A Multiprocessor Platform Based on FPGA Technology Targeted for a Driver Vigilance Monitoring Device

by

Wafik Moussa

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2009

© Wafik Moussa 2009

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Wafik Moussa

Abstract

Medical devices processing images or audio or executing complex AI algorithms are able to run more efficiently and meet real time requirements if the parallelism in those algorithms is exploited. In this research a methodology is proposed to exploit the flexibility and short design cycle of FPGAs (Field Programmable Gate Arrays) in order to achieve this target. Hardware/software co-design and dynamic partitioning allow the optimization of the multiprocessor platform design parameters and software code targeting each core to meet real time constraints. This is practically demonstrated by building a real life driver vigilance monitoring system based on visual cues extraction and evaluation. The application drives the whole design process to prove its effectiveness. An algorithm was built to achieve the goal of detecting the eye state of the driver (open or closed) and it is applied on captured consecutive frames to evaluate the vigilance state of the driver. Vigilance state is measured depending on duration of eye closure. This video processing application is then targeted to run on a multi-core FPGA based processing platform using the proposed methodology.

Results obtained were very good using the Grimace Face Database and when operating the system on one's face. On operating the device, a false positive of eye closure must take place two consecutive times in order to get an alarm, which decreases the probability of failure. The timing analysis applied proved the importance of using the concept of parallelism to achieve performance constraints. FPGA technology proved to be a very powerful prototyping tool for complex multiprocessor systems design. The flexible FPGA technology coupled with hardware/software co-design provided means to explore the design space and reach decisions that satisfy the design constraints with minimum time investment and cost.

Acknowledgements

I would like to thank my supervisor Professor Magdy Salama for his constant help, guidance and support throughout my period of study in the University of Waterloo.

I would also like to express my gratitude to Prof. Ajit Singh for the useful parallel programming graduate course that had a great influence on my research and helped me to tackle several challenges I faced along the way.

Thanks goes to Miami Group members for their constant support especially Salam Gabran for all the help he provided when I first joined this wonderful group.

Last but not least I would like to thank my family. I am unable to express how I appreciate all their sacrifices and efforts.

Dedication

To my parents who provided endless love and support

To my brother and little sister who mean a lot to me

Table of Contents

List of Figures ... ix
List of Tables ... x
Chapter 1 Reconfigurable Chips as a Processing Platform ... 1
1.1 Introduction ... 1
1.2 Field Programmable Gate Arrays ... 2
1.2.1 Structure ... 2
1.2.2 Design Cycle ... 2
1.3 FPGA as a Processing Platform ... 4
1.3.1 Soft Core Processors ... 5
1.3.2 Hardware/Software Co-Design ... 6
1.3.3 Applications of FPGAs as a Processing Platform ... 7
1.4 Conclusion ... 8
Chapter 2 FPGA Based Multi-Core Parallel Processing Platforms ... 9
2.1 Introduction ... 9
2.2 Multiprocessor Systems ... 9
2.2.1 Introduction ... 9
2.2.2 Design Considerations ... 10
2.2.3 Programming Challenges ... 11
2.3 Design Space Exploration ... 11
2.3.1 Introduction ... 11
2.3.2 Intellectual Property and Design Reuse ... 12
2.3.3 Design Space Exploration ... 12
2.4 Practical Considerations ... 13
2.5 Hardware Design Stages ... 14
2.6 Software Design Stages ... 14
2.7 Usage of FPGAs as a Parallel Processing Platform ... 16
2.8 Conclusion ... 16
Chapter 3 Driver Vigilance Monitoring Systems ... 17
3.1 Introduction ... 17
3.2 Research Motivation ... 17

3.3 Drivers Vigilance Monitoring Systems 18
3.3.1 Sensor Based Vigilance Monitoring Systems 18
3.3.2 Visual Cues Based Vigilance Monitoring Systems 19
3.3.3 Hybrid Vigilance Monitoring Systems 22
3.4 Conclusion 22
Chapter 4 Driver Vigilance Monitoring Proposed Algorithm 24
4.1 Introduction 24
4.2 Algorithm Stages 24
4.2.1 Pre-processing 25
4.2.2 Feature Extraction 27
4.2.3 Detection and Segmentation 28
4.2.4 High Level Processing (Classification) 30
4.3 Algorithm Modeling and Execution Results 31
4.3.1 Subject 1 32
4.3.2 Subject 2 34
4.4 Limitations and Proposed Solutions 36
4.5 Targeting the algorithm for a Soft Processor Core 36
4.6 Conclusion 37
Chapter 5 Single Processor System Development 38
5.1 Introduction 38
5.2 Single Processor Implementation 38
5.2.1 Hardware 38
5.2.2 Software 39
5.2.3 Optimizations via Co-Design and FPGA Flexibility 39
5.3 Adding Camera Interface 40
5.3.1 Hardware 40
5.3.2 Software 42
5.3.3 Optimizations via Co-Design and FPGA Flexibility 43
5.4 Single Processor Fully Functional System 44
5.4.1 Hardware 44
5.4.2 Software 44
5.4.3 Optimizations via Co-Design and FPGA Flexibility 44

5.5 Multiprocessor System Implementation 45
5.6 Conclusion 45
Chapter 6 Multiprocessor System Development 46
6.1 Introduction 46
6.2 Design Objectives 46
6.3 Design Requirements 46
6.3.1 Hardware Requirements 47
6.3.2 Software Requirements 48
6.4 Cores Synchronization and Communication 49
6.4.1 Shared Resources Analysis 49
6.4.2 Cores Synchronization and Mutual Exclusion 50
6.4.3 Cores Program and Data Memory Management 52
6.4.4 Cores Communication 53
6.5 Design Space Exploration 54
6.5.1 Software Timing Extraction 54
6.5.2 Software Timing Analysis 55
6.5.3 Building Hardware Accelerators 56
6.6 Final Multiprocessor Design 60
6.7 Conclusion 61
Chapter 7 Results and Future Work 63
7.1 Introduction 63
7.2 Driver’s Vigilance Monitoring Algorithm 63
7.3 Proposed Multiprocessor System Design Methodology 64
7.4 Implemented Driver’s Vigilance Monitoring Device 64
7.5 Conclusion 65
References 66

List of Figures

Figure 1: FPGA Design Cycle ... 3
Figure 2: Proposed Design Methodology ... 15
Figure 3: Vigilance Monitoring Algorithm Stages and Involved Steps ... 26
Figure 4: Face Image and Dissimilarity Measure Value Graph ... 27
Figure 5: Edges after Applying Laplacian of Gaussian Filter and Detecting Zero Crossings ... 28
Figure 6: Column Slices Edge Pixels Count of Facial Image ... 29
Figure 7: Facial Image and Equivalent Edge Pixels Count for Each Row ... 30
Figure 8: Comparison between Open Eyes and Closed Eyes Vertical Edges ... 31
Figure 9: Hardware Layout of the Single Processor Implementation ... 40
Figure 10: Camera Interface Hardware Design ... 42
Figure 11: Single Processor Fully Functional System ... 44
Figure 12: Single Processor Modified Fully Functional System ... 48
Figure 13: Synchronization via Handshaking ... 51
Figure 14: Synchronization via Interrupts ... 52
Figure 15: Synchronization and Communication Model ... 53
Figure 16: Timing of a System Ensuring Continuous Frame Capturing ... 56
Figure 17: Full Multiprocessor System ... 61

List of Tables

Table 1: Timing Analysis.. 54
Table 2: Area Usage of Each Hardware Block 57
Table 3: Area Usage Summary after Adding Hardware Accelerator... 59
Table 4: Timing Analysis after Adding Hardware Accelerator 60

Chapter 1
Reconfigurable Chips as a Processing Platform

1.1 Introduction

The introduction of reconfigurable chips has bridged the gap between pure hardware design approaches and pure software design approaches [1]. Pure hardware design approaches utilize an ASIC (Application Specific Integrated Circuit) and/or a group of commercial off the shelf ICs (Integrated Circuits) to build an electronic system. On the other hand, pure software design approaches utilize a microcontroller or a microprocessor is used to execute programs in order to realize the required functionality. [1]

The hardware approach provides processing platforms that are characterized by fast execution since they are tailored for a specific operation or functionality. The designer optimizes the hardware for the required computation by adjusting the different hardware parameters such as bit width, number of functional blocks, their structure, etc. But once the system is built, it is totally inflexible and cannot be modified after production. Moreover, it requires a lot of design effort to integrate new features or functionality in the system and a full redesign is usually inevitable. [1]

On the other hand, the software design approach solves that inflexibility problem but it sacrifices performance. Software execution involves the traditional processes of instruction fetching, decoding and finally execution, which impose an overhead upon performing the same computation on a dedicated hardware circuit. Moreover it cannot exploit application parallelism efficiently due to the sequential nature of program execution. But modifying the system or integrating new functionality can easily be achieved by changing the program even after production. [1]

Modern reconfigurable chips prove to be a platform that possesses both flexibility and ability to achieve high performance; two qualities that prove useful in achieving superiority over both the pure software and pure hardware design trends. In the next section reconfigurable systems structure and design cycle will be covered. Our focus will be on

FPGAs (Field Programmable Gate Arrays) which are the most famous and widely used reconfigurable chips.

