PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

YAHIA FARES UNIVERSITY OF MÉDÉA



Faculty of Technology

Department of Electrical Engineering

Master thesis

Specialty: Electronic of embedded systems

Real time video processing system based on FPGA

Presented by :

KADDOUR DJILLALI

BENIELLES OUSSAMA

Proposed and Directed by :

Pr. OULD ZMIRLI MOHAMED

Promotion 2020

Dedication

I Dedicate This Work To :

To The One Who Taught Us To Love Life And Respect Others:

Mothers

To The One Who Taught Us To Always Keep My Head High And Never Give Up On My Dreams:

Father's

To Our Supervisor Who Helped Us The Most And Have Been Always Supportive And Collaborative When Ever Its Need:

Mr.Pr.Mohamed Ould-Zemirli

To Those I Love And Who Supported Me Throughout This Project:

Our Entire Family.

Our Class Mate.

To The One And All Those Who Contributed From Near Or Far To Make This Project Possible, Particularly

Mr.A. Sahraoui.

Acknowledged

First of all alhamdulillah we are so grateful and thankful and fully appreciated to allah that he bless us and guide us to reach this point.

We owe so much to our parents due to their support and advices and especially their patient with us through our journey.

We would like to thank our advisor Pr.M.ould zemirli for his insightful comments, strict requirements, and continuous support during our graduate studies.

abstract:

In our work we extracted the uses, benefits, performances and features of computer vision and applied it in embedded systems that is known as embedded vision, with the assistances of the high performance of FPGA ZYBO Z7 and Pcam C5 as hardware, and Vivado HLS, and Xilinx SDK as software to create a video streaming processing system, that contain multiple blocks (edge detection, color inverter and color to gray), as a result we achieved real time data processing and streaming video enhancement.

Keywords :

Embedded System, Real Time, Embedded Vision, Edge Detection, Pcam 5c, Zybo Z7.

Résumé:

Dans notre travail, nous avons extrait les utilisations, les avantages, les performances et les fonctionnalités de la vision par ordinateur et l'avons appliquée dans les systèmes embarqués qui est connu sous le nom de vision embarquée, avec l'aide de la haute performance de FPGA ZYBO Z7 et Pcam C5 comme matériel, et Vivado HLS, et Xilinx SDK comme logiciel pour créer un système de traitement de streaming vidéo, qui contiennent plusieurs blocks (detection de contour , inversion de couleur et couleur a gris), en conséquence nous avons atteint le traitement des données en temps réel et l'amélioration de la vidéo en streaming.

Mots clés :

Système embarqué, Temps réel, Vision embarqué, Détection des contours, Pcam 5c, Zybo Z7

الملخص:

لقد قمنا في عملنا هذا باستخراج واستغلال الاستخدامات، الميزات، المحاسن والاداء الموجود في رؤية الكمبيوتر وطبقناها في الأنظمة الدمجة وهذا ما يعرف بالرؤية المدمجة وذلك بالاستعانة باللوحة ذات الاداء العالي FPGA ZYBO Z7 وكاميرا C5 Pcam كمعدات وVivado HLS وXilinx SDK كبرامج لأنشاء نضام لمعالجة تدفق الفيديو, الذي بدوره يحتوي على وحدات المعالجة (استخراج الحواف، عكس الالوان، من الملون الى الرمادي)، بفضل هذا النظام استطعنا معالجة تدفق البيانات في الوقت الحقيقي وتحسينها .

الكلمات المفتاحية:

الرؤية المدمجة، الرؤية المدمجة الوقت الحقيقي، الكشف عن الحافة, Pcam 5c s, Zybo Z7.

TABLE OF CONTENTS:

General Introduction…. .(1)

Chapter I: Embedded Systems based on the Xilinx Zynq All Programmable SoC……....(3)

I.1 Introduction……...(4)

I.2 Embedded system .....(4)

I.3 Embedded System Classification… .....(5)

I.4 Applications of Embedded Systems .....(6)

I.5 Embedded System Characteristics… .....(8)

I.5.1 Limiting memory…..(8)

I.5.2 Energy consumption…. (8)

I.5.3 Autonomy…. (8)

I.5.4 Mobility…. (9)

I.5.5 Communication… .....(9)

I.5.6 Security…..(9)

I.5.7 Real-time… .....(9)

I.6 Generic Embedded System Architecture .....(10)

I.6.1 Processors…. .(11)

I.6.2 Co-processors .....(12)

I.6.3 Interrupts… .....(13)

I.6.4 Buses… .....(14)

I.6.4.1 System and Peripheral Buses… .....(15)

I.6.4.2 Bus Masters and Slaves .....(16)

I.6.4.3 Bus Arbitration… .....(16)

I.6.4.4 Memory Access .....(16)

I.6.4.5 Bus Bandwidth… .....(18)

I.7 System-on-Chip with Zynq… .....(19)

I.8 Simple Anatomy of an Embedded SoC… .....(22)

I.9 Design Reuse… .....(24)

I.10 Raising the Abstraction LeveL .....(25)

I.11 SoC Design Flow….....(26)
I.12 Conclusion….....(28)
ChapterII: Embedded Vision .....(29)
II.1. Introduction….....(30)
II.2. Embedded Vision architectures… .....(30)
II.3. Embedded Vision Components….....(31)
II.3.1. Sensors… .....(32)
II.3.1.1. Sensor History….....(32)
II.3.1.2. The Charge-Coupled Device….....(33)
II.3.1.3. CMOS Sensors… .....(35)
II.3.1.4. Readout and Control… .....(36)
II.3.2. Processing Unit .....(37)
II.3.3. Embedded vision standard interfaces….....(38)
II.4. Image Operations .....(49)
II.5 Applications for Embedded vision… .....(42)
II.5.1. Industrial Applications .....(42)
II.5.2. Medical Applications .....(42)
II.5.3. Automotive Applications .....(42)
II.5.4. Security Applications .....(43)
II.5.5. Consumer Applications .....(43)
II.6.conclusion… .....(45)
ChapitreIII : Real time video processing system realization .....(46)
III.1. Introduction .....(47)
III.2. Zybo Z7 Board….....(48)
III.3 Pcam 5C imaging module.....(50)
III.3.1 Pcam 5C Module… .....(50)
III.3.2 MIPI CSI-2 Interface .....(51)
III.4. Block diagram of the realized system… .....(51)
III.5. Used Vivado Video IP Cores from the xilinx library ……..(52)

III.6. HLS Video Processing Cores… (54)
III.6.1. Blur Edge Detect… (54)
III.6.2. Color to BW… (55)
III.6.3. Invert Colors… (56)
III.7. Software… (56)
III.7.1. Initialization… (56)
III.7.2. Pipeline mode change… (57)
III.8. Conclusion… (59)
General Conclusion (60)

LIST OF FIGURES :

Figure 1.1: Embedded system… .....(4)
Figure 1.2: Embedded System Classification… .....(5)
Figure 1.3: Embedded system application areas .....(6)
Figure 1.4: Generic embedded system architecture… .....(10)
Figure 1.5: Bus comprised of control, address and data lines .....(14)
Figure1.6: DMA transfer operation… .....(18)
Figure1.7: Comparison of System-on-a-Board (top) and System-on-Chip(bottom)…..…(20)
Figure 1.8: A simplified model of the Zynq architecture ....(22)
Figure 1.9: The hardware system architecture of an embedded SoC(simplified)… .....…(22)
Figure 1.10: Relationship of the software system, hardware system, and Zynq architecture……(24)
Figure 1.11: basic model of the design flow for Zynq SoC….....…(26)
Figure 2.1: Embedded vision block diagram .....(32)
Figure 2.2: Charge readout from a CCD, a shift register……...(33)
Figure 2.3: Overview of pixel acquisition in a sensor…..... ....(35)
Figure 2.4: Schematic of one pixel of a CMOS active pixel sensor…..... ....(36)
Figure 2.5: Mobileye driver assistance system…….…(43)
Figure 2.6: Microsoft Kinect for Xbox 360, a gesture-based game controller…….....(44)
Figure 3.1: Zybo Z7Board…….....(48)
Figure 3.2: image sensor with OmniBSI™ technology (5 megapixel)… .....…(50)
Figure 3.3: Video Pipeline Operation Flowchart……....(51)
Figure 3.4: MIPI RX and Preprocessing……....(52)
igure 3.5: Video Stream Mux/Demux…….....(53)
Figure 3.6: Dynamic Clock Generator .....,.(53)
Figure 3.7: Video Output…….....(54)

Tables list :

Table 2.1 Data Interdependency in Image Space…..... (41)

Abbreviations list:

ADCs and DACs: Digital-to-analog converter / Analog -to- digital converter

APS : active pixel sensor

ARM : Acorn RISC Machine

ASIC : Application-specific integrated circuit

CCD : Charge-Coupled Device

CMOS : complementary metal-oxide-semiconductor

CPU: central processing unit

CRT: cathode ray tube

DDR3L: Double Data Rate

DMA: Direct memory access

DSP: Digital signal processing

FPGA : Field Programmable Gate Arrays

GPPs : General-purpose Polystyrene

GPU : graphics processing unit

HLS : high level synthesis

IP : Intellectual Property

LVDS : low voltage differential signaling

RAM : random access memory

ROM : Read-only memory

RTL : real time logic

SBC : single board computer

SDK : Software development kit

SoC : system on chip

SoM : system on module

TFT : Thin Film Transistor

USB : Universal Serial Bus

General introduction

Embedded Vision is growing rapidly and is increasingly finding its way in many sectors and application areas, EV is an exciting new technology. It has the potential to open up new applications and shape the future of entire industries. It's a revolutionary concept with far reaching implications, it has been a subject of increasing interest and rigorous research for decades now. As a way of emulating the human visual system, the research in the field of EV purports to develop machines that can automate tasks that require visual cognition.

In addition to the rapidly increasing importance and variety of EV applications, this domain of embedded systems warrants specialized focus because EV applications have a number of distinguishing requirements compared to general-purpose systems and other embedded domains. For example, in low- to middle-end general-purpose systems, and in domains of embedded computing outside of EV, performance requirements are often significantly lower than what we encounter in EV. Cost and power consumption considerations are important for some areas of EV as they are in other areas of consumer electronics. However, in some areas of EV, such as medical imaging and surveillance, considerations of real-time performance and accuracy dominate. Performance in turn is strongly related to considerations of buffering efficiency and memory management due to the large volumes of pixel data that must be processed in EV systems. This convergence of high-volume, multidimensional data processing; real-time performance requirements; and complex trade-offs between achievable accuracy and performance gives rise to some of the key distinguishing aspects in the design and implementation of EV systems. These aspects have also helped to influence the evolution of some of the major classes of embedded processing devices and platforms including field programmable gate arrays (FPGAs).

Thus, in our work we exploited the features of Digilent ZYBO Z7, as it is the newest addition to the popular ZYBO line of ARM/FPGA SoC Platforms, beside that we used the Pcam 5C due to their combability with ZYBO board to accomplish our attempt , so our work is divided into 3 chapters, In the first chapter we generalized the concepts of embedded systems including their classification, characteristics, architecture and some of the most actual applications , then the second chapter we talked about the artificial vision in embedded systems as is knowing the

embedded vision, we explain the architecture, components ,some basic of image operations ,and its newest applications, lastly in the third chapter we demonstrated the use of the Digilent Embedded Vision Bundle, Xilinx Vivado, Xilinx Vivado HLS, and Xilinx SDK to create a video processing system.

Chapter 1

Embedded Systems based on the Xilinx Zynq All Programmable SoC

Chapter 1: Embedded Systems based on the Xilinx Zynq All Programmable SoC

I.1. Introduction

The term ‘embedded system’ has become widely used when describing a large number of different applications, with a large number of platforms now available. These range from small microcontrollers and DSPs, to large FPGAs and GPPs, and defining what exactly comprises an embedded system is growing more and more challenging.

The aim of this chapter is to introduce the concept of embedded systems and to provide some examples of where they would be deployed in terms of real-life applications. The scope will then be narrowed to focus on embedded systems, and the generic embedded system architecture will be explored. Individual components of the architecture will be detailed, starting with the processor, and a high-level overview of some of the fundamental operations, such as interrupts, will be discussed. Some basic bus operations and properties, such as bus arbitration, memory transfers and bandwidth, will also be covered [1].

I.2. Embedded system:

As its name suggests, Embedded means something that is attached to another thing. An embedded system can be thought of as a computer hardware system having software embedded in it, an embedded system can be an independent system or it can be a part of a large system. An embedded system is a microcontroller or microprocessor-based system which is designed to perform a specific task.

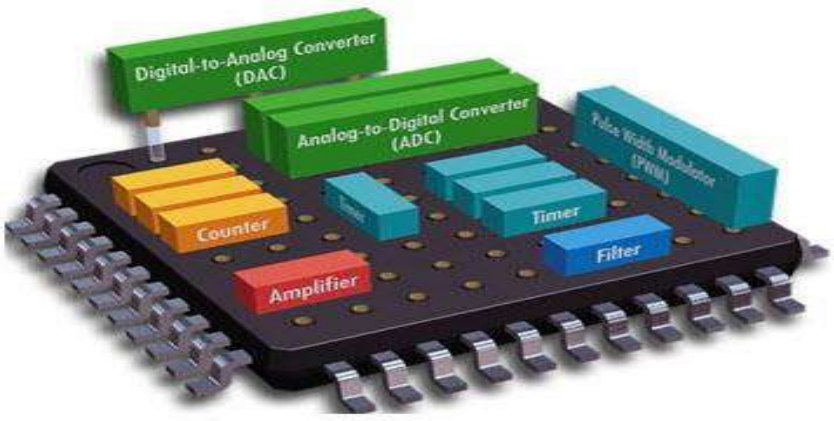


Figure 1.1: Embedded system

An embedded system is a combination of computer hardware and software, either fixed in capability or programmable, designed for a specific function or functions within a larger system, Embedded systems are computing systems, but they can range from having no user interface (UI) for example, on devices in which the system is designed to perform a single task -- to complex graphical user interfaces (GUIs), such as in mobile devices. User interfaces can include buttons, LEDs, touchscreen sensing and more. Some systems use remote user interfaces as well.

An embedded system is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electrical system.[2][3] It is embedded as part of a complete device often including electrical or electronic hardware and mechanical parts. Because an embedded system typically controls physical operations of the machine that it is embedded within, it often has real-time computing constraints. Embedded systems control many devices in common use today.[3] Ninety-eight percent of all microprocessors manufactured are used in embedded systems.[4]

I.3. Embedded System Classification:

Embedded systems are primarily classified into different types based on complexity of hardware & software and microcontroller (8 or 16 or 32-bit).

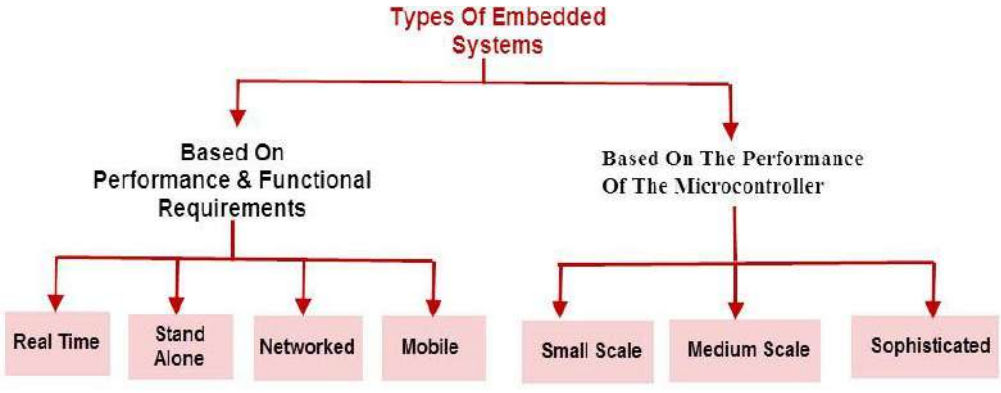


Figure 1.2: Embedded System Classification.

