 16-bit value in the RGB565 format.
The maximum ratio was found using a relative function from the DSP libraries. The mapping was implemented again according to the processing scheme in pg.75. More particularly, the mapping was just calculating the row of the colormap; this was done by multiplying each ratio with the map factor given by the equation:

$$colormap_{row} = ratio_{current} * \frac{127}{ratio_{\max}}$$

Finally, since accessing the external system memory to fetch value is considered a high price operation, we placed the colormap in an internal memory that DSP offer (L2 memory).

Again, we can draw the same conclusions (pg. 88) regarding binning and frame rate as in the "Human Vision Emulation" application. It is worth noticing that in this appplications lower frame rates are achieved; this happen due to the need for traversing more than once a frame in order to perform calculations.

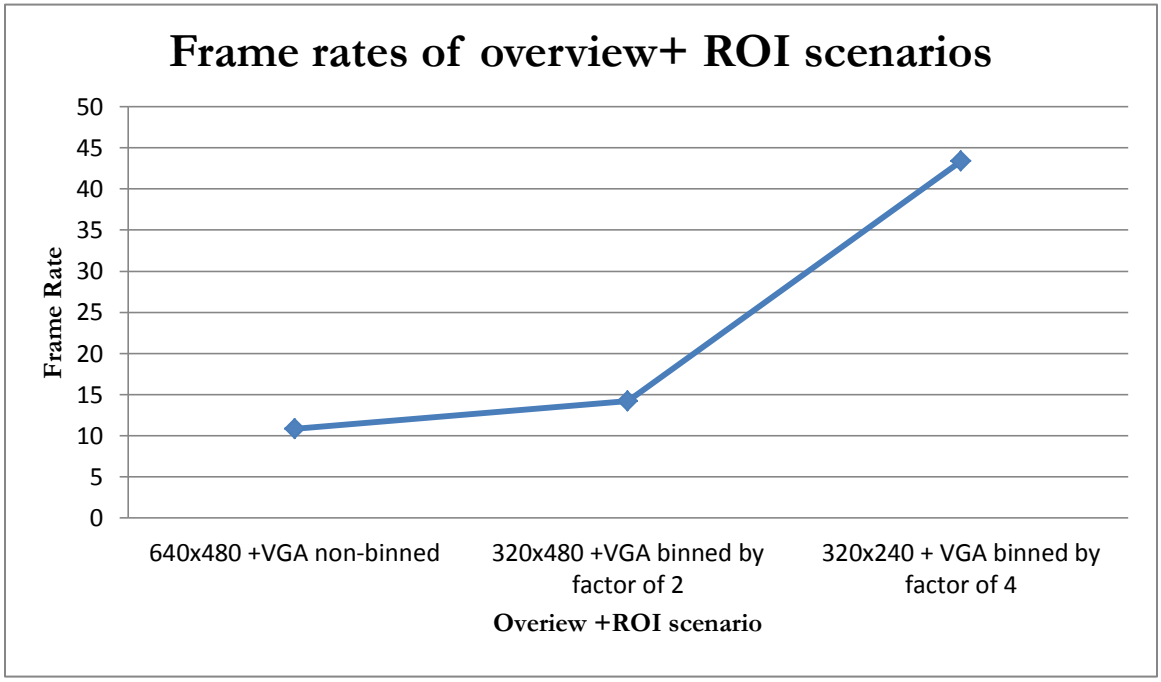


Figure6.12: Frame rate increases with binning factor.

A photo of a hand follows in which pseudocolor mapping is performed (Figure 6.13). Where veins exist the area turns yellow, indicating less red than the rest of the image which is towards dark orange.