1.2 Field Programmable Gate Arrays

1.2.1 Structure

A simple FPGA is mainly composed of arrays of programmable logical blocks. By manipulating the programming bits of a logical block it can serve the designer functional requirement implementing a tiny portion of the target digital circuit. The logical blocks are interconnected together with routing programmable wires. Given that both the logical blocks and routing resources are programmable, any design can be mapped to the logical blocks. Afterwards, appropriate wiring can be achieved via configuring the interconnecting routing resources. [1]

Many structural decisions have to be made by FPGA manufacturers. For example the internal structure of logical blocks, the interconnect shape and programming technology. But those issues are out of the scope of this document; interested readers can refer to [1]. It is sufficient to know that modern FPGAs consist mainly of LUTs (Look up Tables), memory elements and sometimes specialized DSP (Digital Signal Processing) blocks and most of them use Static Random Access Memory (SRAM) as means for programming the LUTs and interconnect circuit.

1.2.2 Design Cycle

FPGA has a very short and low-cost design cycle compared to ASICs. The hardware can be realized or modified by just programming the FPGA in the field. Design cycle cost is mainly designer time.

Design cycles stages are as follows:

1- Design Entry: Using hardware description languages such as VHDL or Verilog.
2- Logic Synthesis: HDL file(s) logic analysis and technology mapping to implement the design logic using device resources such as logic elements.
3- Place and Route: places design on device resources and routes it through interconnections.
4- Assembler: produce programming file for device based on place and route results.

5- Programming: using the assembler output programming file to realize the design on the FPGA.

It should be worth mentioning that the cycle is iterative until satisfactory results are reached. Also there are simulators available to test the hardware before the programming phase. A functional simulation step may be performed to test that the HDL file describes the desired functionality. A post place and route simulation step is optional too in order to ensure that the delays caused by the FPGA elements and interconnect did not cause a behavioral change in the design operation.

Moreover, a set of constraints can be introduced by the designer such as pins assignments, placement constraints for a certain portion of the circuit or even delay constraints. The technology mapping and placement tools can accommodate such constraints [2].

Advanced tools can support intelligent incremental compilation so that the design cycle time can be reduced for upcoming iterations [2]. Figure (1) shows the simple FPGA design cycle stages. Interested reader can refer to the design tools manual for more advanced flows that involve assignment declaration or using third party tools and simulators.

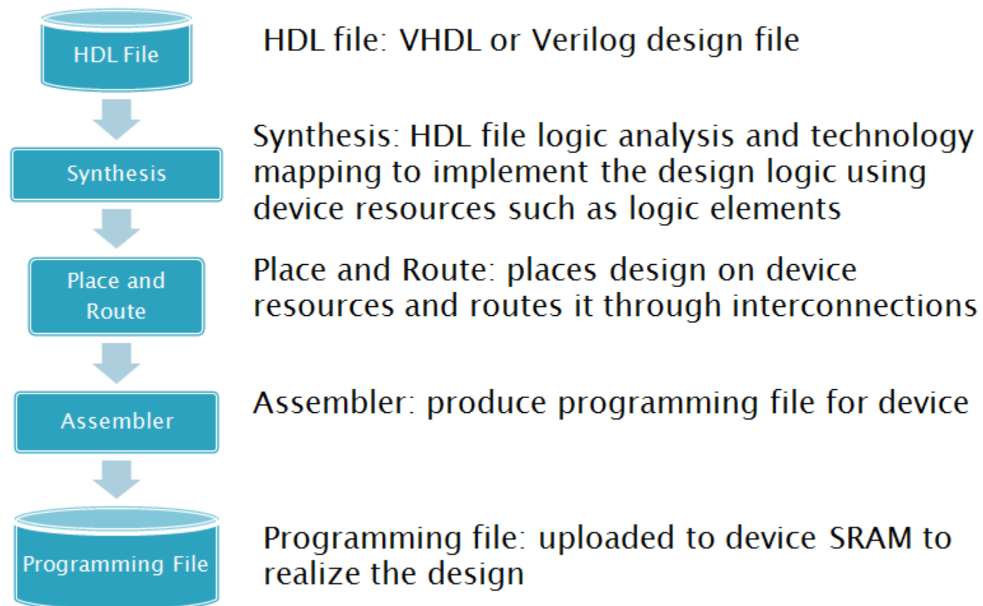


Figure 1: FPGA Design Cycle

1.3 FPGA as a Processing Platform

FPGAs or reconfigurable chips in general, have been initially used as means to implement hardware acceleration blocks. They were usually coupled with a general purpose processor that executes the main program and then use the FPGA to execute subsets of the program that can exploit hardware parallelism to achieve faster execution.

Coupling the general purpose processor was done in several ways [1]. Some general purpose processors had reconfigurable functional units embedded inside them so that the processor may speed up a certain computation. That, however, needs a constant supervision of the main processor and parallelism is not achieved. Another popular way was using the FPGA as a co-processor and that does not require except initialization and result collection from the main processor so the two units can work in parallel.

Due to advancements in FPGA fabrication, some FPGA families actually contain a hardwired microprocessor. For example the Xilinx Virtex II-Pro contain up to two hardwired IBM power PC cores. Thus the processor and the reconfigurable fabric are all inside the FPGA chip. The former structure allows the whole digital system to reside on the same chip or to be a SoC (System on Chip) [3].

The rise of soft core processors targeting FPGA has made a dramatic change to the use of an FPGA as a processing platform. Soft core processors are provided by FPGA manufacturers targeting their FPGAs to be placed on the reconfigurable fabric. They are optimized to use the FPGA logic and interconnect resources efficiently to minimize delay and to deliver best performance [3].

The soft core processor nature allows the designer to specify parameters such as cache memory size and the number of functional units (such as floating point units and hardware multipliers). The designer can also configure the functional units to suit the application with efficient area usage. Moreover, acceleration can be achieved due to the ability to use the remaining area of the reconfigurable FPGA to build helper acceleration blocks tailored for the application [3].

Soft core processors thus offer a high degree of flexibility which renders them very attractive to use for embedded systems. The designer specifies the different parameters to achieve high performance and efficient area usage.

In the next section the soft processor cores features, usage and applications are reviewed. Successful significant acceleration achieved by this method will also be covered afterwards.

1.3.1 Soft Core Processors

Soft core processors are provided usually by FPGA manufacturers or other third party vendors as intellectual property (IP) cores. They have configurable parameters so designers have wide flexibility when instantiating the processor core in their design.

The top two FPGA manufacturers Xilinx and Altera provide the MicroBlaze [4] and Nios II [5] cores respectively.

1.3.1.1 Xilinx MicroBlaze Features

The fixed feature set of the processor includes [4]:

- Thirty-two 32-bit general purpose registers
- 32-bit instruction word with three operands and two addressing modes
- 32-bit address bus
- Single issue pipeline

Optional features include [4]:

- Hardware multiplier
- Hardware divider
- Hardware barrel shifter

1.3.1.2 Altera Nios II Features

The fixed feature set of the processor includes [5]:

- Full 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- 32 external interrupt sources

Optional features include [5]:

- Single-instruction 32 × 32 multiply and divide producing a 32-bit result
- Dedicated instructions for computing 64-bit and 128-bit products of multiplication
- Floating-point instructions for single-precision floating-point operations

■ Single-instruction barrel shifter

1.3.1.3 Usage

The feature that makes soft core processors very attractive as a solution targeting reconfigurable chips is their flexibility. The designer can obtain higher performance on the expense of area by adding more features such as a hardware multiplier. Also he can save more area if execution time is not critical. Moreover soft core processors offer competitive features similar to other processors targeted for embedded systems plus the ability of adding hardware acceleration which even goes beyond the ability of normal processors.

The vast amount of logic resources available on modern FPGA opened the possibility of multi-core processor system on chip using soft core processors. This topic will be covered in details in the next chapter.

A new design methodology emerged that exploits the reconfigurable platform flexibility called Hardware/Software Co-Design.

1.3.2 Hardware/Software Co-Design

Hardware/Software Co-Design is a design methodology through which the designer builds and modifies both the system hardware and software concurrently to achieve better performance. This methodology allows the designer to achieve this goal because it offers both the advantages of pure hardware approach and pure software approach [6].

Reconfigurable chips and mainly FPGAs have actually made those concurrent changes easier because of FPGA flexibility and IP cores availability for reuse [6]. The main problem of co-design is actually selecting the pieces of software code that are going to be processed on dedicated hardware that needs to be implemented on the FPGA. This problem is called hardware/software partitioning [6].

The partitioning process is successively repeated until satisfactory performance is reached. Refinement is achieved by adding constraints on communication delay and load balancing between the processor and custom hardware.

The partitioning problem can be solved on different levels of abstraction. Computer aided design (CAD) tools can be used to solve the problem automatically or it can be solved manually by an expert designer. The partitioning problem is formulated by considering the properties of both the custom hardware blocks and the software running on the processor. This optimization problem can, for example, target the minimization of execution time under the constraint of certain area usage on chip [6].

1.3.3 Applications of FPGAs as a Processing Platform

In literature, FPGAs reconfigurability has been exploited to either implement custom hardware co-processors or accelerators for an external processor or to implement a full system on chip through the usage of soft core processors.
The flexibility of those platforms allows designers to achieve higher performance compared to conventional pure software systems. Moreover they have less cost and design time than ASIC based systems. Various approaches were used in order to reach those goals.

In [7] an FPGA is used to build a hardware acceleration block for visual information processing. A PC is connected to the FPGA through the PCI bus and the system runs MPEG-7 Global Motion Estimation Software with an average factor of improvement of 5 compared to the use of the PC alone.