Further, based on performance and functional requirements of the system embedded system classified into four types such as:

- Real time embedded systems
- Stand alone embedded systems
- Networked embedded systems
- Mobile embedded systems

Thus, based on the performance of the microcontroller, embedded systems are classified into three types such as:

- Small scale embedded systems
- Medium scale embedded systems
- Sophisticated embedded systems

I.4.Applications of Embedded Systems:

Embedded systems are deployed in a large number of devices across a breadth of fields. An overview of some of the more prominent uses for embedded systems is provided in Figure 1.3.

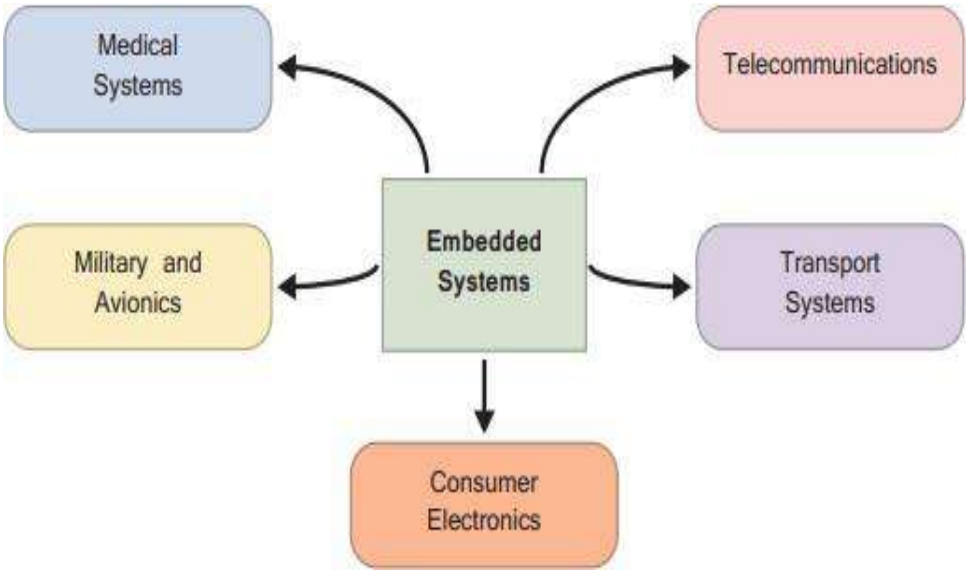


Figure 1.3: Embedded system application areas

Chapter 1: Embedded Systems based on the Xilinx Zynq All Programmable SoC

- **Embedded Systems in Automobiles and in telecommunications:**

  - Motor and cruise control system
  - Body or Engine safety
  - Entertainment and multimedia in car
  - E-Com and Mobile access
  - Robotics in assembly line
  - Wireless communication
  - Mobile computing and networking

- **Embedded Systems in Smart Cards, Missiles and Satellites:**

  - Security systems
  - Telephone and banking
  - Defense and aerospace
  - Communication

- **Embedded Systems in Peripherals & Computer Networking:**

  - Displays and Monitors
  - Networking Systems
  - Image Processing
  - Network cards and printers

- **Embedded Systems in Consumer Electronics:**

  - Digital Cameras
  - Set top Boxes
  - High Definition TVs
  - DVDs

I.5.Embedded System Characteristics:

I.5.1. Limiting memory:

On an Embedded System, memory is at a premium. Some chips, particularly embedded VLSI (**Very large-scale integration**) chips, and low-end microprocessors may only have a small amount of RAM "on board" (built directly into the chip), and therefore their memory is not expandable. Other embedded systems have a certain amount of memory, and have no means to expand. In addition to RAM, some embedded systems have some non-volatile memory, in the form of miniature magnetic disks, FLASH memory expansions, or even various 3rd-party memory card expansions. Keep in mind however, that a memory upgrade on an embedded system may cost more than the entire system itself. An embedded systems programmer, therefore, needs to be very much aware of the memory available, and the memory needed to complete a task.

I.5.2. Energy consumption:

Energy consumption has become a critical factor for such systems, especially after the explosion of the portable devices market. This situation has resulted to the requirement for accurate estimation of the energy consumed by the system for executing certain tasks. Although appropriate methods have been developed and certain models have been provided for the estimation of the energy consumption of embedded processors [5,6].

I.5.3. Autonomy:

Autonomy may be defined as: The ability to operate without human intervention. Autonomy is a discernible trend in the evolution of many types of machines. For example, in a modern car, an increasing number of functions like changing gears, maintaining speed or distance to the vehicle ahead, holding lanes et cetera now require no human intervention. Similarly in aviation, the evolving functionality of autopilots requires decreasing human intervention in the operational phases of a flight. Embedded systems enable autonomy because they make it easy to perform computations and take decisions based on those computations.

I.5.4. Mobility:

In our world today the mobility becomes the essential component of embedded system due to the huge demanding of mobility to adapt with the evolution of battery and wireless communication such as smart phones and laptops.

I.5.5. Communication:

Communication sector has recently been developing very rapidly on individual and mass communications, industrial and commerce field, advertisement and education sectors. As the electronic communication is developing and becoming widespread with new technologies, the need for qualified workforce has increased. Electronic Communication Program aims to train highly qualified, intelligent intermediate workforce who can think analytically, which is needed on communication technologies in social and business life.

I.5.6. Security:

Many embedded system devices including personal computers, cell phones, network routers, smart cards, and networked sensors to name a few need to access, store, manipulate, or communicate sensitive information, making security a serious concern in their design. Security has been the subject of intensive research in the context of general-purpose computing and communications systems.

I.5.7. Real-time:

Real-time and networked embedded systems are important bidirectional bridges between the physical and the information worlds, Real-time programs must guarantee response within specified time constraints, often referred to as "deadlines".[7]

I.6. Generic Embedded System Architecture: [1]

In general, the architecture of an embedded system follows the architecture outlined in **Figure 1.4**.

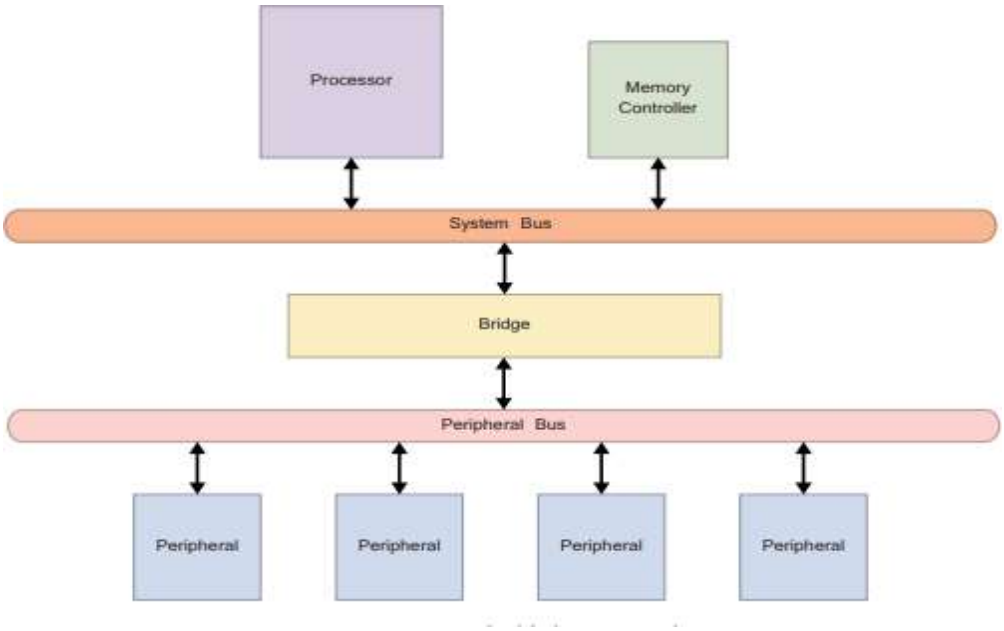


Figure 1.4: Generic embedded system architecture

The individual components in Figure 1.4 are defined as follows:

- **Processor** — This is the 'brains' of the system. It is programmed to perform the tasks specific to the application of the embedded system.
- **Memory Controller** — Memory controllers manage the reading and writing of data to and from main memory in an embedded system. Memory controllers reside in on-chip soft cores, providing an interface between the system memory and all other parts of the system.
- **Peripherals** — These are the components around the central processing unit. Peripherals can be implemented as individual integrated circuits, be contained on-chip with the processor, or may reside in an area of programmable logic such as an FPGA.

- **System Bus** — In an embedded system with multiple buses, the system bus connects the processor, memory controller and other high-speed devices together. As such, it is the bus with the greatest bandwidth in the system.

- **Peripheral Bus** — The second bus creates two separate sections of the system, allowing two arbiters to control and manage communications across the two buses. This allows devices on the peripheral bus to communicate with each other, even when a high-priority processor-memory transaction is taking place on the system bus.

I.6.1. Processors:

A processor is the main controlling unit within an embedded system. It controls and orchestrates the system, supports software and co-ordinates exchanges with peripheral components. In embedded systems where an operating system is used for the governing of the system, it will run on the processor.

There are a variety of processors which can be used in an embedded system, as outlined below:

- **Microprocessor** — A microprocessor is a single chip integrated circuit which contains the complete central processing unit, and nothing else. In order to make a microprocessor functional, external memory in the form of RAM and ROM, and other peripherals must be added. The CPU in a PC is a good example of a microprocessor.

- **Microcontroller** — A microcontroller contains an entire computing system on a single chip. Unlike a microprocessor, a microcontroller comprises of a CPU with a fixed amount of RAM/ROM and peripherals all within a single Integrated Circuit (IC).

- **Digital Signal Processor** — A digital signal processor is a processor which has been designed specifically for the task of digital signal processing, with an instruction set that is targeted accordingly. DSPs are designed to perform fast arithmetic operations, and are capable of performing a multiply-accumulate operation in a single clock cycle. This makes a

DSP very efficient, both in terms of performance and power consumption, when used for specific audio/video tasks, but very poor when used for other tasks due to the restricted instruction set.

- **Embedded Processor** — An embedded processor is one which is physically embedded within the programmable fabric of an FPGA device. Embedded processors come in two varieties — hard and soft processors. Hard processors are those which are built from dedicated silicon outwith the general-purpose logic of the FPGA device, whereas a soft processor is built using the general-purpose logic of the FPGA. A soft processor must be synthesised to fit into the FPGA fabric. In the case of all embedded processors, both hard and soft, the local memory, bus interconnects, memory controllers and internal peripherals must be realised in the FPGA general-purpose logic.

I.6.2. Co-processors:

A coprocessor is a processing core that supplements the functionality of the primary processor, and is optimised for a single, specific task. By off-loading computations from the main processor to one or more co-processing units, the overall system performance can be accelerated.
Whereas the main processor may be used for a variety of different tasks, co-processors are generally used to perform dedicated tasks. Examples of tasks which may be performed on dedicated co-processors are:

- High-speed arithmetic
- Image and video processing
- Digital signal processing
- Data encryption

With reference to FPGA based embedded systems, the programmable logic provides a perfect platform in which to create co-processing cores due to the ability to perform parallel execution. This means that complex tasks that would require a large number of sequential CPU clock cycles to compute can be executed far quicker in a PL based co-processor. These are known as soft

coprocessor cores. Other forms of acceleration can be implemented using dedicated hard processing cores that do not reside in the FPGA fabric. Collectively, this task off-loading processing is known as hardware acceleration.

I.6.3. Interrupts:

An interrupt is a signal which is generated to indicate to the processor that its attention is required. Interrupts can be generated by hardware processing units and peripherals, and also within the software itself. In hardware, an interrupt signal is an asynchronous signal that is generated by a processing unit to indicate the need for attention from the processor. In software, an interrupt is also an asynchronous event which indicated to the processor that a change in execution is needed. The process of polling the generated interrupts, however, is synchronous.

When the processor receives an interrupt it will halt the task that is currently being processed, and jump to the one which has requested attention. This is in contrast with the process of polling, which is the synchronous sampling of a device's status by the software program. Instead of having the processor constantly poll the I/O ports of a device to see if its attention is needed. the device will interrupt the processor.

Hardware interrupts can be further categorized into the following types:

- **Maskable Interrupts (IRQs)** — The trigger event of a masked interrupt is not always important. It is the task of the programmer to decide whether the event should cause the program to jump to the requested execution or not. Examples of devices which may use maskable interrupts include timers, comparators and ADCs.

- **Non-Maskable Interrupts (NMIs)** — These are interrupts which should never be ignored, and are therefore deemed much more important than maskable interrupts. Events which require NMIs include power-on, external reset (from a physical button) and serious device faults.

- **Inter-Processor Interrupts (IPIs)** — In multiple processor systems, one processor may need to interrupt the operation of another processor. In this situation, an IPI will be generated.

I.6.4. Buses:

A bus provides the interconnection by which processors interface with other processors and peripherals. Processors, memory controllers and peripherals connect to the bus via a standard bus interface. The particular bus interface will depend on the particular bus architecture that is in use, but at a basic level a bus consists of address, control (read/write requests and acknowledgements) and data signals. This is detailed in Figure 1.5 where the three main bus signals are highlighted:

- The **address line** transfers the memory addresses, or target port identification numbers, to I/O devices.
- The **control line** governs the control and timing signals of a system, synchronising operations and transmitting control signals such as interrupt requests and acknowledgements.
- The **data line** is responsible for the transfer of data.

Data transfer occurs in a bus cycle.

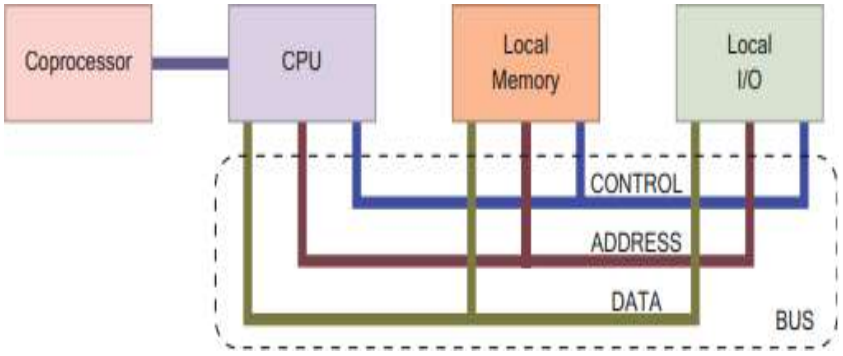


Figure 1.5: Bus comprised of control, address and data lines

Any coprocessors that are present in an embedded system are directly connected, or closely coupled, to the main processor via the processor bus. In a multi-processor system, individual processor subsystems are connected by the system bus.

Buses also contain a bus arbiter which controls access to the bus. Not all connected modules can request access to the bus, and they are categorized accordingly as bus masters and bus slaves.

I.6.4.1. System and Peripheral Buses:

In larger embedded system designs, it may be appropriate to use multiple buses to provide sufficient bandwidth for communication between all processing and peripheral cores. In such systems, the two main types of bus are the system bus and the peripheral bus.

- **System Bus:**

In an embedded system design, the system bus is the main method of providing communication between the various peripheral and processing cores. In smaller embedded system designs, the system bus may be the only bus that is present within the design. In larger, multiple-bus designs, the system bus will be the bus with the largest bandwidth that connects the memory controller and the processor, as well as any high-speed devices. All remaining peripherals, which do not require such high-speed access to the processor and memory controller, will be connected via the peripheral bus.

- **Peripheral Bus:**

In order to split the embedded system design into separate domains, a second bus — known as the peripheral bus — can be added. This can be for a number of reasons, such as making the distinction between low-speed and high-speed devices or to provide dedicated bandwidth for the communication between a group of peripheral cores. This allows a set of peripherals to communicate between themselves in parallel with any communication on the system bus, such as a processor-memory access.