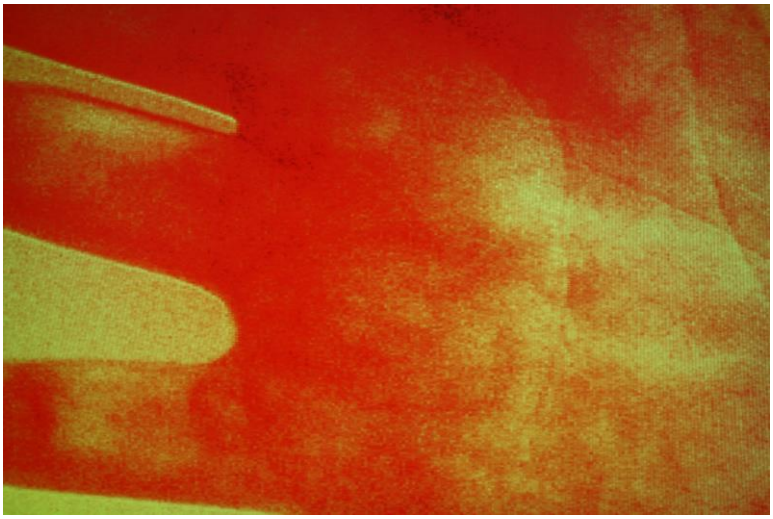


Figure6.13: Psedocolor image of a hand

The flowchart of the application follows:

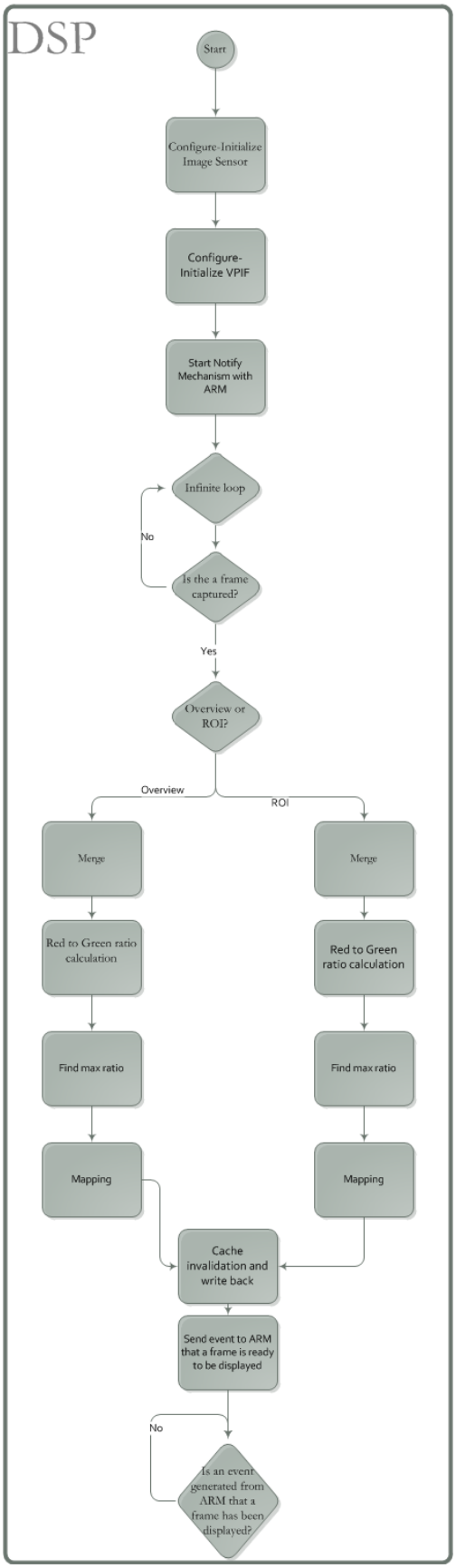


Figure6.14: The "Psedocolor Mapper" flowchart

7 Conclusion

In this thesis, the first step was to carry an exhaustive research in trends of hardware platforms that would be suitable for controlling a CMOS image sensor. During this search we become familiar with different concepts of hardware architecture, but the most important thing that we become familiar with, is the effectiveness of marketing. How well a product will be marketed is far more important to how robust it is. The next step was to efficiently use the capabilities both of the selected image sensor and platform in order to fulfill, to the greatest possible extent, our targets. In some cases our targets was not reached, but in others they were surprisingly surpassed. Closing, I would like to say that the developed hardware platform is according to industry trends, but not comparable to commercial high-end products.

7.1 Hardware-Related Conclusions

The developed platform as a CMOS image sensor controller and frame extractor is adequate. But if it were for a more complicated application to be performed on acquired frames two possibilities should be concerned. The first one is to replace the image sensor by another that would selective perform on-chip demosaicing on its data, so that the processing module will be relieved by the demosaicing processing. The second one is migrating to an analogous processing module of higher processing power.

7.2 Software-Related Conclusions

As far as the software concerns, the most important conclusion that was made was realizing the trade-of between portability and customization. If one wants to develop something optimal exploiting the architectural features of the underlying platform, he should be ready to start over in a possible migration of an application developed to another hardware platform.