In [8] the parallel beam backprojection algorithm is totally implemented in hardware based on an FPGA. The flexibility of the FPGA allowed the designers to implement a variant of the hardware that exploits the parallelism in the algorithm to achieve more acceleration. The first variant executes the algorithm in 19% of the time using a 1 GHz Pentium. The second variant which further exploits the parallelism executes the algorithm in 4.6% of the time using a 1 GHz Pentium.

In [9] a custom vector processor is used as a co-processor with Altera Nios II processor and is compared with the performance of an automated hardware accelerator generator from software functions (Altera Nios C2H [10]). The Nios processor coupled with the co-processor performance is compared against a stand-alone Nios and Nios with automatically generated hardware acceleration block using C2H in three different applications; block

matching motion estimation (video processing), image median filter (image processing) and AES encryption.

Much research has also focused on different approaches to select the optimum parameters for soft core processors [11, 12]. Some work has been also made to dynamically implement hardware acceleration units [13].

1.4 Conclusion

Using FPGA as a processing platform is very attractive because of its flexible nature and low-cost design cycle. The flexibility allows further acceleration of the system by adding custom hardware tailored towards the application. Results reviewed in the literature support that FPGAs are more powerful than both pure hardware and pure software traditional implementation methods.

The high availability of IP cores allows designers to quickly integrate a full processing platform and helper hardware blocks. It also unlocks the ability of design reuse.

Moreover, the design cycle is significantly short due to the emergence of hardware/software co-design approaches. Researchers have also enhanced tools to support the co-design approach.

The literature indicates that soft core processors can be used to efficiently build an embedded system with significant performance edge over the pure software approach. Additionally, the design procedure is far simpler than using a pure hardware approach. In this thesis, soft core processors will be exploited to build the target system. Coupled with the flexibility of this platform and the short design cycle, a platform tailored for the application will be built and optimized for best performance and lowest area on chip.

It is also worth mentioning that the vast amount of logic elements in modern FPGAs allows designers to include multiple processor cores on a single reconfigurable chip. This topic will be discussed thoroughly in the next chapter.

Chapter 2
FPGA Based Multi-Core Parallel Processing Platforms

2.1 Introduction

The ever increasing software complexity of modern applications has always posed a challenge for microprocessor manufacturers. Hardware designers have always strived to boost the performance of their processing platforms to handle those demanding applications [14]. However, the old systematic approach of enhancing a single processor is no longer able to meet the performance requirements, so manufacturers opted to multi-processor platforms [15]. These platforms are now a reality not just a hot research topic. That fact can be justified by the new microprocessor generations possessing multiple CPU cores produced by the leading manufacturers such as Intel [16] and AMD [17].

Embedded systems, especially medical devices, also can make use of such platforms since they run complex algorithms in real-time. Medical devices processing images or audio or executing complex artificial intelligence algorithms would be able to run more efficiently and meet real-time requirements if the parallelism in those algorithms is exploited.

This chapter highlights the usage of FPGAs to prototype and to develop multiprocessor parallel processing platforms. Multiprocessor systems development and challenges involved are discussed, followed by the reasons that make FPGA one of the most suitable platforms for design space exploration. Finally, a literature review is presented showing the effectiveness of FPGAs in tackling this design problem.

2.2 Multiprocessor Systems

2.2.1 Introduction

Hardware manufacturers have been surrounded by a lot of difficulties which slowed down the process of boosting the performance of microprocessors. The old approach of decreasing transistor size and manufacturing single processors with a larger number of transistors is no

longer able to provide significant improvement [15]. A lot of barriers limited the ability of this method to keep up with the ever increasing performance requirements.

The increased power and heat dissipation posed a serious difficulty and a so-called “Power Wall”. Also the inability of memory speed to cope with higher operation frequency of modern processors introduced a “Memory wall”. Moreover, instruction level parallelism (ILP) can no more be exploited via compiler and architectural modifications which is referred to as the “ILP Wall” [14].

So multiprocessor/multi-core systems are becoming the only reasonable resort to work around that problem though both hardware designers and software programmers face many challenges.

2.2.2 Design Considerations

In brief, the design considerations can be expressed as questions spanning different design aspects as follows [14]:

- Hardware
  - What is the structure of the processor core building block?
  - Are all processor cores identical (homogenous) or different?
  - How the processor cores are interconnected?
- Programming Model
  - How to describe applications to exploit the parallel platform?
  - How to program the hardware?
- Applications
  - How to describe applications to exploit the parallel platform?
  - How to program the hardware?

Those questions, though they seem distinct and separable, are actually related and the proposed set of answers must be compatible. A lot of research effort has to be made in order to prove that a proposed system will provide the best performance for the application at hand.

2.2.3 Programming Challenges

When it comes to programming parallel processing platforms a range of challenges are faced. For decades programmers' goal was to design software that executes sequentially. Tools such as compilers and debuggers have been built putting that in mind as a basic fact. But in the parallel processing era, that idea has changed and thus both programmers and tools need to change their perspective. In order to unleash the power of a parallel platform the following practices in software design have to be applied [15]:

- Exploiting "Task Parallelism"
  - Breaking down program into subtasks while analyzing and accounting for data dependency
  - Assigning subtasks to different processors
- Exploiting "Data Parallelism"
  - Data splitting over different processors
- Load Balancing
  - Efficient Task/Data Mapping
  - Scheduling

Also testing and debugging parallel programs is more challenging because scenarios dictated by the presence of more than one processor emerge and need to be tested.

2.3 Design Space Exploration

2.3.1 Introduction

In the previous sections the design considerations and challenges facing the designer of a multiprocessor system were presented. Because of the wide set of design parameters, flexibility and short design cycle time are required in order to achieve the best performance in minimum time. A Field Programmable Gate Array (FPGA) possesses those characteristics which make it the best candidate for design space exploration [14].

FPGA flexibility allowed the emergence of intellectual property components that can be reused in different designs, giving rise to the concept of design reuse. This will be discussed in the next section.

2.3.2 Intellectual Property and Design Reuse

The hardware design flow allows simulation and verification of hardware sub-modules. Those sub-modules can be reused later on as building blocks to realize larger designs. IP (Intellectual Property) components ranging in size and complexity emerged for different applications. They are provided to hardware designers to accelerate their design cycle [18, 19]. Open and protected configurable IP blocks can accelerate FPGA designs significantly, since they are ready made and can be directly used in implementing a larger design.

When it comes to multiprocessor systems there are a wide range of IP cores available to be used ranging in granularity from simple functional units to a full processor IP core. Many processors are now available as IPs to be used in FPGA designs. In fact FPGA vendors often offer these processors (e.g., NIOS II Processor from Altera [5] and MicroBlaze Processor from Xilinx [4]). Moreover some vendors equip their FPGAs with hardwired processor cores.

2.3.3 Design Space Exploration

Exploiting the short and low cost design cycle, the flexibility of the device and IP cores, FPGA proves to be a very powerful design space exploration tool for a multiprocessor system. Modern FPGAs have a vast amount of logic elements allowing the construction of multiprocessor systems on chip (MPSoC). Though FPGAs MPSoC are not claimed to achieve highest performance, they can easily help assessing the ability of the design to overcome a certain challenge and their design iteration is quick with no overhead costs [14]. FPGAs can explore the different design choices as follows:

- Hardware:
    - Easy to vary number of cores
    - Various processor IP cores can be used
    - Most IP cores are configurable (cache size, functional units), even some support custom instructions
    - Open IP processor cores are available providing total freedom in changing the core
    - Easily switch between homogenous and heterogeneous multiprocessor architectures
    - Different interconnection buses and connection topologies can be explored easily
    - Processor cores can be extended with hardware acceleration blocks for certain applications/kernels

- Programming Model:
    - Different programming models can be explored
    - Creating communication hardware or hardware mutex can help in the abstraction of programming model
    - Support of standard libraries such as Message Passing Interface (MPI) can make the programmer's job easier when porting parallel applications
- Applications:
    - MP each with a different HW acceleration block can satisfy a wide range of applications

2.4 Practical Considerations

Though FPGAs have a great potential as a design space exploration tool, the usage of an FPGA in a practical system may not be the best choice for a final product. FPGAs can be used to evaluate relative performance of various MP architectures but they do not achieve the maximum performance for each one. Delays associated with programmable interconnects and the propagation delay between various logic elements – though optimized by the place and route process – are still significant. These factors limit the clock speed that can be used for the system. Moreover, FPGAs are not efficient when it comes to power consumption, so for a portable device, an FPGA will drain a lot of power which is unfavorable.

Thus an FPGA is the best platform for prototyping, testing and evaluation of an architecture, but the design has to be ported to an ASIC in order to maximize performance and minimize power consumption. Another problem that emerges in the process of porting a design is the usage of protected IP cores. All IPs used must be available for deployment on standard cell ASICs in order for migration to succeed.

In the next section the proposed hardware design stages of a multiprocessor system are highlighted. A section that covers the software design stages follows. It should be noted that the hardware and software design stages are overlapping since hardware/software co-design methodology is exploited. The proposed design methodology is illustrated in figure (2).