- **Bus Bridge:**

In order to allow the communication between devices on separate buses, such as the processor (on the system bus) requesting data from a peripheral core (on the peripheral bus), a bridge between the two buses is required. A bridge is connected to both buses and communicates requests between them. On one bus the bridge is connected as a bus master, while having a slave connection on the other.

I.6.4.2. Bus Masters and Slaves:

The modules that connect to the buses in an embedded system can be split into two distinct categories:

- **Bus Masters** have the ability to request access to the bus and, as such, are responsible for initiating data transfers by driving the address and control signals. Bus cycles can be initiated, and other bus modules informed of the bus cycle type.
- **Bus Slaves** do not possess the ability to initiate bus cycles, and instead only monitor bus activity. Signals on the address and control lines are decoded, and when addressed, can either place, or accept, data on/from the bus.

I.6.4.3. Bus Arbitration:

As a bus is shared amongst all devices in an embedded system, there is a requirement to determine which bus master device is permitted to use the bus at any one time. The method for determining this access is called bus arbitration. If more than one master device requests access to the bus at the same time, it is the job of the bus arbiter to decide which device should be granted access first. In simple terms, the requesting master device with the highest priority (lowest assigned number) will be granted access to the bus first, provided a bus cycle is not already in progress. Lower priority masters (higher assigned numbers) will be placed in a waiting queue, and processed upon completion of the higher priority requests.

I.6.4.4. Memory Access:

The way in which memory controllers are accessed in an embedded system can have a huge effect on overall performance. Even if a very efficient type of memory and memory controller is used, the system performance could suffer from poor memory access control. It is important that the system is structured, and accessed, in a way which will maximize the memory bandwidth whilst keeping the required resources to a minimum.

- **Programmable Input/Output (I/O):**

One way of controlling the movement of data between the memory controller and other peripherals is to route all data transfers via the processor. This method of memory transfer is known as programmable I/O, and allows the system to process memory transfers with a minimum amount of resources. This approach requires that the peripheral and the processor are situated on the same bus, and the processor acts as a central point of communication between all peripherals and the memory. If the number of memory transfer requests between peripherals and memory is high, the processor could spend a lot of time performing memory transfers and less time performing other computations.

Programmable I/O is an effective way of managing memory transactions, whilst using a minimum amount of resources, provided that the system implements most of the functionality in the programmable logic. If, however, the processor is required to perform a large number of other computations, other methods may be preferable.

- **Direct Memory Access (DMA):**

One way of reducing the burden on the processor is to use Direct Memory Access (DMA) to perform memory transfers. Using this approach, the processor issues a memory transfer request to the DMA controller, which will then perform the memory transaction. This allows the processor to perform other tasks while the DMA controller performs the transfer. In this situation, the DMA controller acts as both a bus master and a bus slave. As a master, the DMA controller communicates with the memory controller whilst also arbitrating for the bus. While acting as a slave, the DMA controller sets up memory transfers by responding to requests from bus masters (the processor, in most cases). In order to initiate the transaction, the DMA controller must be provided with the following information:

•**Source address** — the address where the data will be read from.

•**Destination address** — the location where the data should be written to.

•**Transfer length** — the number of bytes that should be transferred.

An overview of a **DMA memory transfer** is provided in **Figure 1.6**.

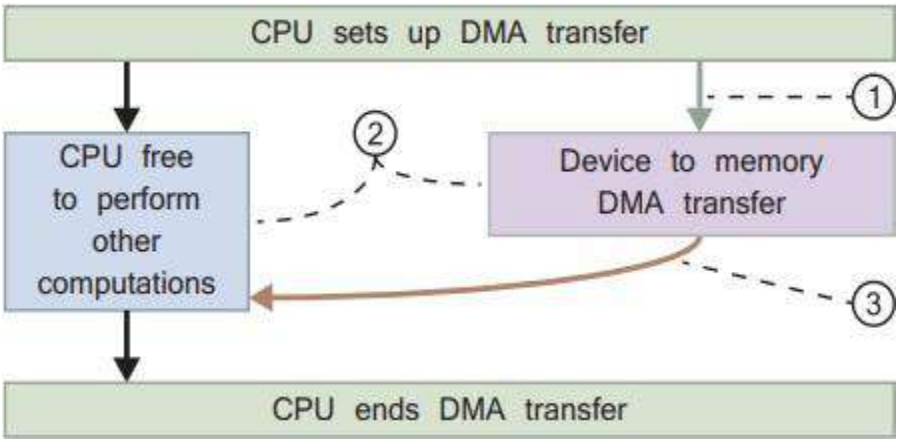


Figure 1.6: DMA memory transfer operation

the **DMA memory transfer** process is:

1. The processor sets up the device wishing to use the DMA to transfer data to memory by issuing a DMA command and disables all DMA interrupts.
2. The DMA controller transfers data from the peripheral device to memory leaving the CPU free to perform other computations.
3. Following completion of the data transfer, an interrupt is sent to the CPU to inform it that it can close the DMA transfer.

I.6.4.5. Bus Bandwidth:

Bus bandwidth is the total amount of data that can be transferred on a bus in a given unit of time. The value for bus bandwidth depends on two parameters:

- **Bus data width** — this is the number of physical lines over which the bus transfers data simultaneously. A bus with 32 individual data lines can transmit 32 bits of data simultaneously.
- **Bus frequency** — this is the speed at which the bus operates. This refers to the number of data bits which can be transmitted/received per second. This is defined in Hertz (Hz).

I.7. System-on-Chip with Zynq:[1]

As you aware, the concept has been around for a while; the implication is that a single silicon chip can be used to implement the functionality of an entire system, rather than several different physical chips being required. In the past, the term SoC has usually referred to an Application Specific Integrated Circuit (ASIC), which can include digital, analogue and radio frequency components, together with mixed signal blocks for implementing analogue-to-digital and digital-to-analogue converters (ADCs and DACs). Focussing on the digital aspect for a moment, an SoC can combine all aspects of a digital system: processing, high-speed logic, interfacing, memory and so on.

All of these functions might otherwise be realised using physically separate devices, and combined together into a system at the Printed Circuit Board (PCB) level. The SoC solution is lower cost, enables faster and more secure data transfers between the various system elements, has higher overall system speed, lower power consumption, smaller physical size, and better reliability. In fact there are a number of compelling reasons for preferring SoCs over discrete component equivalent systems! For a simple graphical comparison of the system-on-a-board and the system-on-chip, consider **Figure 1.7**.

The major disadvantages of ASIC-based SoCs are development time and cost, and lack of flexibility. The non-recurring engineering effort (and cost) of developing an ASIC are significant, making this type of SoC suitable only for high-volume markets where there is no requirement for future upgrades. Representative examples of ASIC-based SoCs are the integrated processors found in PCs, tablets, and Smartphone these typically comprise at least two processor cores, memory, graphics, interfacing, and other functions, and are manufactured in high volume for products with a limited lifetime.

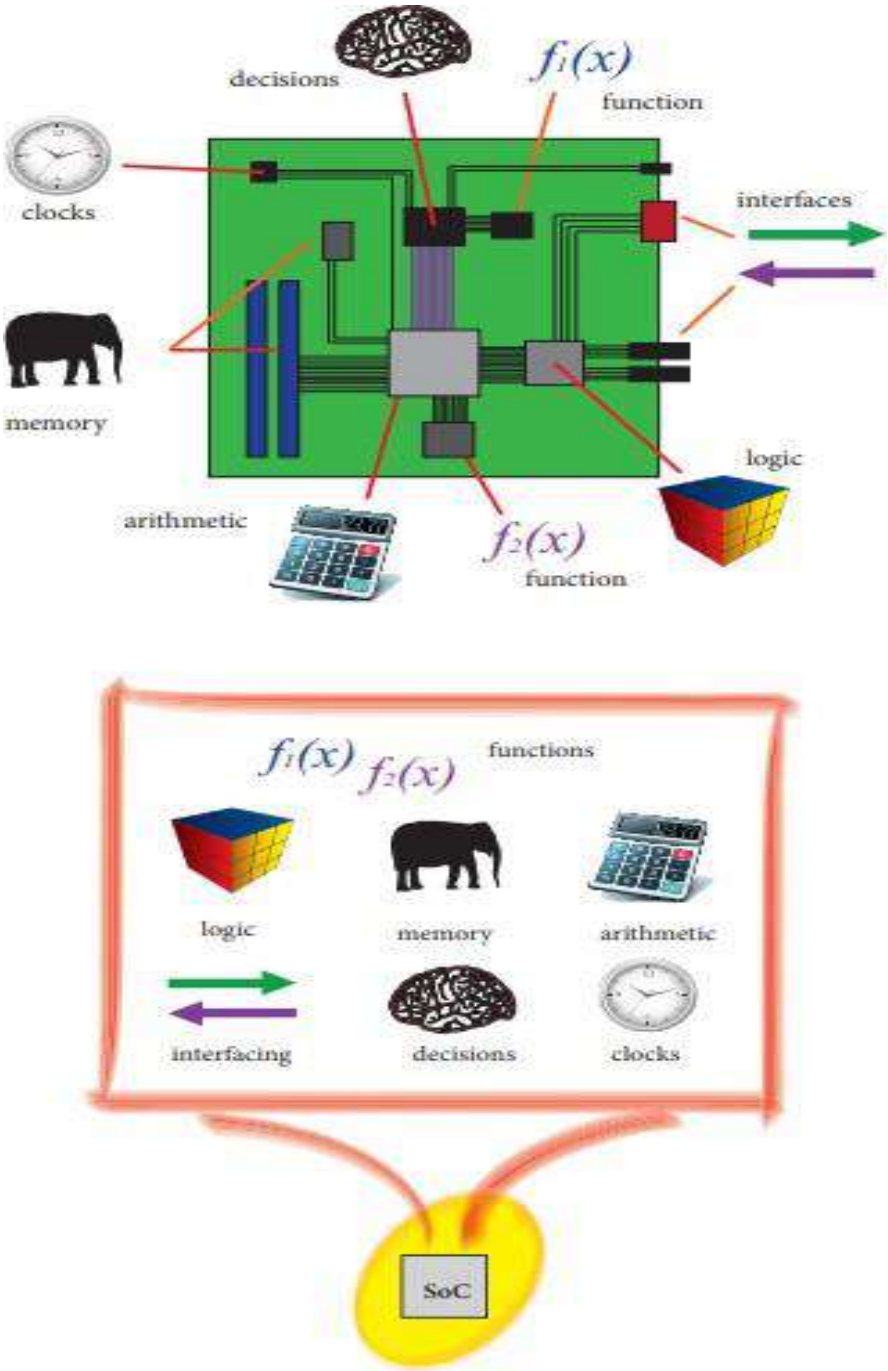


Figure 1.7: Comparison of System-on-a-Board (top) and System-on-Chip (bottom)

The limitations of ASIC SoCs render them incompatible with a significant number of applications, particularly where fast time-to-market, flexibility, and upgrade-ability are of key importance. They also constitute a poor solution for low or medium volume markets. There is a clear need for a more flexible solution, and this is what motivates the System-on-Progammable-Chip, a specific flavor of SoC implemented on aprogra-mmable, reconfigurable device. The natural solution has long been the FPGA.

FPGAs are inherently flexible devices that can be configured to implement any arbitrary system, including embedded processors if needed. FPGAs can also be reconfigured as often as desired, thus offering a more fundamentally flexible platform than ASICs for implementing SoCs. There is virtually no risk in deploying an FPGA in applications where system upgrades are required.

Now, Zynq provides an even more ideal platform for implementing flexible SoCs: Xilinx markets the device as an 'All-Programmable SoC' (APSoC), which perfectly captures its capabilities. Figure 1.8 introduce a high level model of its architecture. Note that Zynq comprises two main parts: a Processing System (PS) formed around a dual-core ARM Cortex-A9 processor, and Programmable Logic (PL), which is equivalent to that of an FPGA. It also features integrated memory, a variety of peripherals, and high-speed communications interfaces.

The PL section is ideal for implementing high-speed logic, arithmetic and data flow subsystems, while the PS supports software routines and/or operating systems, meaning that the overall functionality of any designed system can be appropriately partitioned between hardware and software. Links between the PL and PS are made using industry standard Advanced extensible Interface (AXI) connections.

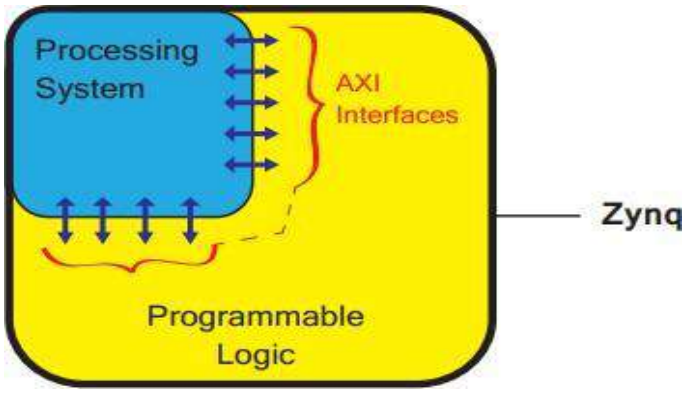


Figure 1.8: A simplified model of the Zynq architecture

I.8. Simple Anatomy of an Embedded SoC: [1]

At this early stage, it is useful to set out a basic model for the types of digital systems to be discussed in this work. These will be systems incorporating a processor, memories, and peripherals, along with buses connecting the various elements together. (This represents the **hardware system**, and here we consider a very simplified architecture) This model of the hardware system is shown in **Figure 1.9**.

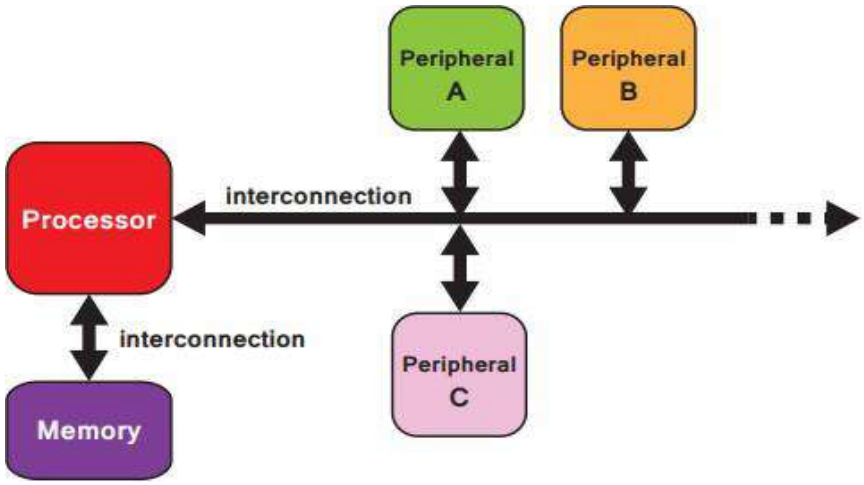


Figure 1.9: The hardware system architecture of an embedded SoC (simplified).

The processor can be regarded as the central element of the hardware system. The **software system** (a software 'stack') is run on the processor, comprising applications (usually based on an Operating System (OS)), and with a lower layer of software functionality for interfacing with the hardware system. Communication between system elements takes place via interconnections. These may be in the style of direct, point-to-point links, or buses serving multiple components. In the latter case, a protocol is required to manage access to the bus. Note that, although a single bus with connected peripherals is shown in Figure befor, a processor may serve several connected buses.

Peripherals are functional components residing away from the processor, and in general these perform one of three functions: coprocessors — elements that supplement the primary processor, usually optimised for a certain task cores for interacting with external interfaces, e.g. connecting to LEDs and switches, codecs, etc.; and additional memory elements. Later in the book, we will consider peripherals in more detail, but at this point it is useful to regard them as discrete functional blocks that can be designed, tested and integrated into a system, and also 'packaged' for later reuse.

Figure 1.10 provides a view of the hardware system shown in Figure 1.9, mapped to the Zynq device (depicted in Figure 1.8). The architectures of both have been substantially simplified, but the objective at this stage is to provide a high level clarification of how embedded SoCs map to Zynq devices. The PS has a fixed architecture and hosts the processor and system memory, whereas the PL is completely flexible, giving the designer a 'blank canvas' to create custom peripherals, or to reuse standard ones. The interconnections are implemented via AXI interfaces linking the PS and PL.