7.3 Future Work

There are many scenarios to be proposed as future work, but there are two important aspects to be addressed regardless of what the future of this platform will be. These are: memory-based performance optimization and system connectivity.

7.3.1 Memory Management

One major problem with current memory subsystems is that they were originally designed for one-dimensional data access and thus cannot properly address the spatial locality necessary for two-dimensional or three-dimensional image data. For instance, the horizontally adjacent pixels Img2D[i][j] and Img2D[i][j + 1] are separated only by a few bytes in memory, whereas the vertically adjacent pixels Img2D[i][j] andImg2D[i + 1][j] are separated by several bytes of pixel data in memory [57].
As it expected, image data to be processed should be placed in processor's cache and taking into account that cache misses poses barriers to real-time implementations, one want to achieve the least possible cache misses. Due to the fact that many pixel- level operations (including demosaicing) require access to neighboring (horizontally and vertically) pixels, cache misses can be a serious source of performance loss.
So, we could boost current performance by employing a DMA data transferring scheme instead of caching to feed the processor chip data from the external DDR2 memory. The DMA controller (EDMA3) would manage the movement of data without CPU assistance, leaving it free to focus on time critical computations rather than becoming engaged in data management. More precisely, a multi-buffering DMA data transfer scheme could be used; where efficient usage of multiple buffers, usually placed within on-chip memory, would allow us to perform concurrent processing and movement of data.

7.3.2 Connectivity

It is of paramount importance in a smart camera system to be able to feed the streaming data to the outside world through an established protocol. The OMAP-L138-LCDK is equipped with standard I/O interfaces and storage interfaces. The platform is able to process VGA resolution in frame rates higher than 30 fps so we will examine connectivity assuming VGA resolution. The desired connection speed is:

$$desired\ speed_{16-bit\ quality} = bits\ per\ pixel * resolution * frames\ per\ second$$
$$= 16 * (640 * 480) * 30 = \ 147456000\ bit/s = 140.63\ Mbit/s$$

From available I/O interfaces USB2.0 (480 Mbit/s) could sufficiently handle streaming data in VGA resolution real-timely. The Ethernet (100Mbit/s) could be also used (as it is suggested from the next equation) if the video data is of 8-bit quality.

$$desired\ speed_{8-bit\ quality} = bits\ per\ pixel * resolution * frames\ per\ second$$
$$= 8 * (640 * 480) * 30 = \ 73728000\ bit/s = 70.31\ Mbit/s$$

References

[1] E. Davies, Machine Vision: Theory, Algorithms, Practicalities. San Francisco, CA: Morgan Kauffmann Publishers (2005)

[2] TELEDYNE DALSA: CCD vs. CMOS, Which is better? It's complicated… http://www.teledynedalsa.com/imaging/knowledge-center/appnotes/ccd-vs-cmos/

[3] Kenneth R. Spring, Thomas J. Fellers and Michael W. Davidson: Introduction to Charge-Coupled Devices (CCDs), Nikon Microscopy U http://www.microscopyu.com/articles/digitalimaging/ccdintro.html

[4] John Coghill, DALSA: Digital Imaging Technology 101 (2003) https://classes.yale.edu/04-05/enas627b/lectures/EENG427l07DigitalImaging.pdf

[5] Molecular Expressions, Optical Microscopy primer, Digital Imaging in Optical Microscopy, Renato Turchetta , Kenneth R. Spring, Michael W. Davidson: " Introduction to CMOS image sensors" http://micro.magnet.fsu.edu/primer/digitalimaging/cmosimagesensors.html

[6] ST Journal of System Research: Image Processing for Digital Still Camera (2001) http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.3072&rep=rep1&type=pdf

[7] Bio-Imaging Systems ,CCD vs. CMOS white paper (2006) http://www.berthold-jp.com/pdf/ccd_vs_cmos.pdf

[8] Gretchen Alper: "CCD vs. CMOS Image Sensors in Machine Vision Cameras" (2011) http://info.adimec.com/blogposts/bid/39656/CCD-vs-CMOS-Image-Sensors-in-Machine-Vision-Cameras

[9] Dave Litwiller, DALSA: CCD vs. CMOS Facts and Fiction Photonics Spectra (January 2010)

[10] DALSA, white paper : "Application Set Imager Choices"

[11] Nasser Kehtarnavaz and Mark Gamadia: Real-Time Image and Video Processing: From Research to Reality, Morgan & Claypool (2006)