2.5 Hardware Design Stages

- **Stage 1: A Simple Single Processor System**
  A soft processor core is deployed and integrated with a memory controller and interfaced to either on chip or off chip memory. The system in this stage is capable of running a sequential program that implements the required algorithm in C language.
- **Stage 2: Building Helper Interfaces**
  Helper interfaces are built for the required system to operate. For example a UART or a flash memory controller can be added. Other devices used for the user interface such as an LCD controller can be incorporated too. Functions to use those peripherals should be built and tested.
- **Stage 3: Adding Extra Cores and Design Exploration**
  Extra cores are then added. Hardware design decisions must be taken either through experimentation or problem analysis. To ensure that the cores are functional, testing programs are run in parallel.
- **Stage 4: Adding Inter-processor Communication Capability**
  Communication capabilities are added at this stage. Shared memory or message passing mechanisms can be deployed depending on the application. The system is then ready for software partitioning.
- **Stage 5: Building Hardware Accelerators**
  Depending on each core's role in the system, hardware acceleration blocks can be added. This process requires the analysis of each core program and finding the most computationally expensive functions and implementing a hardware acceleration block that can perform the same computation. This step is optional; it exploits the FPGA flexibility to further achieve a shorter execution time than conventional multiprocessor systems. Multiprocessor parameters are tuned in this stage too.

2.6 Software Design Stages

- **Stage 1: Modeling**
  The algorithm is modeled on Matlab or in C language and verified to implement the required functionality. The model is used as a base for verification in the next stages.

- **Stage 2: Porting Code to C Language**
  The model code is converted to C language. All modeling language built-in functions called must be written from scratch in C. The code is then compiled and fully executed on the target processor.
- **Stage 3: Building Helper Functions**
  Helper functions for the embedded device are built. They are used to control the external peripherals and user interface. Message passing or shared memory utilization functions must be built also to achieve inter-processor communication.
- **Stage 4: Parallelizing the Code**
  Finally the code is parallelized. In that stage even load balancing should be achieved to obtain significant acceleration. Data and task division between cores is one of the toughest procedures in building efficient parallel processing systems. Dynamic partitioning changes on the software side are done in this stage too.

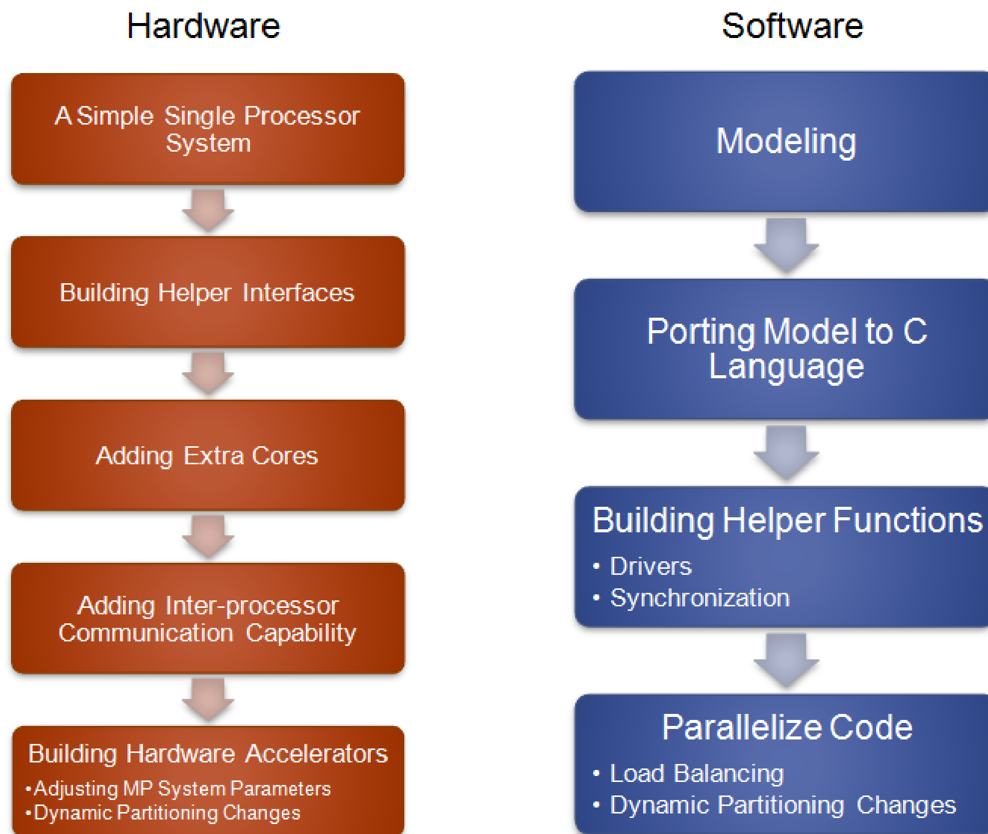


Figure 2: Proposed Design Methodology

2.7 Usage of FPGAs as a Parallel Processing Platform

In the literature, FPGAs have been used not only as a design space exploration tool, but also as a platform that significantly boosts the execution speed of a certain algorithm or application running on a single processor.

In [20] a design space exploration technique based on cosimulation is presented. Automated design tools that solve the software partitioning problem were also created in order to accelerate MPSoC design cycle. In [21] and [22] two frameworks are created to convert a sequential program into an MPSoC implementation automatically. In [23] a research tool is built to automatically build models and optimize the number of cores and partition the software on them.

2.8 Conclusion

Multiprocessor platforms are becoming the processing platform of choice when the application requires fast execution time dictated by either real-time constraints or high volume of data to be processed. This is mainly because single processor improvement is facing a lot of fabrication challenges and improvement bottlenecks.

FPGAs prove to be an ideal platform for design space exploration of multiprocessor systems to overcome design challenges and to help in making design decisions. The flexibility and low cost design cycle, coupled with the availability of processor IP cores, makes the process of exploring alternatives smooth and quick.

In the light of the information reviewed in the literature, FPGAs seem ideal for the construction of a cheap multiprocessor embedded system in a limited time. The flexibility and short design iteration time will help integrating the system and writing parallel programs in a short time with a significant performance gain.

In this research work, FPGA multi-core platform will serve as a processing engine for a driver assistive device. The platform will adapt to the application in order to achieve real-time video processing constraints. A focus on the device function, which is driver vigilance monitoring, and the research done in that field will be given in the next chapter.

Chapter 3
Driver Vigilance Monitoring Systems

3.1 Introduction

Driving is a complex task where the driver is responsible of watching the road, taking the correct decisions on time and finally responding to other drivers' actions and different road conditions. Vigilance is the state of wakefulness and ability to effectively respond to external stimuli. It is crucial for safe driving.

According to recent statistics driver fatigue or vigilance degradation is the main cause of 17.9% of fatalities and 26.4% of injuries on Ontario roads [24]. Vigilance levels degrade mainly because of sleep deprivation, long monotonous driving on highways and other medical conditions and brain disorders such as narcolepsy. Drowsy drivers face difficulties in maintaining the level of concentration required on the road. They fail to respond to different road stimuli because of their delayed response time, which ultimately leads to serious accidents.

3.2 Research Motivation

Traffic regulation authorities strive to achieve maximum road safety. Since drivers' vigilance degradation and falling asleep on the wheel is one of the most common reasons of traffic accidents, vigilance monitoring is considered a hot research topic. According to a recent Canadian survey conducted by Traffic Injury Research Foundation, 1 in 5 Canadian drivers fell asleep at the wheel over a 12-month period. Moreover, 57% report having driven when drowsy over the same period [25]. Drivers are unaware of the serious consequences of driving in that state, so instead of pulling over and taking a break they just continue driving. An artificial intelligence device alerting the driver to pull over when he/she starts to sleep may save lives. These devices are called driver vigilance monitoring systems.

In this chapter different types and principles of operation of driver vigilance monitoring systems will be highlighted. Also an assessment of each type's advantages and disadvantages will be discussed.

3.3 Drivers Vigilance Monitoring Systems

3.3.1 Sensor Based Vigilance Monitoring Systems

Sensor based vigilance monitoring systems are based on different kinds of sensors. They are subdivided into (a) mechanical sensors installed on the car itself to observe the vehicle motion pattern or the driver-vehicle interface interactions and (b) sensors attached to the driver to measure physiological signals variation. Some systems incorporate both types of sensors in order to achieve higher accuracy of detecting the driver state of vigilance.

3.3.1.1 Mechanical Sensors

In the literature, mechanical sensors are widely used to measure different parameters of the car motion or driver interaction with the driver-vehicle interfaces in order to detect the current state of driver's vigilance. Those systems assess the driver's state of vigilance indirectly via measuring different variables.

The pressure exerted by the driver on the steering wheel falls beneath a certain threshold for a drowsy driver [26]. This measure is somehow unreliable unless another sensor for the car state of motion is incorporated in the system. This is mainly because drivers tend to take their hands off the steering wheel at stop state. Also the way drivers hold the steering wheel may require multiple pressure sensors distributed all over the steering wheel and processing of those multiple signals is required in order to estimate the current grip level.

Another measure used is the time derivative of the force exerted on the acceleration pedal [26]. The pattern of this measure varies according to the driver state of alertness. The presence of spikes in the pattern over the general trend indicates that the driver is alert. The pattern flattens when the driver is drowsy. This method is very cheap since it requires only a force sensor on the acceleration pedal and it is non-intrusive. But the system in [26] has not been tested in a real driving environment, so this hypothesis has not been experimentally validated.

In [27] and [28] multiple sensors are incorporated into the system, they measure the position of the car inside the traffic lane (lateral car position), the steering wheel angle and

vehicle speed. The signals are collected and wavelet transformation is applied followed by a statistical analysis to extract features that are finally passed to a classifier to determine the driver's state of vigilance. There is a serious disadvantage of using this method because these signals vary with different road conditions such as bends, winds and road deformation. The isolation of these factors requires a significant amount of experimentation and classifier training. Thus false positives may render the system unreliable.