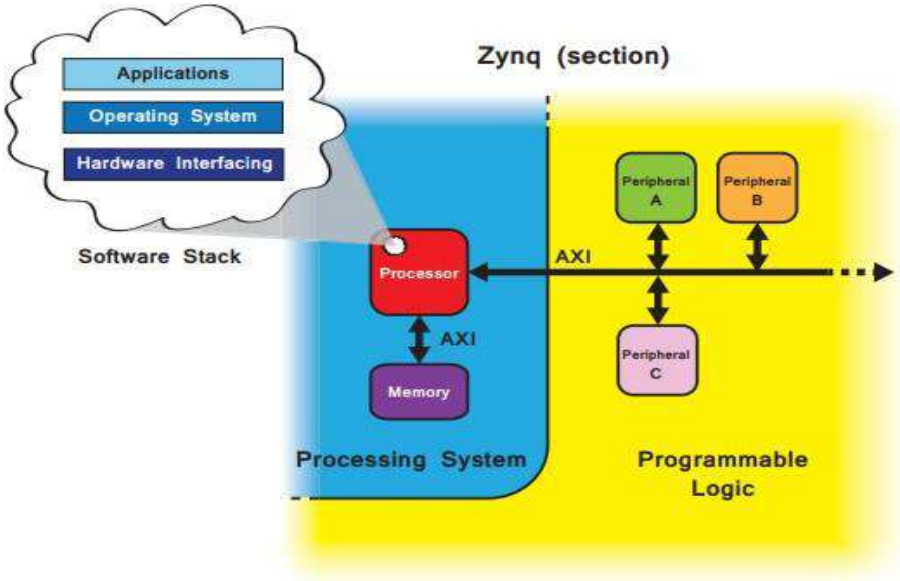


Figure 1.10: Relationship of the software system, hardware system, and Zynq architecture.

The software system can also be seen on the left-hand side of Figure 1.10. Software is hosted on the processor, which here is the ARM Cortex-A9, residing within the Zynq PS. It comprises a hierarchy of software elements.

I.9. Design Reuse: [1]

The development of a complete embedded system is a significant design task, and there are particular advantages to undertaking the design on a platform such as an FPGA or Zynq device, which make the process more straightforward. The underlying PL hardware is structured, and its performance characteristics are well known and integrated into the software development tools. Moreover, given this stable, common development platform, there is huge scope for design reuse. Intellectual Property (IP) functional blocks — corresponding to the peripheral components seen in **Figure 1.10** — can be sourced from Xilinx libraries (provided with the design tools), reused from previous projects, or brought in from third parties or open source repositories, before being integrated together to form the system design.

Zynq is an SoC, and a wide variety of standard IP is available, meaning that there is no need to redesign these components. By raising the abstraction level in this way, and reusing components in the form of pre-tested and verified IP, development can be accelerated and costs can be lowered.

Given its importance to the SoC design philosophy, one of the key themes of this work will be design reuse. We will consider the various sources of IP, including Xilinx libraries, mechanisms for generating one's own IP, and sources of third party IP.

I.10. Raising the Abstraction Level:[1]

A recurring trend in various software and hardware design processes is that of raising the level of abstraction. The motivation is clear: if the designer can effectively create systems with a lower requirement for explicit design input, while also supporting robust test processes, then there is great potential to accelerate the design process.

In terms of FPGA and Zynq design, advances in High Level Synthesis (HLS) mean that designers can create system components by specifying them using less detail than traditional, Register Transfer Level (RTL) methods, and instead rely upon the design tools to infer logic and optimize where possible, in accordance with user-supplied direction. Naturally this places a degree of trust in the development tools, which must be robust and produce repeatable and reliable designs. To meet this need, Xilinx has introduced the Vivado HLS tool, a high level synthesis development tool which specifically targets Xilinx devices.

I.11. SoC Design Flow: [1]

A multitude of different models have been proposed for the SoC design flow with varying levels of complexity, but initially we aim to define the design flow for SoC development (as applied to Zynq) in very simple terms. The basic stages are shown in Figure 1.11.

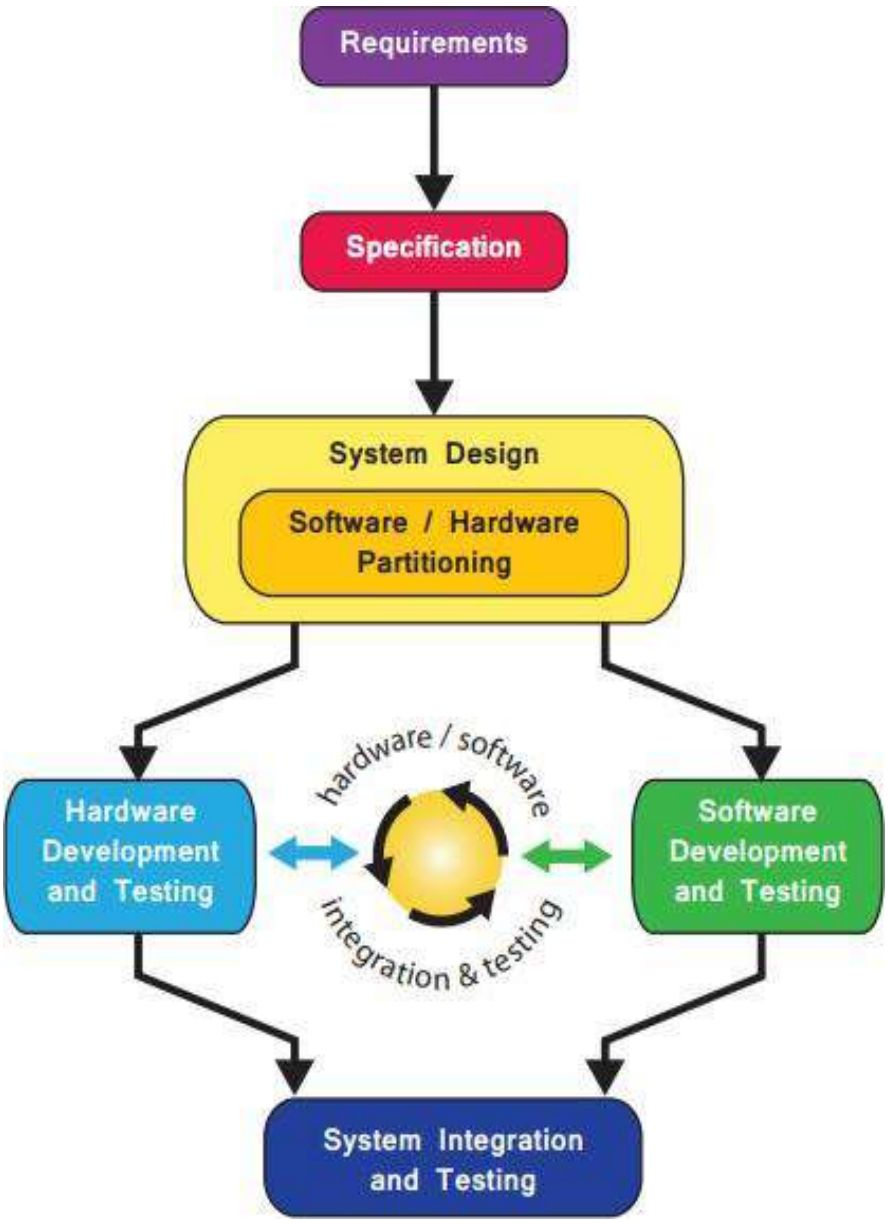


Figure 1.11: basic model of the design flow for Zynq SoC

Naturally, as in any design project, the first stage is to define the desired behaviours of the system, i.e. to create an appropriate specification from a set of requirements. This is depicted as the starting point at the top of the diagram, and it forms the basis of the system design that is subsequently developed.

As mentioned earlier, the Zynq architecture combines an ARM processor (for software elements of the designed system) with FPGA fabric (predominantly for hardware elements of the system, although additional processors can also be implemented here too, if desired). A key element of the system design stage, which comes next, is therefore to partition the intended functionality appropriately between software and hardware, and to define the interfaces between the two sections. Of course, it is possible that this partitioning will subsequently be adjusted as the designers iterate the system towards completion.

Having partitioned the system, software and hardware development can then progress in parallel, to a large extent. In terms of hardware development, the task is to identify the necessary functional blocks to achieve the design, and to thereafter assemble them through some combination of design reuse and new IP development, and make appropriate connections between the blocks. Similarly, the software aspect of the project can be realised through developing custom code or by reusing pre-existing software. Verification of both software and hardware will be required, and this forms an integral and important part of the process.

I.12. Conclusion:

In this chapter, the concept of an embedded system has been introduced and a generic embedded system architecture has been explored. The role of processors within embedded systems has been discussed, and some of the processing features, such as processor cache and execution cycles have been covered. The function of coprocessors was also introduced as well as the use of software/hardware interrupts.

Communication between all components within an embedded system is reliant on the inclusion of a bus system, and their functionality has been discussed in this chapter. The requirement for multiple bus systems was introduced, and the distinction between bus master and slave devices was reviewed. Bus arbitration and memory access techniques were summarized, and the importance of bus bandwidth was discussed.

Chapter 2

Embedded Vision

II.1. Introduction:

Embedded vision is the integration of computer vision in machines that use algorithms to decode meaning from observing pixel patterns in images or video. The computer vision field is developing rapidly, along with advances in silicon and, more recently, purpose designed embedded vision processors. Most commonly defined, an embedded vision system is any microprocessor-based system with image sensor functionality that isn't a standard personal computer. Tablets and smart phones fall into this category, as well as more unusual devices such as advanced medical diagnosis instruments and robots with object recognition capabilities. So, to put it simply, embedded vision refers to machines that understand their environment through visual means [7].

II.2. Embedded Vision architectures [8]:

Powerful, highly compact processing capability is the true enabler of embedded vision – small processing platforms can now do the work that traditionally took large, bulky computers. For embedded vision systems, processing components may take on several different configurations, including system on chip (SoC), system on module (SoM), single board computer (SBC), and fully custom designs.

SoC is a processing architecture where the central processing unit (CPU), graphics processing unit (GPU), interface controller and more are all located on a single chip. The development of this technology was driven predominantly by large mobile device manufacturers, as the cost to develop SoCs can be millions or even billions of dollars.

SoMs actually contain SoCs but also include RAM, power management and bus systems along with the image processing power of the SoCs. SoMs are most often used along with a carrier board with plug connectors for power or network connections. These systems help companies overcome the high costs of SoC development while providing great flexibility for developers to tweak the system to their needs.

SBCs are essentially SoMs that are integrated onto a single circuit board with a series of connectors for peripheral devices. They are SoMs and carrier boards in one – this comes with cost

and performance tradeoffs. SBCs have low initial costs and are easy to implement, but given their pre-integrated nature, they do not have much flexibility to adapt to specific applications.

Fully custom processing designs are not common – they are basically SBCs custom-built for one specific application. They typically arise when a company determines the cheapest option for their embedded vision application is to develop their own SBC. Custom embedded processing architectures are cost-effective as there are no unnecessary components, but since they are built for one specific purpose, they provide little to no flexibility in imaging functions.

There are two main types of processing architectures found in embedded vision systems. Even within each of these, there is variation in functionality between different manufacturers' proprietary technology. Despite the diversity of architectures, compact image processing forms the foundation of embedded vision technology.

II.3. Embedded Vision Components [8]:

Embedded vision is an exciting new technology with incredible potential in a wide range of industries, only recently made possible through the miniaturization of vision and processing components. Broadly speaking, an embedded vision system combines image capture and image processing capabilities into one system. However, embedded vision systems come in many different configurations.

At the most basic level, an embedded vision system includes a sensor or camera module, a standard interface and some form of processing architecture. There are a variety of options regarding each of these components for any embedded vision system – choosing the right one depends on understanding design, performance and cost tradeoffs.

To learn how embedded vision systems function, it is important to take a look at how processing architectures, types of processors, standard interfaces, and vision and imaging modules work to create a whole embedded system.

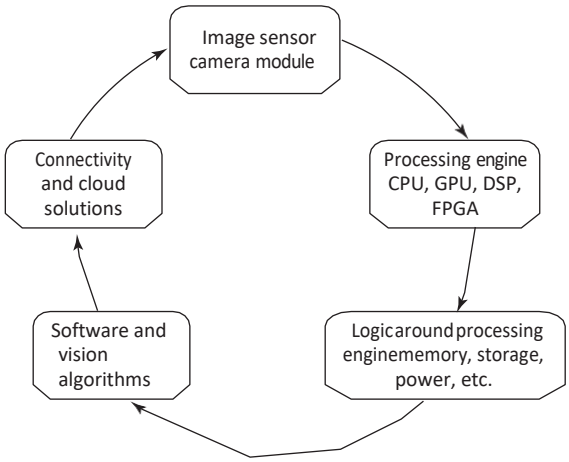


Figure 2.1: Embedded vision block diagram [1]

II.3.1. Sensors [9]:

Many of the choices for how digital image sensors work and much of the terminology used are historical. Their first applications were to replace analog sensors, that is, image sensors with an analog output signal. To understand today's sensors, a brief look at these historical choices helps.

II.3.1.1. Sensor History:

The history goes back to 1897 when Ferdinand Braun, a German physicist, invented what came to be known as the cathode ray tube (CRT, Braun tube, "Braunsche Ro¨hre").

Various methods for sensing video images electronically have been developed since the 1920s. In the 1960s, the vidicon cathode ray tube became the standard in television recording up through the 1990s. This tube is also built on the principle of CRTs, very similar to television sets, only that the electron beam measures the amounts of photons that hit the photoconductive layer in the image plane instead of lighting up a phosphorous emissive layer as in a TV set.

The readout order of lines in vidicon tubes is synchronized with the transmission signal (NTSC, PAL, or SECAM) and the display scan order of TV tubes. For example, the NTSC signal has 525 scan lines of which 486 are visible. A frame is split into two fields, the first field containing all

even lines and the second field all odd lines. The reasons for interlaced video transmission and display were band- width limitations and picture quality, which depended on light-emissive phosphors. Within a line, pixels are transmitted sequentially from left to right. Synchronization pulses (low voltage levels) are inserted after every line and after every field.

Photoelectric detectors in digital image sensors work surprisingly similar. They also exploit the photo effect, which turns a photon into an electric charge. Upon striking a photodetector, a photon excites an electron that then jumps from one semiconductor band to another, making a hole in one and adding a surplus electron to the other. For charge measurement, the hole is filled and the electron is pulled off by a current that can be measured in a voltage differential. Called the photodiode, photo detector, or photoactive region, these components can be implemented in MOS technology or in TFT technology.

II.3.1.2. The Charge-Coupled Device:

The charge-coupled device (CCD) was invented around 1970. As shown in Fig.2.2, a CCD transfers the acquired analog charge (electrons) from one photodetector through other photodetectors across the chip. This operation is called a "shift," making CCDs a type of shift register. A CCD uses a MOS photodetector for photon acquisition and for readout tasks. Note that the charge is transferred without creating a voltage. Only in the readout stage does the charge get converted into a voltage.

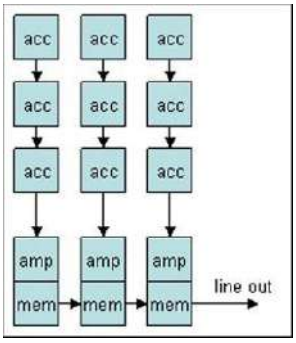


Figure 2.2: Charge readout from a CCD, a shift register.

There are three types of readout architectures:

1_ Full-frame devices consist of large charge accumulators and minimal additional circuitry around every pixel. They require a mechanical shutter because the entire sensor area is exposed to the light and would continue to accumulate charge as the image is transferred out.