[12] H. Hunter and J. Moreno: "A New Look at Exploiting Data Parallelism in Embedded Systems," Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, pp. 159–169 (October/November 2003)

[13] H. Broers, W. Caarls, P. Jonker, and R. Kleihorst: "Architecture Study for Smart Cameras," Proceedings of the European Optical Society Conference on Industrial Imaging and Machine Vision, pp. 39–49 (June 2005)

[14] E. Davies, Machine Vision: Theory, Algorithms, Practicalities. San Francisco, CA: Morgan Kauffmann Publishers (2005)

[15] J. Ackenhusen, Real-Time Signal Processing: Design and Implementation of Signal Processing Systems. Englewood Cliffs, NJ: Prentice-Hall (1999)

[16] Texas Instruments, Inc.: TMS320DM320 CPU and Peripherals Technical Reference Manual (2004)
[17] M. Houston: General Purpose Computation on Graphics Processors (GPGPU). Stanford University Graphics Lab, http://graphics.stanford.edu/~mhouston/ (2005)
[18] C. Bruyns and B. Feldman: "Image Processing on the GPU: A Canonical Example," Computer Architecture Course Project, Department of Computer Science, University of California Berkeley (Fall 2003)
[19] R. Yang and M. Pollefeys: "A Versatile Stereo Implementation on Commodity Graphics Hardware," Journal of Real-Time Imaging, Vol. 11, No. 1, pp. 7–18 (February 2005)
[20] Kenji Tajima , Akihiko Numata, Idaku Ishii, Photron Limited,1-1-8 Fujimi, Chiyoda-ku: "Development of a high-resolution, high-speed vision system using CMOS image sensor technology enhanced by intelligent pixel selection technique" , Dept. of Artificial Complex Systems Engineering, Graduate School of Engineering, Hiroshima University, 1-4-1 Kagamiyama, Higashi-Hiroshima, Tokyo, JAPAN
[21]b Dept. of Artificial Complex Systems Engineering, Graduate School of Engineering,
[22]Hiroshima University, 1-4-1 Kagamiyama, Higashi-Hiroshima, JAPAN
[23] Design and implementation of an automatic traffic sign recognition system on TI OMAP-L138 Phalguni, Kishan Ganapathi, Venugopala Madumbu, Resmi Rajendran and Sumam David: Department of Electronics and Communication Engineering, National Institute of Technology Karnataka, Surathkal, India, Texas Instruments (India), Bangalore, India (2013)
[24] Michael Bramberger, Andreas Doblander, Milan Jovanovic, Arnold Maier, Bernhard Rinner, and Allan Tengg: Embedded Smart Cameras as Key Components in Reactive Sensor Systems. In COGnitive systems with Interactive Sensors (COGIS) (2006)
[25] Michael Bramberger, Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach: Distributed Embedded Smart Cameras for Surveillance Applications. IEEE Computer, 39(2):68–75 (February 2006)
[26] Michael Bramberger, Roman P. Pflugfelder, Arnold Maier, Bernhard Rinner, Bernhard Strobl, and Helmut Schwabach: A Smart Traffic Camera for Stationary Vehicle Detection. In Proceedings of the Workshop on Intelligent Solutions in Embedded Systems (WISES). Vienna University of Technology (June 2003)
[27]Markus Quaritsch, Markus Kreuzthaler, Bernhard Rinner, Horst Bischof, and Bernhard Strobl. Autonomous Multicamera Tracking on Embedded Smart Cameras. EURASIP Journal on Embedded Systems (Special Issue on Embedded Vision System) (2007)
[28] Aptina Imaging: MT9M024 and MT9M034: Register Reference Register Descriptions Rev. B (2011)
[29] Aptina Imaging: MT9M034: 1/3-Inch CMOS Digital Image Sensor Features Rev. E (2010)
[30] Texas Instruments Wiki: "L130/C6748 Development Kit (LCDK)" http://processors.wiki.ti.com/index.php/L138/C6748_Development_Kit_%28LCDK%29
[31] Texas Instruments Wiki: "OMAP" http://en.wikipedia.org/wiki/OMAP
[32] ARM: "ARM926EJ-S™ Technical Reference Manual", Revision: r0p5 (2001-2008) http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0198e/index.html
[33] Texas Instruments: TMS320C674x DSP Megamodule Reference Guide (August 2010) http://www.ti.com/lit/ug/sprufk5a/sprufk5a.pdf
[34] Texas Instruments: TMS320C674x DSP CPU and Instruction Set Reference Guide (July 2010) http://www.ti.com/lit/ug/sprufe8b/sprufe8b.pdf