3.3.1.2 Physiological State Sensors

The measurement of physiological signals can reflect the driver's state of vigilance. Systems exploiting the driver's physiological state use sensors attached directly to the driver followed by signal processing techniques to evaluate the driver's alertness.

In [29] electroencephalogram (EEG) brain signals are measured using surface sensors attached to the driver scalp. The four EEG signals delta, theta, alpha and beta are fed to a digital signal processor (DSP) and a neural network is constructed to assess the state of vigilance.

In [30] both EEG and ECG (Electrocardiogram) signals are fed to the system via 128 electrodes. Statistical analysis is then applied to reach a decision about the state of attention of the driver.

The problem with this kind of systems in general is their intrusive nature. The sensors have to be directly in contact with the driver which renders those systems very inconvenient.

3.3.1.3 Hybrid Sensors

In order to achieve higher efficiency, these systems combine both mechanical sensors and physiological state sensors. Normally these systems are not cost efficient and still have the same problem of driver's inconvenience.

3.3.2 Visual Cues Based Vigilance Monitoring Systems

This kind of vigilance monitoring system depends on a camera module that captures the driver's facial image. The image is processed to extract different visual cues that enable the

system to assess the vigilance state of the driver. Different methods of image processing have been proposed in the literature.

In [31] a face model is used to extract features of interest such as eyelid closure and head orientation. Several image processing techniques are used in order to extract these parameters. The face is detected using a color distribution model then geometrical properties of the human face are used to locate the eyebrows. Afterwards, the eye is located based on its relative position to eyebrows. This method proves to be efficient because it can adapt to different face orientations and it can detect if the eyes are blocked from the camera view. The extracted parameters are then used directly to evaluate the driver vigilance in real-time.

In [32] and [33] the eyes are detected using circular Hough transform with dynamic diameter which detects circles in the image. The circle pairs found are then checked for similarity then a neural network classifier is applied to the circle pairs found to verify that the circles detected are actually the eyes and if the classifier result is negative then the eyes are either blocked or closed. This method proves to be efficient and simple because there is no need for face detection and the validation step ensures correct operation of the system.

In [34] two concentric rings of infrared LEDs (light emitting diodes) are used around the camera module. The inner ring illuminates for even fields captured, resulting in an image where the iris is very bright. The outer ring illuminates for odd fields, resulting in a dark iris. The subtraction of odd image from the even one, after de-interlacing, results in an image where background is deemphasized and the eyes appear as bright circles. Still noise blobs may appear in pairs. So using eye geometrical properties a support vector machine classifier is trained to find out which pair of bright circles represents the actual driver eyes. The downside of this method is the usage of extra hardware and more image processing of the images captured. But it certainly pays back because of the increased efficiency and simplicity of the algorithm applied afterwards.

In [35] a hierarchical dynamic Bayesian classifiers network is employed to detect the face and facial landmarks of the driver. The network is organized so that a top-down approach of eye detection is achieved. First the face is detected, followed by eye regions detection. From there, the eye brows are detected to finally reach the eye. The classifiers can communicate

together to achieve the dynamic property of the network. The particle filtering mechanism is adjusted through communicating of each node with its parent. Different parameters can be extracted from the eye classifier such as driver's intent beside the vigilance. This method complexity is very high and this is its major disadvantage.

In [36] the fact that the variance of the grey level intensity of an open eye is greater than that of a closed eye is exploited. Starting with a candidate eye region the mean and variance projection in both vertical and horizontal direction is calculated. Using those values the system detects whether the eyes are open or closed. Also pose estimation is incorporated into the system to add extra reliability in case of blocked eye. This system is simple and can achieve the requirements with minimum cost.

In [37] the same setup of infrared LEDs used in [34] is deployed to locate the eye pupil. A validation step then follows. The only difference between [37] and [34] is the usage of Kalman filtering in order to predict the pupil position in the upcoming frame to reduce processing required to detect the pupil.

In [38] and [39] the authors of [37] further enhance the system by integrating other conditions such as weather, information about sleep history and work environment to the visual cues by using a Bayesian network model. This increases the complexity of the system significantly.

In [40] a very simple system is employed, first of all the eye area region is fed as an input (skipping the eye detection step), then using two given images – one with the eye opened and one with the eye closed – a distance measure is calculated between the new image and the two stored images. A distance threshold is then applied to determine the state of the driver's eyes. Though very simple, the system does not provide sufficient reliability because the variation of the face angle and lighting conditions can affect the system's decision.

In [41] and [42] the eyes are detected using infrared LEDs and tracking is done using Kalman filtering. A finite state machine is then used to model transitions between different states of eye opening and closure. Also two states are reserved for the status of tracking whether it is working or lost. Both the eye opening and face pose are fused together as a measure of vigilance using a fuzzy classifier.

Vigilance monitoring systems based on visual cues all possess the advantage of being non-intrusive which makes them very appealing to vehicle manufacturers because they are very convenient for the driver. The cost of a camera module is relatively low; the major system cost is the processing platform. Most of the research work reviewed here just models the system on a PC which makes it inapplicable in real life.

3.3.3 Hybrid Vigilance Monitoring Systems

The hybrid vigilance monitoring systems depend on both sensors and visual cues in evaluating the vigilance level of the driver. The sensor signals and visual cues extracted are fused together and are processed using artificial intelligence algorithms.

In [43] an architecture is proposed for that system. The sensor data concerning the vehicle such as speed, environmental conditions is collected and forwarded to a traffic risk estimation module. Parameters extracted from visual cues and driver-vehicle interface sensors are forwarded to a vigilance diagnosis module. Both modules extract data and send it over to a hierarchical manager to assess the current risk level. The risk level is then forwarded to a driver warning system that takes decision to fire the alarm. Another module works on identifying the driver so that the hierarchical manager might take the driving behavior and physical conditions into consideration. No further information about the operation of each unit is provided in the paper; it just discusses the overall architecture.

It is obvious that this is a sophisticated system with high complexity. Thus this solution has a relatively high cost and training will require a significant time in order to achieve acceptable results.

3.4 Conclusion

In this chapter the importance of driver vigilance monitoring systems is discussed. They can help reduce the number of traffic accidents on the road significantly. Different kinds of systems and research efforts made in that field were covered.

It is clear that sensor-based systems face a lot of difficulties. First of all, intrusive sensors attached to the driver are inconvenient and are not applicable for personal vehicles which

represent a large portion of traffic. Usage of mechanical sensors alone is unreliable since efficiency is relatively low.

Vigilance monitoring based on visual cues extraction by a camera has been subject to extensive research. Many methods have been discussed in this chapter. The main shortcoming of this method is the need for real time processing of video frames which requires a high speed platform. Research has been mainly done using modeling languages on a PC. In this thesis research will be taken two steps further, first the algorithms will be targeted to an embedded processor ready for deployment. Then acceleration will be obtained using multiprocessors on an FPGA to achieve real time constraints. This acceleration might prove useful in the future. It will allow building a more complex hybrid system without violating the real-time constraints.

A hybrid vigilance monitoring system architecture was discussed in order to achieve coverage of current research efforts in this field.

Chapter 4
Driver Vigilance Monitoring Proposed Algorithm

4.1 Introduction

In this chapter, the proposed algorithm for driver vigilance monitoring is discussed. The algorithm depends on visual cues extracted from the driver image captured by a digital camera to be installed on the dashboard. The overall system can be classified as a video-processing device. The algorithm will be targeted to run on a reconfigurable platform (an FPGA to be more specific). Then the platform flexibility and ability to accommodate multiple processing cores will be exploited to meet the real-time constraints of the application. This will be explained in detail in the next chapters. It is worth mentioning that accelerating the algorithm run time opens the door for adding more complex visual cues or even implementing a hybrid system that depends on other sensors as explained in chapter 3.

4.2 Algorithm Stages

The algorithm proposed is a computer vision algorithm that aids in the detection of the current driver state of vigilance. It detects the current state of the driver eyes in a certain frame (open or closed). Applying this algorithm on consecutive video frames may aid in the calculation of eye closure period. Eye closure periods for drowsy drivers are longer than normal blinking, a fact that can be exploited to monitor a driver state of vigilance. The algorithm consists of four major stages as the majority of computer vision algorithms:

1- Pre-processing

   The input image size is adjusted by rescaling and RGB to greyscale conversion takes place.

2- Feature Extraction

   The pre-processed image is then used to extract features. This is mainly done by applying edge detection and using face symmetry properties.

3- Detection and Segmentation

The edges and face center are then used to detect the subject face and a novel segmentation technique is applied to extract the eye region which is our region of interest.

4- High Level Processing (Classification)

Finally a classification step is applied in order to decide whether the subject eyes are open or closed in this image of the video stream.

In the next subsections, each stage is further explained in details showing the exact steps involved in achieving our target. Figure (3) shows the four stages and the steps involved in the algorithm.

4.2.1 Pre-processing

Pre-processing is divided into:

4.2.1.1 Image Rescaling

Through the manipulation of the registers present in the camera module which delivers a maximum resolution of 5 Megapixels, pixels are skipped during the scan out process to decrease the image resolution to VGA (640x480). More reduction is applied by the frame grabber through further skipping in order to reach a lower resolution image. This step is performed to decrease processing load without sacrificing important details.

4.2.1.2 Bayer Format to Greyscale Conversion

The camera module pixels are output in a Bayer pattern format consisting of four “color components” - Green1, Green2, Red, and Blue (G1,G2, R, B) - representing three filter colors.

The Bayer format is converted by the frame grabber to RGB format. The RGB values are then converted to greyscale.