2_ Frame transfer devices have half the CCD covered with an opaque material. The electronic shutter moves the charge "under cover" and slowly reads it out. Full-frame and frame transfer devices have the disadvantages that the charge needs to be shifted down the entire sensor array. Pixels in the first rows hence incur a larger amount of readout noise. To obtain interlaced output from a frame device, every other row must be discarded before data serialization. This gives these devices different characteristics than vidicon tubes of comparable specs, namely shorter integration time and reduced light sensitivity.

3_ Interline transfer devices have storage circuitry at every other line or column, avoiding this large shift noise accumulation and allowing for fast readout. Interline transfer devices can be read out in arbitrary row order, for example, in interlaced order. This makes them particularly well suited for analog output.

The charge accumulators are depleted on readout. Trace remaining charge is considered reset noise for a CCD. In systems without a mechanical shutter, charge accumulates in the photodetectors during the readout stage. To achieve a shorter exposure time, this charge needs to be depleted before exposure starts. This is accomplished by initiating a partial readout that depletes the charge but does not process the measured quantity.

A disadvantage of CCD sensors is that the photonic element can overflow: too bright a light source causes charge to spill into neighboring accumulators which, in turn, causes bright blobs called blooming or, during readout, vertical lines called smear.

Figure.2.3. shows the pixel flow from photon to digital signal. This is a generalized pipeline, applicable to both CCD and CMOS sensors with the noted differences.

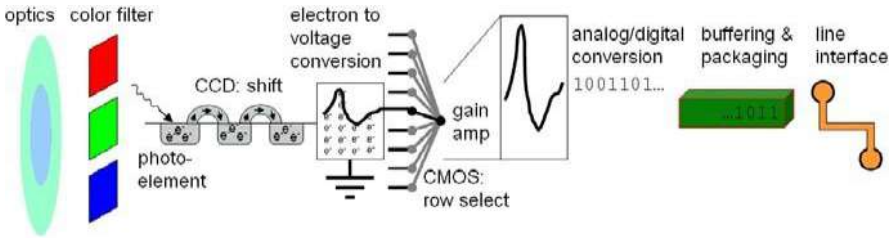


Figure 2.3: Overview of pixel acquisition in a sensor.

II.3.1.3. CMOS Sensors:

The so-called active pixel sensor (APS) dedicates circuitry to every pixel to convert the electric charge into voltage. This type of image sensor is typically manufactured with CMOS transistors, hence its more common name "CMOS sensor." Every pixel has its own amplifier, typically three or more transistors (see Fig. 1.3). Therefore, an important issue with CMOS imagers is the pixel fill factor—that is, how much of the space that one pixel takes up is dedicated to photon acquisition and how much is spent on related hardware such as charge conversion, amplification, and readout circuitry. Good fill factors for CMOS sensors are 50% or above, but they can be much lower. A CCD has close to a 100% fill factor. Multilayered sensor arrays, called "stacked dies" or 3D VLSI as opposed to planar VLSI technology, can offset some of these disadvantages and achieve higher fill factors for CMOS sensors.

CMOS sensors typically consume less power than a CCD, have less image lag, and can be manufactured on cheaper and more available semiconductor fabrication lines. Unlike CCDs, CMOS sensors can combine both the image sensor function and image processing functions within the same integrated circuit. CMOS sensors also permit readout in an arbitrary fashion, some even provide for per-pixel access, but most support windowing, meaning readout of an arbitrary rectangular region. This capability can be used to achieve higher frame rates than full-frame readout. Naturally, a frame rate of, for example, 500 Hz reduces the exposure time to 2 ms at best, hence limiting the amounts of photons that can be acquired in such a short time.

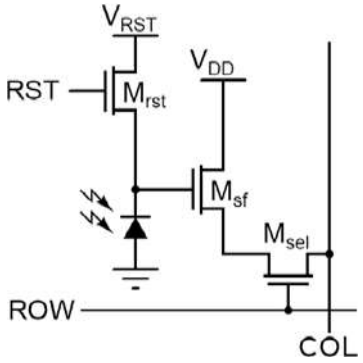


Figure 2.4: Schematic of one pixel of a CMOS active pixel sensor.

Dark scenes therefore require a higher gain, which introduces noise in the readout process. Similar to a memory chip, there is a crisscross grid of readout lines running across the image array. The row-select lines select one of the array rows. The column lines transport the pixels' activation down to the readout electronics, typically an amplifier followed by a multiplexer. In contrast to CCDs, the activation in a CMOS is voltage rather than charge, and it is transported to the readout directly, without requiring a shift through all pixels in the same row below the current pixel.

II.3.1.4. Readout and Control:

Image sensor control and pixel readout is facilitated with on-chip logic. This circuitry drives the reset signal lines, initiates the charge transfer, chooses one row- select line at a time, sets the gain level, packages the outgoing pixel data and responds to input control commands. These commands are specified in protocols and sent over a control bus to the sensor (see the next section). The sensor reacts by setting control registers accordingly. Its logic reads out the registers regularly, for example, during the reset cycle, so that the next frame that is to be captured con- forms to parameters such as exposure, windowing, and gain. Permissible times for the arrival of control packets and their synchronization with frame readout differ between the various manufacturers and from chip to chip.

Many sensors, particularly the newer CMOS sensors, permit extensive readout customization, including selection of a contiguous subset of rows, windowing (select rows and columns) and image flipping (reverse order of rows and/orcolumns). Note that when reading out a partial image from a sensor with color filters on the photoelements pixels, two or more lines must be read out to

accurately reconstruct the color information for one line since (e.g., for a Bayer pattern) the red and blue pixel filters are applied on alternating lines.

II.3.2. Processing Unit [8]:

An embedded vision system's image processing capability are determined not just by overall processing architecture, but by the type of processor being used. While there are several different types of image processors, the main two types are graphics processing units (GPUs) and field programmable gate arrays (FPGAs).

GPUs are widely deployed in embedded vision systems because of their ability to deliver large amounts of parallel computing potential, especially to accelerate the portions of processing that deal with pixel data. General purpose GPUs are one of the most common forms, as they're built to meet the power constraints for a diverse range of applications.

All GPUs leverage software for imaging algorithms, which comes with advantages and disadvantages. With GPUs, tweaking or altogether changing imaging functions is as easy as changing the system's software instructions. This provides great flexibility for one system to perform multiple imaging functions. On the other hand, software is always slower than hardware – latency is higher by multiple degrees of magnitude in GPUs.

FPGAs have been gaining favor in recent years for a few reasons. First and foremost, their main advantage is their low latency levels – they are hardware representations of algorithms, making them much faster. FPGAs are also far more energy efficient despite the fact that they provide the highest processing potential with computing performance in the trillions of multiply-accumulates/sec (tera-MACs). FPGAs can simultaneously accelerate multiple portions of a computer vision pipeline – GPUs cannot.

The main drawback of FPGA processors is that they lack the flexibility of GPUs. It takes far longer to prepare a FPGA for another imaging task than tweaking an algorithm in GPUs. However, this has been changing in the past few years as developers have been introducing open source computer vision libraries with pre-configured interface blocks for quicker and easier changes.

While there are certainly other types of image processors, including application-specific integrated circuits (ASICs) and digital signal processors (DSPs), GPUs and FPGAs are two of the most popular. Both have their advantages and disadvantages depending on how they will be deployed in an embedded vision application.

II.3.3. Embedded vision standard interfaces [8]:

The connection between the sensor and image processor is important – it can be a bottleneck for performance if chosen incorrectly. As with other embedded vision components, there are several possibilities for embedded interfaces.

USB 3.0 is often a simple plug and play solution for some embedded vision systems, just as they are for traditional machine vision systems. However, they rarely feature a compact design, which immediately rules them out as a solution for many embedded vision systems designs that have small space requirements.

Flat flex cables, such as those that follow the MIPI CSI-2 standard from MIPI Alliance, are a more powerful, compact solution for camera integration. However, these types of cables necessitate each individual component at a board-level be adapted to the flat flex cable's capabilities. This can get costly, and flat flex cables often cannot exceed 20 centimeters in length – two major drawbacks for certain applications.

One of the more reliable interface solutions for embedded vision has been low voltage differential signaling (LVDS) where cables can be several meters long. LVDS cables are typically used with FPGA processors for integration and adjustment of system-specific read-routines. LVDS cables do not use standardized plugs or data transfer protocols, meaning integration costs will be high unless there are openly documented transfer protocols.

The transfer of data between a sensor and processor is a vital process in any embedded vision system. Once again, there are several potential interfaces an embedded system could use, and each has their cost and performance tradeoffs.

II.4. Image Operations [9]:

This section addresses the question of how well certain operations are suited to embedded implementation.

An algorithm's speed performance on a DSP or FPGA often differs vastly from its performance on a general-purpose processor (GPP/CPU). The following algorithm characteristics lend themselves to execution in special hardware:

- data streams and sequential data access, as opposed to random access.
- multiple, largely independent streams of data.
- high data rates with few instructions per datum.
- A fixed data packet size, or a size below a tight bound.
- stream computations that can be broken up into pipeline stages, that is, the same (possible parameterized) set of computations can be applied to lots of data.
- the operations require only fixed-precision values (integer or fixed-point fractions)
- algorithms are parallelizable at instruction and module level, that is, little between- stream communication is necessary.

Desirable operations to be executed on high-performance hardware in the embedded system are typically lower-level functions that pre-process the image for further high-level analysis and might include:

- convolution/cross correlation, 2D and separable filters (Gaussian, wavelets etc.),
- subsampling at regular intervals,
- Canny edge detection
- sum of absolute differences (SAD) of an image region of interest (ROI) with a /stored patch
- edge orientation and histogram calculation, integral

- image calculation,
- online calculation of global or ROI statistics: min, max, mean, or variance, higher-order moments,
- online Bayer sample pattern conversion into RGB, HSV, GRAY, etc. (depending on camera color pattern this requires a one-line buffer).
- online color space conversion (RGB2GRAY, RGB2HSV).

An "online" algorithm is able to calculate its result even though only a subset of the data is available at any one time, commonly referred to as a window into the data that is slid over the entire data. This is easily possible for simple operations such as the minimum or mean value, but only possible through approximation for median calculation, for example.

Table 2.1 shows common operations and the complexity of their data access pat- terns. For the best speed and memory performance, the algorithm accesses only a single pixel at any one time, and the sequence of pixel accesses is known before- hand. On the opposite end of the spectrum are algorithms that require large amounts of data for the calculation of one result and their data access pattern is not known before hand, for example, when it is dynamically dependent on the result of previous calculations. Hence, the speed and memory complexity of the methods mentioned in Table 2.1 increases from top to bottom. Some algorithms create a data space that is accessed in subsequent passes, rather than accessing the image space multiple times.

Pixel processing: a single pass over the image is sufficient, and a pixel's new value is only determined by exactly one source pixel value. • lookup-tables (LUT) • Gray level or color thresholding • color space conversion • brightness correction • arithmetic operations • logic operations
N-pass: multiple passes over the image and data space are necessary; however, only one source pixel value determines the new pixel value. • count, min, max, avg, stddev • histogram equalization or histogram matching • Hough transforms
Fixed-size block access: the values of pixels in an area of known and fixed size determines the output value. • morphology • convolution, filtering • pyramids • wavelets • KLT (Lucas-Kanade) feature tracking
Data-independent, global access: multiple source pixel values from pixels all over the image determine the outcome of the operation. The access pattern is known, however. • Viola-Jones • warping or remapping for distortion correction
Data-dependent, random access: multiple source pixel values from pixels all over the image determine the outcome of the operation. The access pattern is determined by the values read from the source pixels. • naive flood fill • contour finding

Table 2.1 Data Interdependency in Image Space.

II.5 Applications for Embedded vision [7]:

II.5.1. Industrial Applications:

Vision-processing-based products have established themselves in a number of industrial applications. The most prominent one being factory automation where the application is commonly referred to as machine vision. It identifies the primary factory automation sectors as:

- Automotive—motor vehicle and related component manufacturing.
- Packaging—packaging machinery, packaging manufacturers, and dedicated packaging companies not aligned to any one industry.
- Robotics—guidance of robots and robotic machines.
- Semiconductors and Electronics—semiconductor machinery makers, semiconductor device manufacturers, electronic equipment manufacturing, and assembly facilities.

II.5.2. Medical Applications:

Embedded vision and video analysis have the potential to become the primary treatment tool in hospitals and clinics, and can increase the efficiency and accuracy of radiologists and clinicians. The high quality and definition of the output from scanners and X-ray machines makes them ideal for automatic analysis, be it for tumor and anomaly detection, or for monitoring changes over a period of time in dental offices or for cancer screening.

II.5.3. Automotive Applications:

Vision products in automotive applications can serve to enhance the driving experience by making us better and safer drivers through both driver and road monitoring. Driver monitoring applications use computer vision to ensure that driver remains alert and awake while operating the vehicle. These systems can monitor head movement and body language for indications that the driver is drowsy, thus posing a threat to others on the road. They can also monitor for driver distraction behaviors such as texting, eating, and so on, responding with a friendly reminder that encourages the driver to focus on the

road instead. An example of an emerging high-volume embedded vision application is automotive /safety systems based on vision which is shown in Figure 2.5.



Figure 2.5: Mobileye driver assistance system.

II.5.4. Security Applications:

Vision technology was first deployed for security applications, addressing the fundamental problem that the millions of security video cameras vastly outnumber the availability of humans to monitor them in real time. Moreover, studies have shown that the attention span of a human is only about 20 minutes when watching a camera feed. In most cases, video cameras historically only helped to review t

II.5.5. Consumer Applications:

The days of the traditional game controller may well be numbered, as sensors and video analytics enable detection of body movements and even facial expressions to become the interface between player and game. Consumer applications for embedded vision products have taken off in a big way since the launch of the first-generation Xbox Kinect (Figure 2.6), the fastest selling consumer electronic device in history. The PlayStation Move quickly followed suit and now software developers are rapidly developing titles that take advantage of the technology. Consumer applications for vision are not limited to the gaming industry, however. Remote control applications are already in use today, via vendors such as GestureTek and PrimeSense (for the Kinect, now owned by Apple), which eliminate the need for a physical remote control.

Furthermore, embedded vision can be used in other household applications such as set top boxes, televisions, PCs, smart phones and tablets, lighting, heating, and air conditioning to further

eliminate the need for physical controls.



Figure 2.6: Microsoft Kinect for Xbox 360, a gesture-based game controller.

II.6. Conclusion:

This chapter tried to shed light on the hardware components available for embedded vision systems. It presented several types of image sensors as well as their readout styles, speeds, and interface styles. The CMOS active pixel image array sensor, because of its region of interest readout and its flexibility and controllability was identified as a leading candidate for use in an embedded vision system, but CCDs have their place for systems with the highest demands on sensor sensitivity.

Several hardware storage and processing components (including DSPs, various system-on-a-chip combinations, FPGAs, GPUs, and memories) have been considered as building blocks from which a vision system might be realized.

Chapter 3
Real time video processing system realization

III.1. Introduction:

In our project, we used the Digilent Embedded Vision Bundle, Xilinx Vivado, Xilinx Vivado HLS, and Xilinx SDK to create a video processing system. The Digilent Embedded Vision bundle includes the Zybo Z7-20 Zynq-7000 ARM/FPGA SoC development board and the Pcam 5C 5- megapixel (MP) color camera module.
The Digilent ZYBO Z7 is the newest addition to the popular ZYBO line of RM/FPGA SoC Platforms. The ZYBO Z7 surrounds the Zynq with a rich set of multimedia and connectivity peripherals to create a formidable single-board computer, even before considering the flexibility and power added by the FPGA. Hardware designers and software developers can seamlessly integrate FPGA and CPU functionality. The ZYBO Z7's video-capable feature set includes a MIPI CSI-2 compatible Pcam connector, HDMI input, HDMI output, and high DDR3L bandwidth [10].