[35] Texas Instruments: OMAP-L138 DSP+ARM Processor Technical Reference Manual (December 2011) http://www.ti.com/lit/ug/spruh77a/spruh77a.pdf
[36] Texas Instruments Wiki: "OMAP-L1x/C674x/AM1x SoC Architectural Overview" http://processors.wiki.ti.com/index.php/OMAP-L1x/C674x/AM1x_SoC_Architectural_Overview
[37] Oshana, Rob: "Introduction to JTAG" Embedded Systems Design. (5 April 2007)
[38] ITU-R Recommendation BT.656: Interfaces for digital component video signals in 525-line and 625-line television systems operating at the 4:2:2 level of Recommendation ITU-R BT.601 (Part A).
[39]http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1120-8-201201-I!!PDF-E.pdf
[40]Texas Instruments Wiki: OMAP-L138 Software Design Guide http://processors.wiki.ti.com/index.php/OMAP-L138_Software_Design_Guide
[41] Michael Melkonian: Get by without an RTOS, Embedded Systems Programming VOL. 13 NO. 10 (September 2000) http://w3.ualg.pt/~rmarcel/get%20by%20without%20an%20rtos.pdf
[42] Texas Instruments Wiki: "StarterWare 01.10.01.01 User Guide" http://processors.wiki.ti.com/index.php/StarterWare_01.10.01.01_User_Guide
[43] Aptina Imaging: Leveraging Dynamic Response Pixel Technology to Optimize Inter-scene Dynamic Range, An Aptina White Paper (2010) http://www.aptina.com/products/technology/DR-Pix_WhitePaper.pdf
[44] Robert A. Maschal Jr., S. Susan Young, Joe Reynolds, Keith Krapels, Jonathan Fanning, and Ted Corbin : Review of Bayer Pattern Color Filter Array (CFA) Demosaicing with New Quality Assessment Algorithms http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA513752
[45]Robert Oshana: DSP Software Development Techniques for Embedded and Real –Time Systems, Elsevier Inc (2006)
[46] Bayer, B. E. Color Image Array. U.S. Patent 3 971 065 (July 1976)
[47] Li, X.; Gunturk, B.; Zhang, L. Image Demosaicing: A Systematic Survey. In Proc. SPIE, volume 822, 68221J (2008)
[48]Cambridge in colors: "Demosaicing Artifacts", http://www.cambridgeincolour.com/tutorials/camera-sensors.htm
[49] RGB "Bayer" Color and Microlenses http://www.siliconimaging.com/RGB%20Bayer.htm
[50] HDRSoft, www.hdrsoft.com/resources/dri.html#tone_mapping
[51] Raul Moreno: "Digital Images 8-bit vs. 10-bit and Log vs. Linear" (2010) http://www.qvolabs.com/Digital_Images_ColorSpace_Log_vs_Linear.html
[52] E. Dougherty and P. Laplante: "Introduction to Real-time Imaging", Bellingham, WA/Piscataway, NJ: SPIE Press/IEEE Press (1995)
[53] N. Kehtarnavaz: "Real-Time Digital Signal Processing Based on the TMS320C6000",.Amsterdam Elsevier (2004)
[54] Adams, J. E. :Interactions Between Color Plane Interpolation and Other Image Processing Functions in Electronic Photography. In Proc. SPIE, volume 2416, pages 144–151 (1995)
[55] Texas Instruments, Technical Training Organization: TMS320C6000 Optimization Workshop (March 2011)

[56] Cambridge in colors: "Noise reduction by image averaging " (2014) http://www.cambridgeincolour.com/tutorials/image-averaging-noise.htm
[57] B&W Tek: "Spectrometer Knowledge" (2014) http://bwtek.com/spectrometer-introduction/
[58] Costas Balas: "Lecture1: Human Vision" Advanced Topics in Electronic Imaging-HPY 603 (2013)
[59] K. Brifault and H. Charles: "Data Cache Management on EPIC Architecture: Optimizing Memory Access for Image Processing," ACM SIGARCH Computer Architecture News, Vol. 32, No. 3, pp. 35–42 (June 2004)