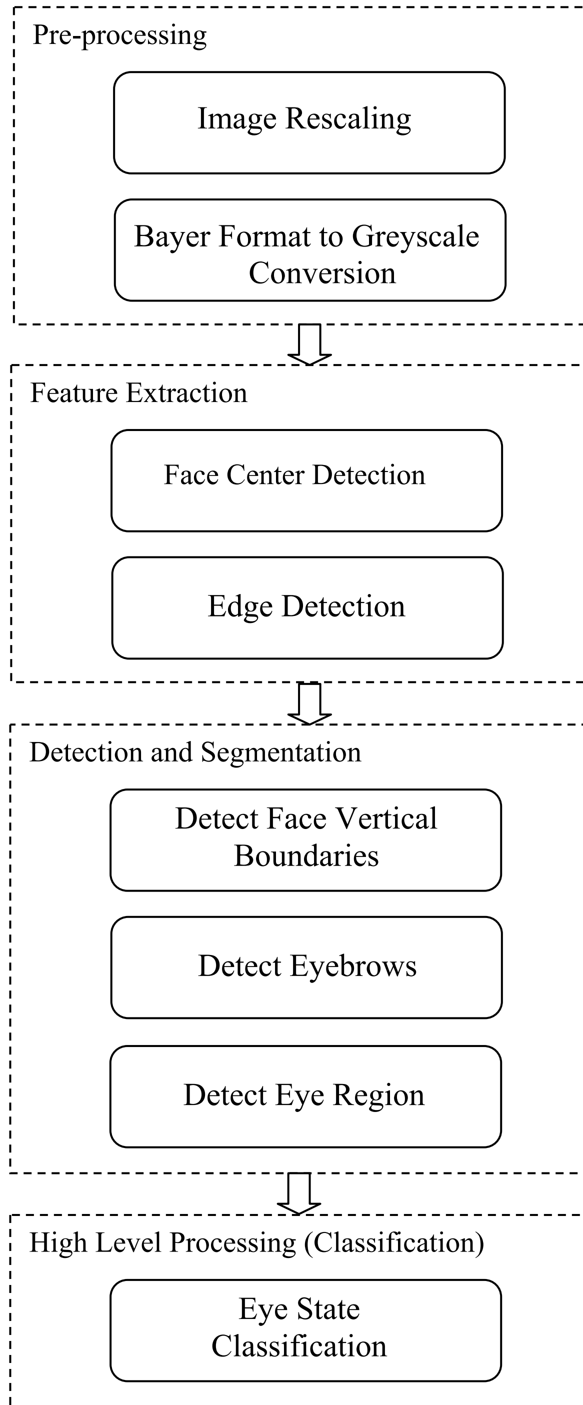


Figure 3: Vigilance Monitoring Algorithm Stages and Involved Steps

4.2.2 Feature Extraction

This stage can be further subdivided into:

4.2.2.1 Face Center Detection Using Symmetry

Depending on the symmetry properties of the human face, the center of the face can be easily detected using the following steps:

1- Scan the picture using all vertical columns from ¼*number of columns to ¾* number of columns.
2- Calculate absolute difference in intensity between corresponding pixels in each two columns with equal distance around the current column (in the range of ¼*number of columns).
3- Find the minimum absolute difference signifying maximum symmetry and mark as face center.

Figure (4) below shows a face image from Grimace Face Database [45] and the dissimilarity measure calculated around each column. It is clear that column with minimum value corresponds to the face center.

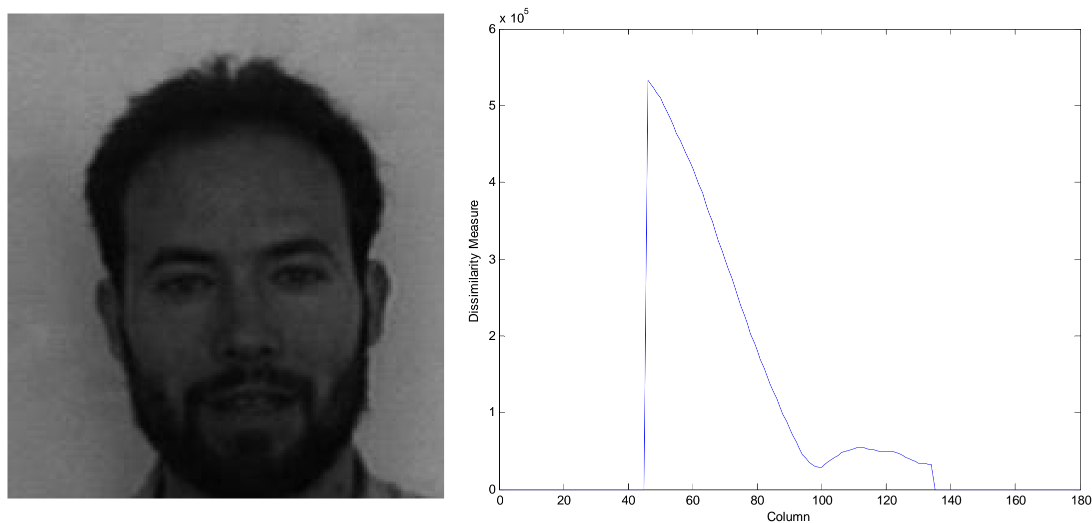


Figure 4: Face Image and Dissimilarity Measure Value Graph

4.2.2.2 Edge Detection

Applying Laplacian of Gaussian filter on the image facilitates the detection of edges by finding zero crossings. For further processing two binary images are created. One with both horizontal and vertical zero crossings (vertical and horizontal edges) and one with only horizontal zero crossings (vertical edges only). Figure (5) shows a facial image after applying Laplacian of Gaussian filter and detecting zero crossings.

Figure 5: Edges after Applying Laplacian of Gaussian Filter and Detecting Zero Crossings

4.2.3 Detection and Segmentation

We need to extract the eye region which is our region of interest. This is done through a series of steps that depend on human face characteristics given the features we extracted in the previous stage.

4.2.3.1 Detect Face Vertical Boundaries

In this step starting with the face center we detected earlier and the binary edges image:

1- Scan the edges image left of the centerline using a 3 pixel wide column.
2- Sum up the pixels in the 3 pixel wide columns slice.
3- Find the maximum, which signify the left edge of the face.
4- Repeat the previous 3 steps but going right from the centerline to find the right edge.

Figure (6) below shows the values calculated by summing up edges according to the previous steps. The maximums on the two sides of the face center mark the left and right boundaries of the face with an acceptable accuracy.

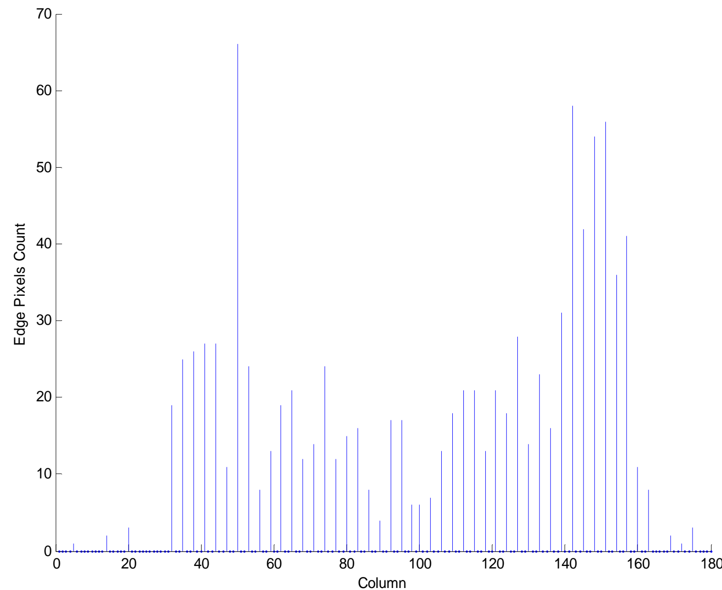


Figure 6: Column Slices Edge Pixels Count of Facial Image

4.2.3.2 Detect Eyebrows

In order to detect the eye region which is our final goal we have to depend on a visual cue that exists in both open eye and closed eye images. So our algorithm depends on eyebrows in order to extract the eye region. Searching for eyebrows proceeds in the following steps:

1- Scan the edges image starting from ¼*rows to 3/5 *rows using a 5 pixel wide row.

2- In the 5 pixel row slice, sum up the pixels' intensity in a 32 pixel wide region away from the center by 8 pixels on both sides with a positive sign. Then sum up the pixels' intensity from the center till the 8th pixel on both sides with a negative sign. This will ensure there is an edge discontinuity near the center (signifying eyebrows). This method excludes continuous edges such as hair and forehead edges.

3- Apply dynamic eyebrows detection method which starts by finding the maximum summation calculated in step 2. Then taking the 5 rows around the maximum and calculating the variance of each row original pixels in columns center-15 to center-30. Then mean variance of the 5 rows is calculated.

4- If the mean variance calculated is higher than a certain threshold this maximum is eliminated and step 4 is repeated again. This dynamic method ensures that the edges of the eyes are not mistaken as eyebrows. Since the eye has a high variance opposed to eyebrows which normally have a uniform grey level and thus low variance.

Figure (7) below shows the values calculated by summing up row slice edges according to the previous steps. Notice that the maximum sum is not equivalent to the eyebrows location. The dynamic method explained in the previous steps is able to eliminate maximums that do not satisfy the variance threshold constraint and successfully locate the eyebrows (refer to section 4.3 for results).