The Pcam 5C is an imaging module meant for use with FPGA development boards. The module is designed around the Omnivision OV5640 5MP color image sensor. This sensor includes various internal processing functions that can improve image quality, including automatic white balance, automatic black level calibration, and controls for adjusting saturation, hue, gamma and sharpness. Data is transferred over a dual-lane MIPI CSI-2 interface, which provides enough data bandwidth to support common video streaming formats such as 720p (at 60 frames per second) and 1080p (at 30 frames per second). The module is connected to the FPGA development board via a 15-pin flat- flexible cable (FFC) [10].

In this work, a Pcam 5C supplies real-time high definition video to the HDMI input of a Zybo Z7-20. The Zybo can then perform edge detection, inverted color or grayscale by configuring bottom two switches (sw1 and sw0) on the Zybo Z7-20. The processed signals are sent to the HDMI output of the Zybo Z7-20 for display on an HD monitor.

.

III.2. Zybo Z7 Board: [11]

The Zybo Z7 is a feature-rich, ready-to-use embedded software and digital circuit development board built around the Xilinx Zynq-7000 family. The Zynq family is based on the Xilinx All Programmable System-on-Chip (AP SoC) architecture, which tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic. The Zybo Z7 surrounds the Zynq with a rich set of multimedia and connectivity peripherals to create a formidable single-board computer, even before considering the flexibility and power added by the FPGA. The Zybo Z7's video-capable feature set, including a MIPI CSI-2 compatible Pcam connector, HDMI input, HDMI output, and high DDR3L bandwidth, was chosen to make it an affordable solution for the high end embedded vision applications that Xilinx FPGAs are popular for. Attaching additional hardware is made easy by the Zybo Z7's Pmod connectors, allowing access to Digilent's catalog of over 70 Pmod peripheral boards, including motor controllers, sensors, displays, and more.

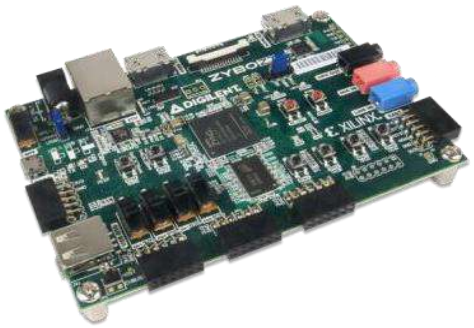


Figure 3.1 Zybo Z7 Board

ZYNQ Processor:

- 667 MHz dual-core Cortex-A9 processor
- DDR3L memory controller with 8 DMA channels and 4 High Performance AXI3 Slave ports
- High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
- Low-bandwidth peripheral controllers: SPI, UART, CAN, I2C
- Programmable from JTAG, Quad-SPI flash, and microSD card
- Programmable logic equivalent to Artix-7 FPGA

Memory:

- 1 GB DDR3L with 32-bit bus @ 1066 MHz
- 16 MB Quad-SPI Flash with factory programmed 128-bit random number and 48-bit globally unique EUI-48/64™ compatible identifier

USB and Ethernet:

- Gigabit Ethernet PHY
- USB-JTAG Programming circuitry
- USB-UART bridge
- USB 2.0 OTG PHY with host and device support

Switches, Pushbuttons, & LEDs:

- 6 push-buttons (2 processor connected)
- 4 slide switches
- 5 LEDs (1 processor connected)
- 2 RGB LEDs (1*)

Power:

- Powered from USB or any 5V external power source

Audio and Video:

- Pcam camera connector with MIPI CSI-2 support
- HDMI sink port (input) with/without* CEC
- HDMI source port (output) with CEC
- Audio codec with stereo headphone, stereo line-in, and microphone jacks

Expansion Connectors:

- 6 Pmod ports (5*)
- 8 Total Processor I/O
- 40 Total FPGA I/O (32*)
- 4 Analog capable 0- 1.0V differential pairs to XADC

III.3 Pcam 5C imaging module: [12]

III.3.1 Pcam 5C Module:

The Pcam 5C is an imaging module meant for use with FPGA development boards. The module is designed around the Omnivision OV5640 5 megapixel (MP) color image sensor. This sensor includes various internal processing functions that can improve image quality, including automatic white balance, automatic black level calibration, and controls for adjusting saturation, hue, gamma and sharpness. Data is transferred over a dual-lane MIPI CSI-2 interface, which provides enough data bandwidth to support common video streaming formats such as 1080p (at 30 frames per second) and 720p (at 60 frames per second). The module is connected to the FPGA development board via a 15-pin flat-flexible cable (FFC) that is pin compatible with the connector found on the popular Raspberry Pi development board. The Pcam 5C comes with a 10 cm flat-flexible cable and a factory-installed fixed focus lens with M12 lens mount, so it is ready to use out of the box.



Figure 3.2 image sensor with OmniBSI™ technology (5 megapixel)

OV5640 (color) image sensor is a low voltage, high-performance, 1/4-inch 5 megapixel CMOS image sensor that provides the full functionality of a single chip 5 megapixel (2592x1944) camera using OmniBSI™ technology in a small footprint package. It provides full-frame, sub-sampled, windowed or arbitrarily scaled 8-bit/10-bit images in various formats via the control of the Serial Camera Control Bus (SCCB) interface. The OV5640 has an image array capable of operating at up to 15 frames per second (fps) in 5 megapixel resolution with complete user control over image quality, formatting and output data transfer. All required image processing functions, including exposure control, gamma, white balance, color saturation, hue control, defective pixel canceling, noise canceling, etc., are programmable through the SCCB interface or embedded microcontroller.

The OV5640 also includes a compression engine for increased processing power. In addition, Omni Vision image sensors use proprietary sensor technology to improve image quality by reducing or eliminating common lighting/electrical sources of image contamination, such as fixed pattern noise, smearing, etc., to produce a clean, fully stable, color image.

III.3.2 MIPI CSI-2 Interface:

The Pcam 5C includes a dual-lane MIPI CSI-2 interface that is used to transmit image data from the OV5640 image sensor on the Pcam 5C to the attached host. It consists of three differential pairs, two for data and one for a clock signal, which are all routed to the 15-pin FFC connector. The clock and data signals are driven by the image sensor and should be treated as inputs by the attached host. The physical layer of the MIPI CSI-2 interface is called D-PHY. It is supported by the I/O blocks on Xilinx Ultrascale devices natively, but requires some passive circuitry external to the FPGA in order to be connected to Xilinx 7-series devices.

III.4. Block diagram of the realized system:

We cover the entire process of capturing video data, processing it, and outputting it. A simplified diagram of the video pipeline is shown below [12].

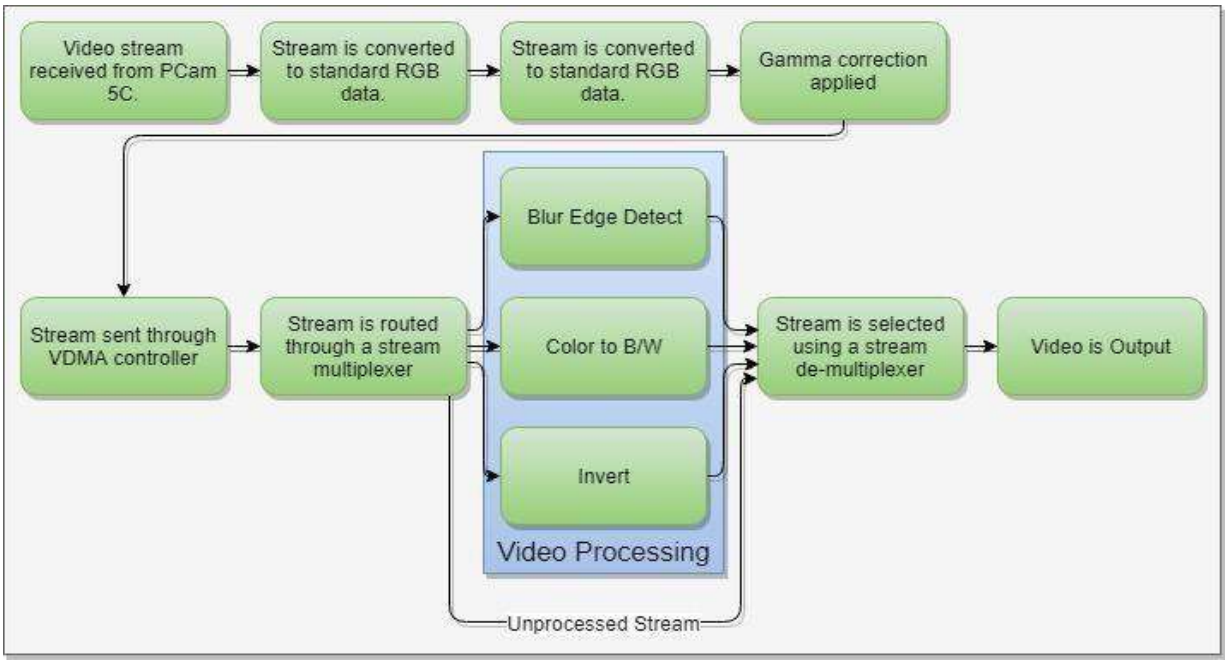


Figure 3.3: Video Pipeline Operation Flowchart

III.5. Used Vivado Video IP Cores from the xilinx library [10]:

- ✓ **Vivado SoftWare :** gonna allow us to quickly integrate and make all the connection between our Ip blocks from the **xilinx library**

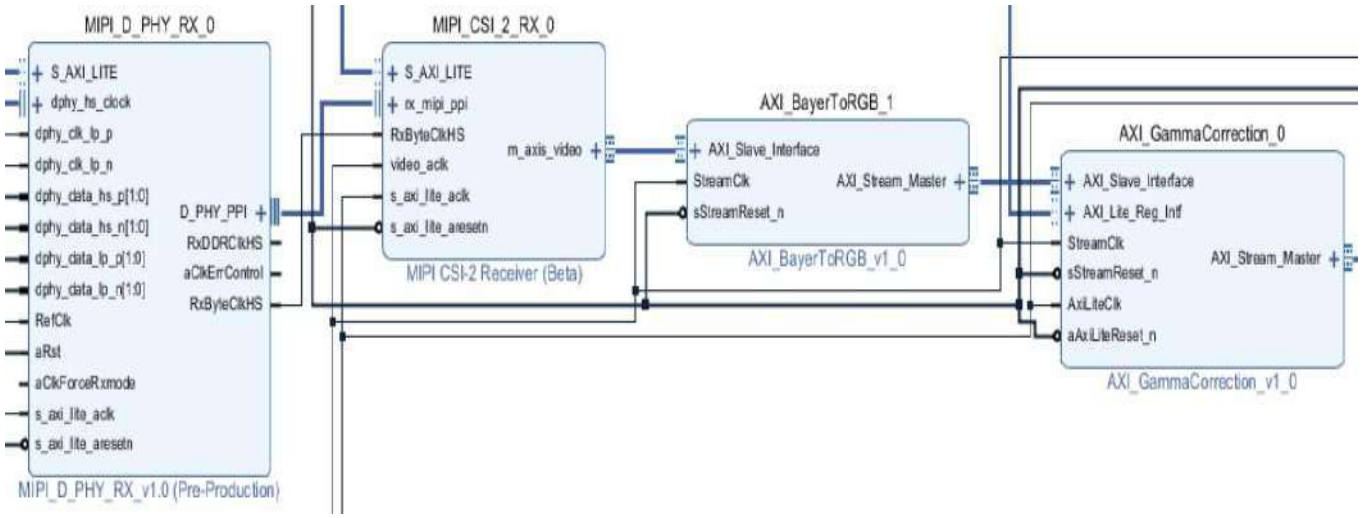


Figure 3.4: MIPI RX and Preprocessing

- **MIPI_D_PHY_RX** – Implements the physical layer of the interface, MIPI D-PHY. Data is de- serialized and the two data lanes merged together.

- **MIPI_CSI_2_RX** – Implements the protocol layer of the interface, MIPI Camera Serial Interface 2 (CSI-2). It interprets data words as pixels, lines, and frames. Video data is formatted as RAW RGB data and packed into an AXI-Stream interface forming the input side of the video processing pipeline.

- **AXI_BayerToRGB** – Interpolates the Bayer-filtered RAW RGB data into standard RGB data (32 bits: 2 bit padding, 30 bits red, green, and blue).

- **AXI_GammaCorrection** – Applies gamma correction to the video stream while also reducing the RGB color depth to 8 bits per color (24 bits: 8 red, 8 green, 8 blue). This block applies one of five gamma factors to the incoming image: 1, 1/1.2, 1/1.5, 1/1.8, and 1/2.2.

\\

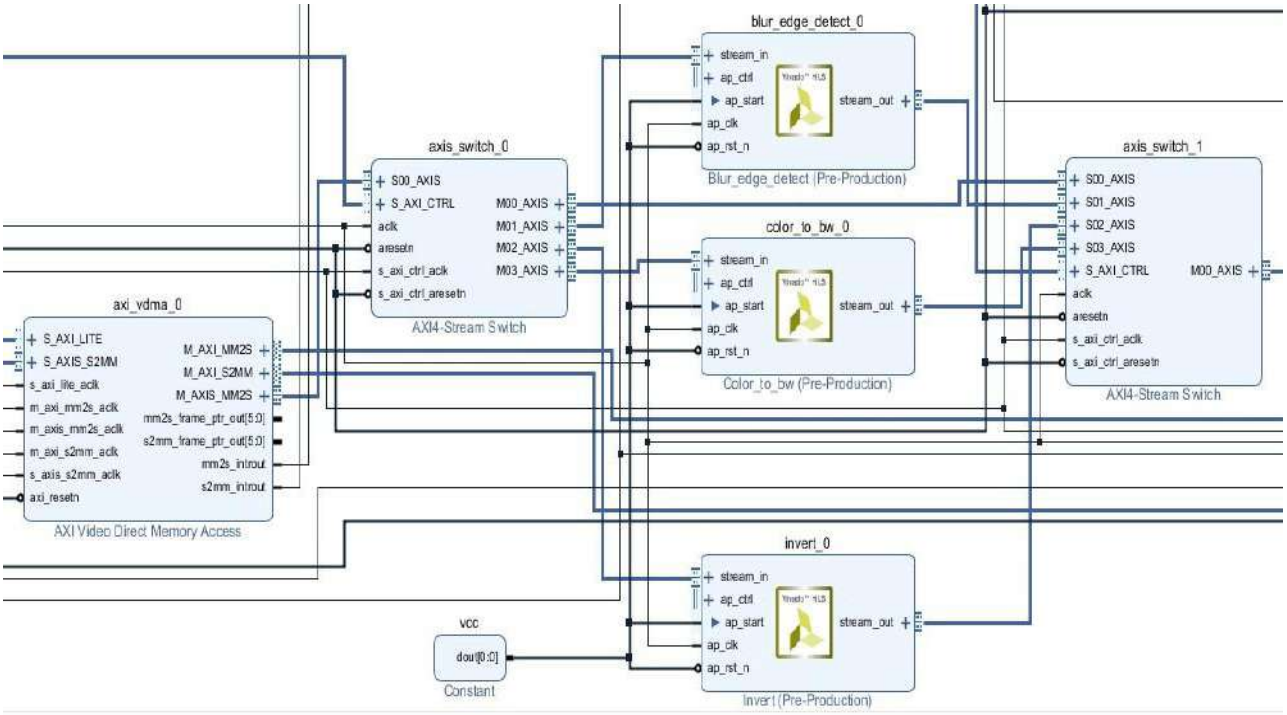


Figure 3.5: Video Stream Mux/Demux

- **AXI_Video Direct Memory Access** – Implements three frame buffers in DDR memory allowing access from within the processor, if needed. This block is connected to the Zynq memory subsystem via two high performance memory access ports. Doubling the input
- **AXI4-Stream Switch 0** – Muxes the stream to one of 4 locations
- **AXI4-Stream Switch 1** – De-muxes the video stream to one source.frame rate of 30Hz to the output frame rate of 60Hz is achieved with frame buffering.

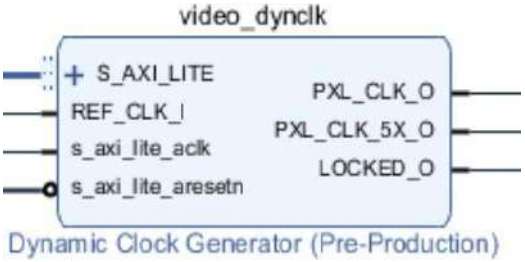


Figure 3.6: Dynamic Clock Generator

Dynamic Clock Generator – Generates the pixel clock used by the HDMI output. The pixel clock frequency changes based on the output resolution set by the processor.

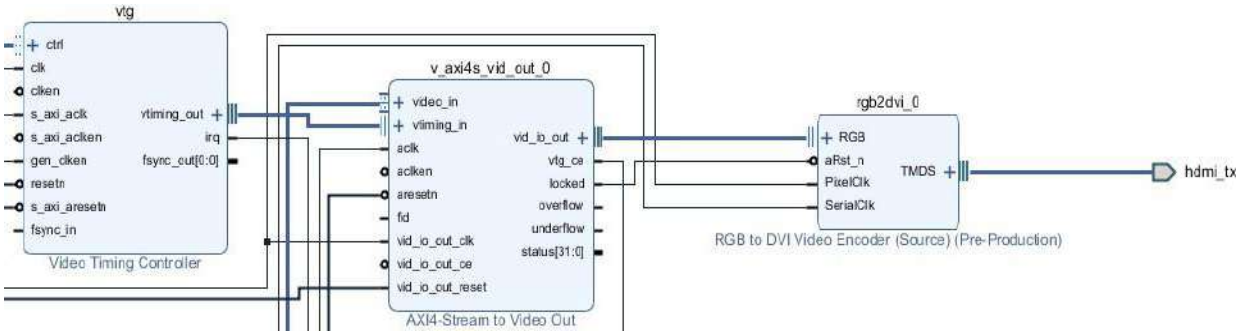


Figure 3.7: Video Output

- **Video Timing Controller** – Generates the video control signals used by the HDMI output. These signals include VSync, HSync, VBlank, HBlank, field id, and active video.

- **AXI4-Stream to Video Out** – Converts the AXI video stream to a video output interface. This block uses the video signals from the video timing controller and pairs them with the RGB data coming in from the video stream demux.

- **RGB to DVI** – Encodes RGB data as DVI to be output on the HDMI Source connector.

III.6. HLS Video Processing Cores: [10]

The HLS_video API was used to generate the video processing IP in this design. Each function is given the DATAFLOW directive to tell Vivado HLS that this will constantly stream data. The rgb img matrixes (img0, img1, etc) are given STREAM directives. This directive generates a FIFO for each variable and informs Vivado HLS that these operations will be performed in parallel.

III.6.1. Blur Edge Detect:

AXI-Stream interface is interpreted as hls::Mat (matrix) data type for easy interfacing with other HLS OpenCV functions. It is then turned to grayscale. The greyscale image matrix is then given a gaussian blur and sent through two Sobel filters, one for the X direction, one for the Y direction, and then added together. It is then converted to RGB values, then converted back to AXI video stream format and sent to the output.

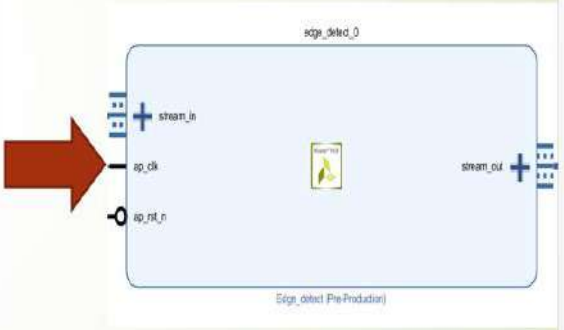
```
void blur_edge_detect(stream_t &stream_in, stream_t &stream_out)
{
	int const rows = MAX_HEIGHT;
	int const cols = MAX_WIDTH;

	rgb_img_t img0(rows, cols);
	rgb_img_t img1(rows, cols);
	rgb_img_t img2(rows, cols);
	rgb_img_t img3(rows, cols);
	rgb_img_t img4(rows, cols);

	hls::AXIvideo2Mat(stream_in, img0);
	hls::CvtColor<HLS_RGB2GRAY>(img0, img1);
	hls::GaussianBlur<5,5>(img1, img2, (double)5, (double)5);
	mysobelxy(img2,img3);
	hls::CvtColor<HLS_GRAY2RGB>(img3, img4);
	hls::Mat2AXIvideo(img4, stream_out);
}
void mysobelxy(rgb_img_t &src, rgb_img_t &dst)
{
	int const rows = MAX_HEIGHT;
	int const cols = MAX_WIDTH;

	rgb_img_t img0(rows, cols);
	rgb_img_t img1(rows, cols);
	rgb_img_t img2(rows, cols);
	rgb_img_t img3(rows, cols);

	hls::Duplicate(src, img0, img1);
	mysobel(img0, img2, 1);//Sobel transform in X direction
	mysobel(img1, img3, 0);//Sobel transform in Y direction
	hls::AddWeighted(img2, 1, img3, 1, 0, dst);
}
```



III.6.2. Color to BW:

AXI-Stream interface is interpreted as hls::Mat data type, then turned to greyscale. This matrix is then converted back to AXI video stream format and sent to the output.

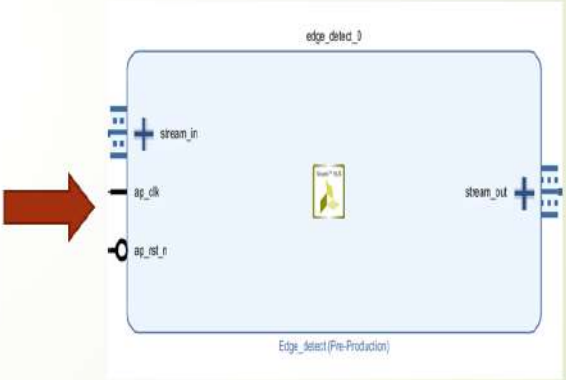
```
void color_to_bw(stream_t &stream_in, stream_t &stream_out)
{

	int const rows = MAX_HEIGHT;
	int const cols = MAX_WIDTH;

	rgb_img_t img0(rows, cols);
	rgb_img_t img1(rows, cols);

	hls::AXIvideo2Mat(stream_in, img0);
	hls::CvtColor<HLS_RGB2GRAY>(img0, img1);
	hls::Mat2AXIvideo(img1, stream_out);
}
```

HLS code



Block VIVADO

III.6.3. Invert Colors:

```
void invert(stream_t &stream_in, stream_t &stream_out)
{

	int const rows = MAX_HEIGHT;
	int const cols = MAX_WIDTH;

	rgb_img_t img0(rows, cols);
	rgb_img_t img1(rows, cols);

	rgb_pix_t pix(250,250,250);

	hls::AXIvideo2Mat(stream_in, img0);
	hls::SubRS(img0, pix, img1);
	hls::Mat2AXIvideo(img1, stream_out);
}
```

HLS code



Block VIVADO

The stream is first converted into matrix format. The matrix is subtracted by the scalar matrix "pix". This essentially inverts the colors. The matrix is then converted back to AXI video stream format and sent to the output.

III.7. Software:

The processor code is written in Xilinx SDK. The program first initializes the hardware, then scans for changes in the switches to route the video stream to where it needs to go [1].

III.7.1. Initialization:

```
int main()
{
	init_platform();

	ScuGicInterruptController irpt_ctl(IRPT_CTL_DEVID);
	PS_GPIO<ScuGicInterruptController> gpio_driver(GPIO_DEVID, irpt_ctl, GPIO_IRPT_ID);
	PS_IIC<ScuGicInterruptController> iic_driver(CAM_I2C_DEVID, irpt_ctl, CAM_I2C_IRPT_ID, 100000);
	SWITCH_GPIO sw_gpio(GPIO_SW_DEVID);
	AXI_SWITCH src_switch(SRC_AXIS_SW_DEVID);
	AXI_SWITCH dst_switch(DST_AXIS_SW_DEVID);

	SWITCH_CTL axis_switch_ctl(src_switch, dst_switch, sw_gpio, XPAR_AXIS_SWITCH_0_NUM_MI, 0);

	OV5640 cam(iic_driver, gpio_driver);
	AXI_VDMA<ScuGicInterruptController> vdma_driver(VDMA_DEVID, MEM_BASE_ADDR, irpt_ctl,
			VDMA_MM2S_IRPT_ID,
			VDMA_S2MM_IRPT_ID);
	VideoOutput vid(XPAR_VTC_0_DEVICE_ID, XPAR_VIDEO_DYNCLK_DEVICE_ID);

	pipeline_mode_change(vdma_driver, cam, vid, Resolution::R1280_720_60_PP,
OV5640_cfg::mode_t::MODE_720P_1280_720_60fps);


	xil_printf("Video init done.\r\n");
```

- The main platform is initialized. This enables caches then resets the PL (FPGA).
- Interrupt controller is initialized.
- The GPIO (cam enable pin) and IIC controllers are initialized and are connected to the interrupt controller in software.
- The switches and stream mux and demux are initialized and added to the switch ctl object.
- The camera is then initialized using the IIC and cam enable controllers.
- The VDMA is initialized and tied to the interrupt controller.
- The dynamic clock generator and video timing controller are initialized.
- The video pipeline has its video formats configured and started (1280 x 720).

III.7.2. Pipeline mode change:

```
void pipeline_mode_change(AXI_VDMA<ScuGicInterruptController>& vdma_driver, OV5640& cam, VideoOutput& vid, Resolution res, OV5640_cfg::mode_t mode)
{
	//Bring up input pipeline back-to-front
	{
		vdma_driver.resetWrite();
		MIPI_CSI_2_RX_mWriteReg(XPAR_MIPI_CSI_2_RX_0_S_AXI_LITE_BASEADDR, CR_OFFSET, (CR_RESET_MASK & ~CR_ENABLE_MASK));
		MIPI_D_PHY_RX_mWriteReg(XPAR_MIPI_D_PHY_RX_0_S_AXI_LITE_BASEADDR, CR_OFFSET, (CR_RESET_MASK & ~CR_ENABLE_MASK));
		cam.reset();
	}

	{
		vdma_driver.configureWrite(timing[static_cast<int>(res)].h_active, timing[static_cast<int>(res)].v_active);
		Xil_Out32(GAMMA_BASE_ADDR, 3); // Set Gamma correction factor to 1/1.8
		//TODO CSI-2, D-PHY config here
		cam.init();
	}

	{
		vdma_driver.enableWrite();
		MIPI_CSI_2_RX_mWriteReg(XPAR_MIPI_CSI_2_RX_0_S_AXI_LITE_BASEADDR, CR_OFFSET, CR_ENABLE_MASK);
		MIPI_D_PHY_RX_mWriteReg(XPAR_MIPI_D_PHY_RX_0_S_AXI_LITE_BASEADDR, CR_OFFSET, CR_ENABLE_MASK);
		cam.set_mode(mode);
		cam.set_awb(OV5640_cfg::awb_t::AWB_ADVANCED);
	}

	//Bring up output pipeline back-to-front
	{
		vid.reset();
		vdma_driver.resetRead();
	}

	{
		vid.configure(res);
		vdma_driver.configureRead(timing[static_cast<int>(res)].h_active, timing[static_cast<int>(res)].v_active);
	}

	{
		vid.enable();
		vdma_driver.enableRead();
	}
}
```

This function changes the resolution that the video stream is received and sent.

- The video pipeline is reset
- The gamma value is set and the camera is reinitialized.
- The mode is set, this can be 720p60, 1080p15, 1080p30, and disabled.
- The auto white balance is enabled (This can also be configured).
- The video output controller is reset and configured to output the specified resolution.
- The video output controller is enabled.

III.8. Conclusion:

In this chapter, we have introduced the features and the powerful performance of Digilent Embedded Vision Bundle (Zybo Z7-20 ,Pcam C5), and we have realized an embedded vision system to generate a streaming video processing system by designed multiple blocks (edge detection, color inverter and color to gray), with the assistance of high level syntheses HLS.

By deigned this system We were able to handle real time streaming data processing, due to the high-speed parallel processing of the programmable logic FPGA , and the premade open-source IP cores that Digilent offers.

General Conclusion

In this project, the concept of an embedded system has been introduced and a generic embedded system architecture has been explored. The role of processors within embedded systems has been discussed, and some of the processing features as well as the communication between all components within an embedded system.

The concept of embedded vision systems has been covered. Beside that, it presented several types of image sensors as well as their readout styles, speeds, and interface styles.

We have realize an embedded vision system that based on a Digilent Embedded Vision Bundle to generate a data streaming video processing system in real time, that contain multiple blocks: edge detection, color inverter and color to gray with the ability of being selected using switches.

This work allowed us to explore the practical world and help us to see one of the actual and most important application in embedded systems field, also we discover the powerful of the FPGA zybo-Z7 card and the flexibility that Xilinx software (vivado, SDK, HLS) offers, as a result we achieved real time data processing and streaming video enhancement.

Perspective:

We were looking for establishing wireless communication with another FPGA to send our steaming Video data In real time.

References:

[1] Louise H. Crockett, Ross A. Elliot, Martin A. Enderwitz , Robert W. Stewart , "The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC", University of Strathclyde Glasgow, Scotland, UK, 2014.

[2] Michael Barr, "Embedded Systems Glossary". Neutrino Technical Library. Retrieved 2020-05-10.

[3] Heath, Steve, Embedded systems design. EDN series for design engineers, Newnes, 2003

[4] Michael Barr; Anthony Massa, "Programming embedded systems: with C and GNU development tools", O'Reilly, pp. 1–2, 2006

[5] Vivek Tiwari, Sharad Malik,and Andrew Wolfe, “Power Analysis of Embedded Software: A First Step Towards Software Power Minimization,”IEEE Transaction on Very Large Scale Integration (VLSI) Systems, Vol.2, No.4, pp. 437-445, December 1994.

[6] N.Kavvadias, P.Neofotistos, S .Nikolaidis, K. Kosmatopoulos and Th. Laopoulos, “Measurement Analysis of the Software-Related Power Consumption in Microprocessors”, IEEE Trans. on Instrumentation and Measurement, Vol.53.N.4, August 2004.