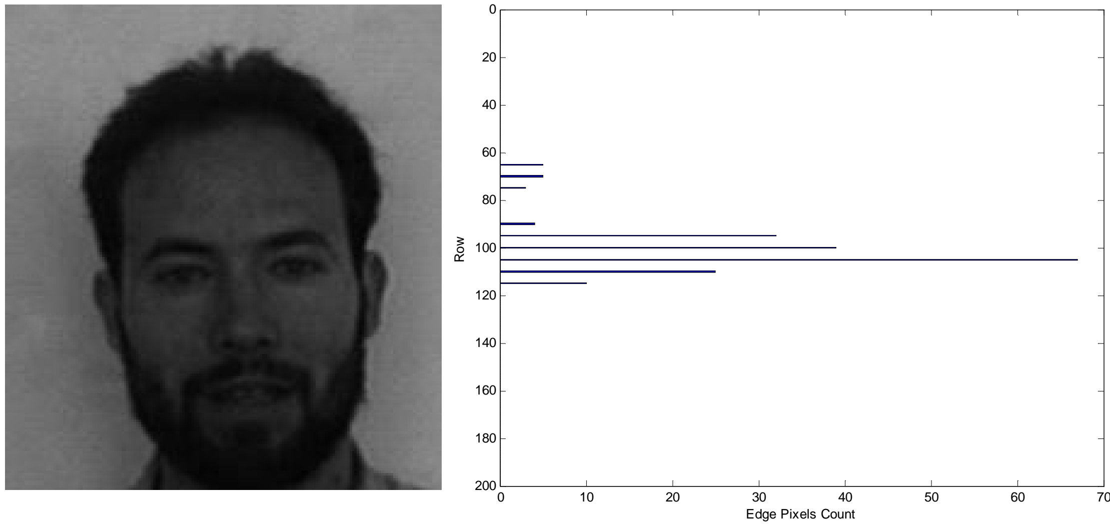


Figure 7: Facial Image and Equivalent Edge Pixels Count for Each Row

4.2.3.3 Detect Eye Region

Starting with eyebrows location, pick up the area below it with 3 pixels and of 16 pixels height as the eye region. This number can be tuned because in the application the face distance from the camera can be assumed constant (refer to section 4.3 for results).

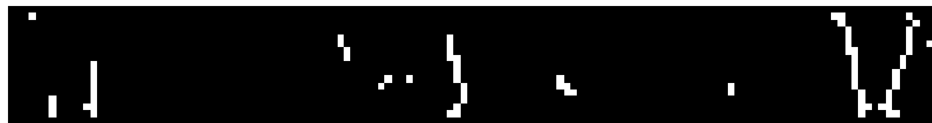
4.2.4 High Level Processing (Classification)

In this final stage we try to find out if the eye is open or closed in the image. This can be easily related to the number of vertical edges in the eye region, for closed eyes has absolutely no vertical edges but an open eye has numerous edges (pupil edges and eye opening vertical edges). So using the following steps we can reach a decision:

1- Sum vertical edges of eye region in a 32 pixel wide area 8 pixels away from the center.
2- High sum signifies open eyes and low sum signifies closed eyes. A threshold can be obtained by experimentation.

Figure (8) below compares vertical edges detected in the eye region of an open eye facial image and a closed eye facial image. There is a significant difference in the vertical edges count in the region the algorithm sums up leading to accurate detection of eye state.

(a) Open Eye Vertical Edges



(b) Closed Eye Vertical Edges

Figure 8: Comparison between Open Eyes and Closed Eyes Vertical Edges

4.3 Algorithm Modeling and Execution Results

The algorithm was modeled and different parameters were tuned through testing and experimentation. The Mathworks Inc. Matlab [44] was used as a modeling platform due to the availability of different readymade functions in the Matlab Image Processing Toolbox. The toolbox came in handy to rapidly prototype the algorithm. Moreover, it helped debug the algorithm showing the effect of application of each step on the given image.

A set of facial images was used to evaluate the algorithm effectiveness from the Grimace Face Database [45] provided for free from the Computer Vision Science Research Group, University of Essex, UK. This database was selected because each subject is asked to perform different facial expressions including eye closure, which is essential for our research.

Below algorithm execution results are shown for two subjects each opening eyes for one image and closing them for the other. The algorithm accurately detects the eyes region and the threshold selected in the classification stage is able to provide the required distinction between open eye and closed eye images.

4.3.1 Subject 1

4.3.1.1 Open Eyes

Original Image

Face Center Detection Result

Face Vertical Boundaries Detection Result

Eyebrows Detection Result

Eye Region Detection Result

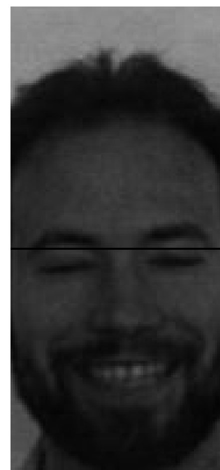
Eye State Detection Decision = Eyes Open

4.3.1.2 Closed Eyes

Original Image

Face Center Detection Result

Face Vertical Boundaries Detection Result



Eyebrows Detection Result

Eye Region Detection Result

Eye State Detection Decision = Eyes Closed

4.3.2 Subject 2

4.3.2.1 Open Eyes

Original Image

Face Center Detection Result

Face Vertical Boundaries Detection Result

Eyebrows Detection Result

Eye Region Detection Result

Eye State Detection Decision = Eyes Open

4.3.2.2 Closed Eyes

Original Image

Face Center Detection Result

Face Vertical Boundaries Detection Result

Eyebrows Detection Result

Eye Region Detection Result

Eye State Detection Decision = Eyes Closed

4.4 Limitations and Proposed Solutions

The algorithm somehow faces numerous problems in detecting the eye region related to different face postures. For instance tilted faces fail the face center detection step. The detected center is skewed leading to miscalculation of eyebrows location and consequently eye region location.

The proposed solution here is to detect that the face is tilted before proceeding with further steps. This can be achieved by calculation of face center position relative to the left and right face edges. If center is skewed then the face is tilted and driver is assumed to be vigilant. This assumption is based on the fact that a tilted face does not imply a drowsy driver, since it is harder to maintain that posture in case of sleepiness. Normally a sleepy driver cannot tilt his/her face to watch the road conditions around him/her. Using that logical assumption, such cases are directly considered as open eye frames.

Another problem is the nodding forward case. This case does not affect symmetry, yet the eyes will fail to be detected. But there is actually no point of implementing a solution here, since the algorithm normal operation will reach the conclusion of a closed eye which is normally implied by a nodding down driver.

Finally, drivers wearing sunglasses prevent the system from detecting the eye. Thus our system falls unable to proceed under this condition. An extension can be made in order to work around this case by using special hardware. But the problem was not tackled in this research.

The algorithm could accurately detect the eye state of all subjects in the database that posses eye open and eye closed images (12 subjects – 24 images). This gave confidence that the algorithm can perform well for images that are going to be captured in the final system using the camera module.

4.5 Targeting the algorithm for a Soft Processor Core

For the purposes of this research an Altera FPGA chip will be used. Thus the Altera Nios II soft core processor has been selected as the multiprocessor system building block.

The first step to build this system is to run the whole algorithm on a single Nios II processor. So the Matlab model is converted manually into C language, building all the built-in Matlab functions in C language from scratch.

This step took place in the early development stages and the results were verified against the Matlab model to ensure consistency.

4.6 Conclusion

In this chapter, the proposed algorithm for driver vigilance monitoring is discussed. The algorithm is explained in details. Each stage importance and involved steps were highlighted. Also some results were displayed which proves the effectiveness of the proposed algorithm.

The algorithm is relatively simple compared to the other algorithms discussed in chapter 3. This simplicity makes it more suited to run on a portable device. The limitations of the algorithm were covered and how we worked around them in order to allow the system to be more robust.

Finally the first step of the software implementation was covered, which is targeting the algorithm for a single soft processor core based on an FPGA. In the next chapters the hardware and software implementation details will be covered using the proposed design methodology.

Chapter 5
Single Processor System Development

5.1 Introduction

With the driver's vigilance monitoring algorithm modeled and verified to be functional on Matlab, the process of system development started. The hardware/software co-design methodology discussed in chapter 2 minimizes design time through the proposed concurrent hardware and software development stages. Minimization is achieved by concurrently adapting the software code to major hardware architectural changes. Throughout the system development process FPGA technology proves to be an ideal design space exploration platform because of its flexible dynamic nature.

In this chapter each stage of the single processor system implementation details and design choices will be discussed. Moreover, each section will highlight the role of hardware/software co-design and/or dynamic partitioning in reaching a more optimized design in less time. The effectiveness of the proposed design methodology in chapter 2 will be assessed throughout the upcoming chapters.

5.2 Single Processor Implementation

5.2.1 Hardware

The first stage was to implement a single Nios II processor system. The system was integrated using Altera SOPC (System on a Programmable Chip) Builder software. Different IP components were used such as an SRAM memory controller to interface the processor to an external SRAM memory chip. The memory chip acted as program and data memory for the processor. A debug module was also incorporated in the system so that the software development environment can communicate with the processor through a USB connection via standard input and output streams.

The whole system hardware was configured on an Altera Cyclone II FPGA based on a DE2 evaluation board. The evaluation board provided a lot of external interfaces and helper

chips. For instance, it houses a 512KB SRAM chip that was wired to the SRAM controller on the FPGA. For a complete list of resources on the evaluation board please refer to the user manual [46].