[7] S. R. Vijayalakshmi; S. Muruganand, "Embedded Vision : An Introduction", Mercury Learning & Information, 2019

[8] https://www.visiononline.org/embedded-vision/embedded-vision-fundamentals, consulted the 19/05/2020

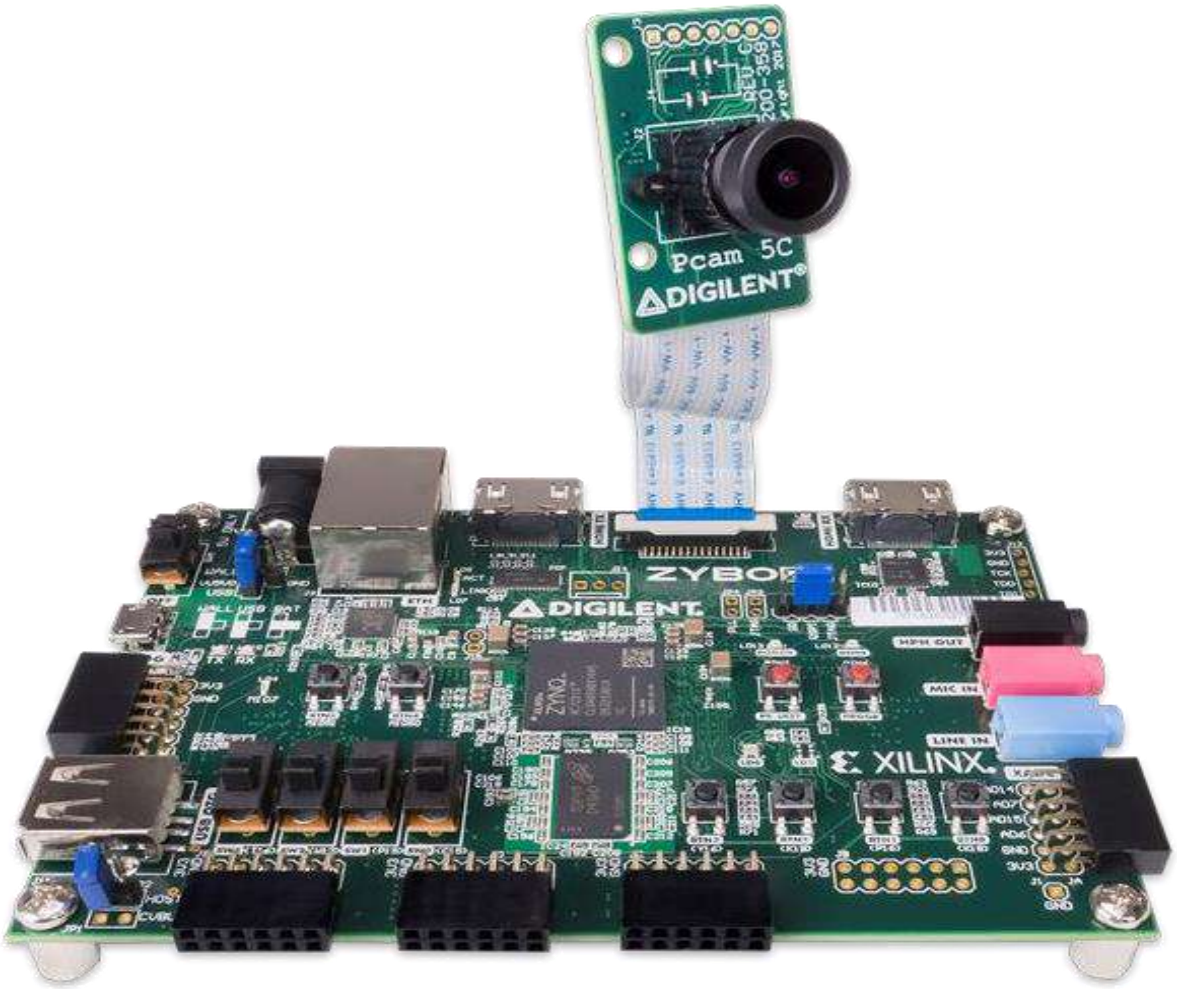
[9] Branislav Kisacanin, Shuvra S. Bhattacharyya, Sek Chai, "Embedded Computer Vision", Springer, 2009.

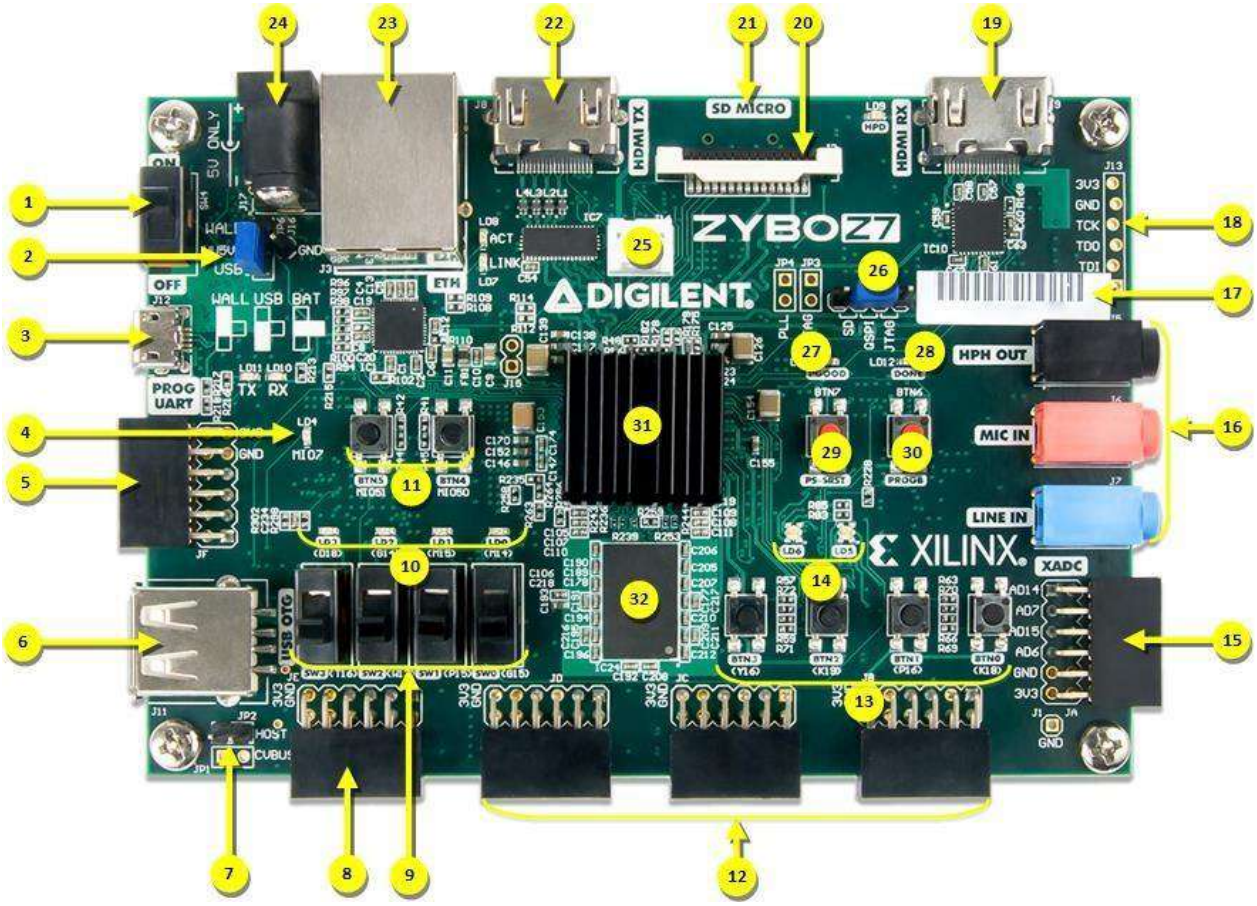
[10] Embedded Vision Demo, Digilent, 2019.

[11] datasheet of Board Reference Manual.

[12] OV5640 (Pcam C5) datasheet, OmniVision, 2011

Annex

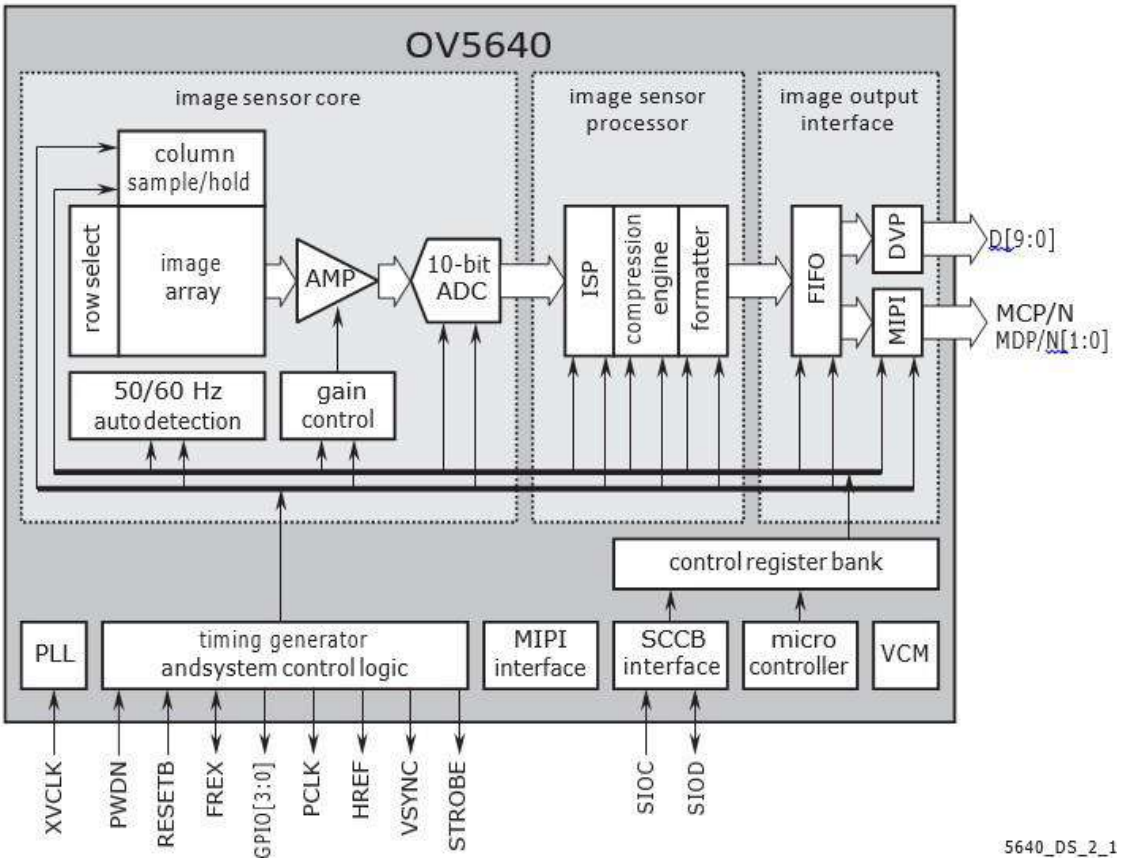




Callout	Description	Callout	Description
1	Power Switch	17	Unique MAC address label
2	Power select jumper	18	External JTAG port
3	USB JTAG/UART port	19	HDMI input port
4	MIO User LED	20	Pcam MIPI CSI-2 port
5	MIO Pmod port	21	microSD connector (other side)
6	USB 2.0 Host/OTG port	22	HDMI output port
7	USB Host power enable jumper	23	Ethernet port
8	Standard Pmod port	24	External power supply connector
9	User switches	25	Fan connector (5V, three-wire) *
10	User LEDs	26	Programming mode select jumper
11	MIO User buttons	27	Power supply good LED
12	High-speed Pmod ports *	28	FPGA programming done LED
13	User buttons	29	Processor reset button
14	User RGB LEDs *	30	FPGA clear configuration button
15	XADC Pmod port	31	Zynq-7000
16	Audio codec ports	32	DDR3L Memory

** denotes difference between Z7-10 and Z7-20.*

figure 2-1 OV5640 block diagram



Edge Detection using Gaussian Filter





Original Image

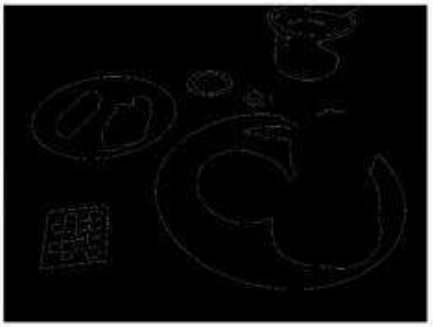


Image After Edge Detection

```
#include "hls_video.h"

typedef ap_axiu<24,1,1,1> interface_t;
typedef hls::stream<interface_t> stream_t;

void edge_detect(stream_t& stream_in, stream_t& stream_out);

#define MAX_WIDTH 1280
#define MAX_HEIGHT 720

typedef hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC3> rgb_img_t;

#define INPUT_IMAGE "fox.bmp"
#define OUTPUT_IMAGE "fox_output.bmp"
```

edge_detect.h

```
#include "edge_detect.h"

void edge_detect(stream_t& stream_in, stream_t& stream_out)
{
  int const rows = MAX_HEIGHT;
  int const cols = MAX_WIDTH;

  rgb_img_t img0(rows, cols);
#pragma HLS STREAM variable=img0 depth=1 dim=1
  rgb_img_t img1(rows, cols);
#pragma HLS STREAM variable=img1 depth=1 dim=1
  rgb_img_t img2(rows, cols);
#pragma HLS STREAM variable=img2 depth=1 dim=1
  rgb_img_t img3(rows, cols);
#pragma HLS STREAM variable=img3 depth=1 dim=1

  hls::AXIvideo2Mat(stream_in, img0);
  hls::CvtColor<HLS_RGB2GRAY>(img0, img1);
  hls::Sobel<1,0,3>(img1, img2);
  hls::CvtColor<HLS_GRAY2RGB>(img2, img3);
  hls::Mat2AXIvideo(img3, stream_out);
}
```

edge_detect.cpp

```cpp
#include "edge_detect.h"
#include "hls_opencv.h"

int main()
{
  int const rows = MAX_HEIGHT;
  int const cols = MAX_WIDTH;

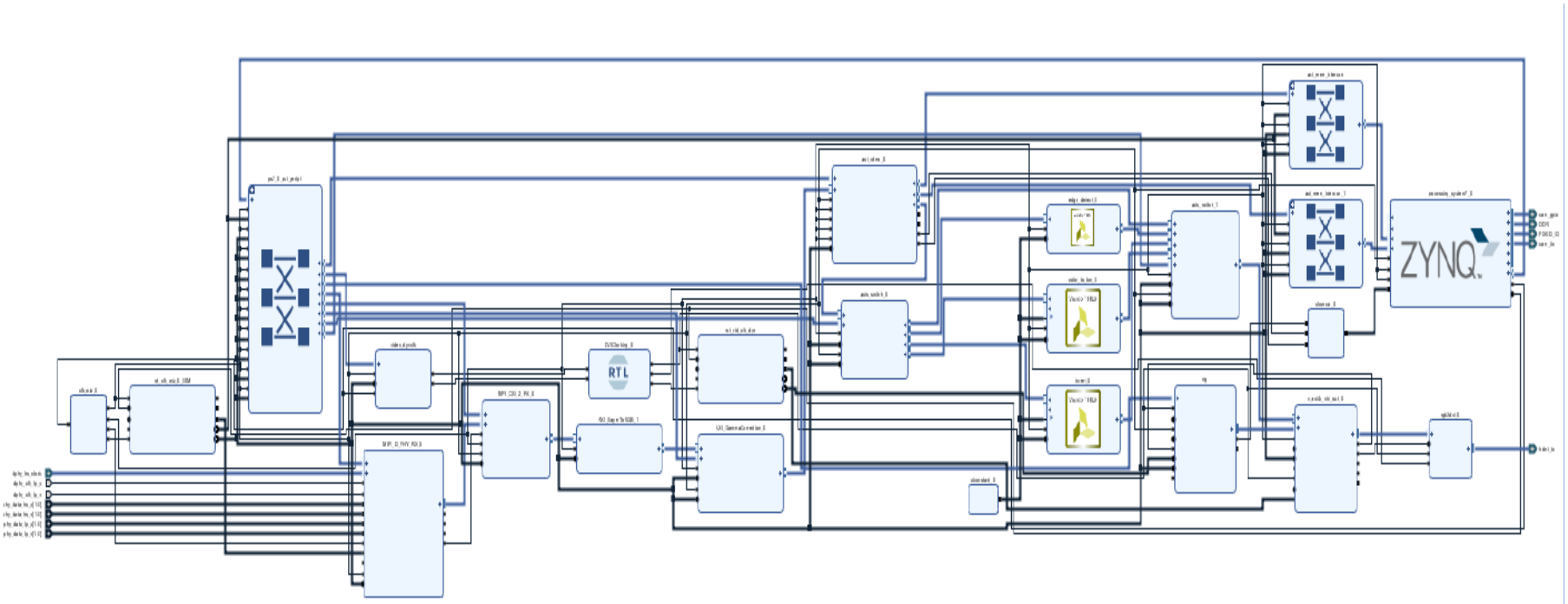
  cv::Mat src = cv::imread(INPUT_IMAGE);
  cv::Mat dst = src;

  stream_t stream_in, stream_out;
  cvMat2AXIvideo(src, stream_in);
  edge_detect(stream_in, stream_out);
  AXIvideo2cvMat(stream_out, dst);

  cv::imwrite(OUTPUT_IMAGE, dst);

  return 0;
}
```

edge_detect_test.cpp


RTL
ZYNQ