5.2.2 Software

The Matlab code was previously ported to C language. So the code was compiled and targeted towards the Nios processor. The images used to test the model were given as inputs stored on the system memory.

The operation was verified by printing out the outputs of different algorithm stages and comparing them against the outputs obtained from Matlab. The system obtained similar results to that of the Matlab model and so the final decision about each image eyes state was identical to the one obtained by the model.

5.2.3 Optimizations via Co-Design and FPGA Flexibility

Even in this very early stage, co-design methodology and FPGA flexibility allowed system optimization and tuning to obtain better performance.

It is worth mentioning that the Nios II processor operates on a 50 MHz clock and that is considered slow compared to state of the art processors. The processing time for an image was noticeably long. Thus it became apparent that this performance would not be acceptable even in the presence of more than one core. But the reconfigurable platform can significantly enhance this performance by using special hardware accelerators and modifying the processor architecture to speed up the execution of a certain algorithm.

After a quick timing analysis it was noticed that the edge detector 2d convolution process is very time consuming because it acts on floating point numbers. The Nios processor executes floating-point multiplications using software emulation. Therefore, on the hardware side, a floating-point unit (FPU) core was added. Also embedded multipliers on the Cyclone II fabric were used to implement a full hardware multiplier for the processor core integer multiplications.

On the software side, drivers for the FPU were added and thus each multiplication of type float was forwarded to the FPU instead of using software emulation. The same was applied for the integer multiplications.

So the system was optimized in a matter of hours to reduce the execution time significantly. A relatively weak processing core was able to handle the task in acceptable time (considering that multiple cores will be used). Figure (9) shows the architecture of the single processor implementation.

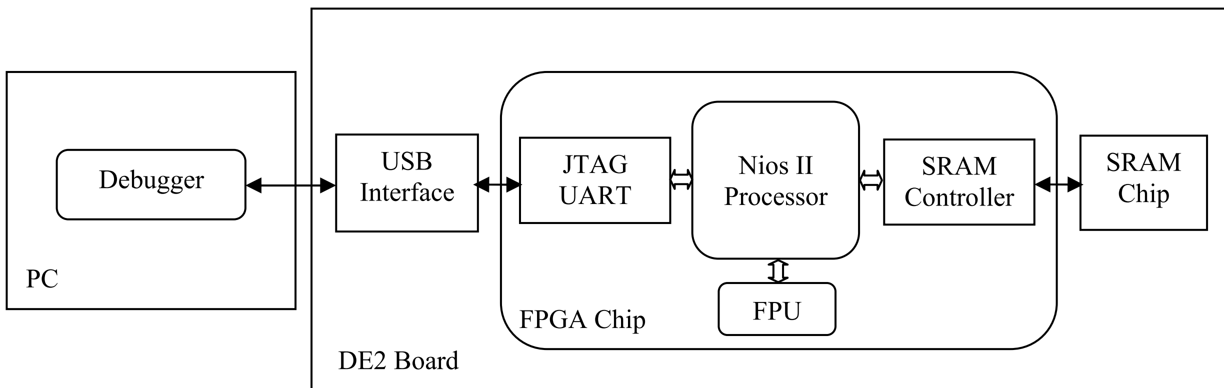


Figure 9: Hardware Layout of the Single Processor Implementation

5.3 Adding Camera Interface

5.3.1 Hardware

In this stage a camera module equipped with a CCD (Charge Coupled Device) image sensor and programmable logic was interfaced to the system. The TRDB_T5M camera module used is an advanced 5 Megapixel color camera. The sensor is connected to a digital output controller that scans the R, G, B pixels in Bayer's raw format sequence. The digital output speed, resolution, color gain, exposure and many other parameters are programmable using registers in the digital output controller. The registers can be read or written using the I2C bus. For more information refer to the camera module datasheet [47].

The camera input and output pins were connected to the DE2 board prototyping general purpose pins. A frame grabber hardware module was built on the FPGA to capture the image.

The frame grabber hardware is shown in Figure (10) and can be divided into blocks in the following manner:

1- Capture and Control Unit:
   The unit receives the data stream produced by the camera module. It ensures that the values captured belong to a valid frame. It keeps track of the current line position in the frame and pixel position in the line.

   The unit has 2 control inputs; one to start streaming frames and one to stop streaming frames when the current frame ends.

2- Bayer to RGB Unit:
   The unit is responsible for receiving the data from the capture unit and deducing the Red, Green and Blue components at each pixel position. This is achieved using a line buffer so that the four values (R,G1,G2,B) containing colour information are analyzed together. The unit produces the red, green and blue components along with a data valid signal to ensure that the current data displayed on the outputs is valid.

3- Four port SDRAM Controller:
   The SDRAM controller has two writing ports and two reading ports. The RGB values are divided to two 16-bit values written in different locations simultaneously using the two write ports. The ports use FIFO (First In First Out) modules to ensure that no data is going to be lost because of the high frequency of the camera output. The memory writes data on the edge of the camera clock and data valid line is used as write enable. Moreover, the controller has two read ports, which are connected to the Nios II processor via parallel input output units (PIO). The processor generates the read enable signal and reading clock to ensure that data is fed in at a rate that the processor can handle. Finally the controller is connected to the SDRAM chip available on the DE2 board.

4- I2C Controller and Programmer:
   The controller connects to the I2C camera module lines in order to provide the programmer logic access to the camera in the correct format. The programmer then sends the registers address and data in order to program the resolution to 640x480 and to adjust other camera parameters.

Parallel Input Output units (PIOs) were added to the processor in order to connect it to the hardware interface:

1- Reset: output PIO to drive the reset input of the camera interface.

2- Start Capture: Output PIO to drive the start capture input of the capture and control unit.

3- End Capture: Output PIO to drive the end capture input of the capture and control unit.

4- RGB: Input PIO connected to the output signal from the two SDRAM memory read ports.

5- Read Enable: Output PIO to memory read enable signals.

6- Memory Read Clock: Output PIO to memory read clock to supply outputs in sequence.

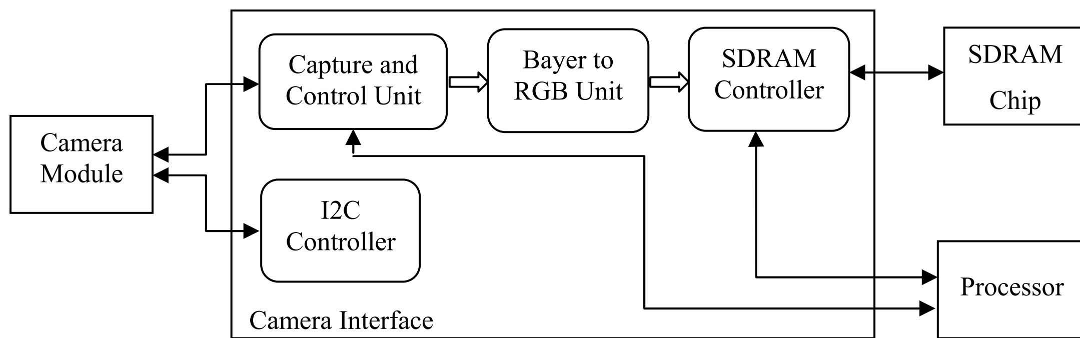


Figure 10: Camera Interface Hardware Design

5.3.2 Software

For the software component, a driver function was created in the following manner:

1- The reset is signalled via the reset PIO.

2- A start capture command is issued via the start capture PIO.

3- An end capture is issued afterwards right away in order to maintain only one frame in the memory.

4- Memory read enable is set high via the read enable PIO.

5- Memory read clock is triggered and a pixel is retrieved by the processor.

6- Step 5 is repeated 640*480 times to retrieve all the pixels.

The driver was modified because a lower resolution can be obtained by sub sampling and thus accelerating the upcoming processing. So we skip odd rows and columns and then the final image resolution obtained is 320x240 pixels.

Also the algorithm does not require color information, so only the green component was obtained and then treated like greyscale information. It is not accurate for the naked eye, but for the computer vision algorithm at hand the color quality is not significant as long as edges are preserved.

5.3.3 Optimizations via Co-Design and FPGA Flexibility

Co-design and dynamic partitioning were used intensively in the camera interface design step. Many alternatives were considered, tested and evaluated. The partitioning of the capture functionality between hardware and software was done dynamically by changing the hardware architecture and simultaneously adapting the software to the change.

First of all, minimal custom hardware was used. The camera module was connected directly to the processor via PIOs. The software has then to deal with data arrangement and Bayer to RGB conversion. Moreover, it has to emulate an I2C controller. But this design failed because the frequency in which the camera operates is very high and the processor fails to correctly acquire the image.

The second step was implementing a clock divider to slow down the camera module input clock and try to get the data. This method partially succeeded but it was not reliable and the module sometimes produced garbage because of the long scanning time.

Another approach was adopted in which the data is saved to a memory and then dumped out so that the rate is controllable. This succeeded and from there the final design emerged.

The final design allowed the conversion of Bayer to RGB on hardware minimizing the software role and accelerating the capture speed. Also an I2C controller IP block was coupled by the required sequences as a state machine lifted the burden of camera programming off the processor.

5.4 Single Processor Fully Functional System

5.4.1 Hardware

A UART module was added to the processor in order to send debugging messages to Matlab and allow real time verification of the results. Since there was no way to display images on the embedded system, the UART is used to transmit the image captured and different results such as face center, eyebrows position, eye region boundaries and final decision for each frame processed. Matlab will then be used to display results as images and markers to serve debugging purposes. The overall system architecture is shown in figure (11).

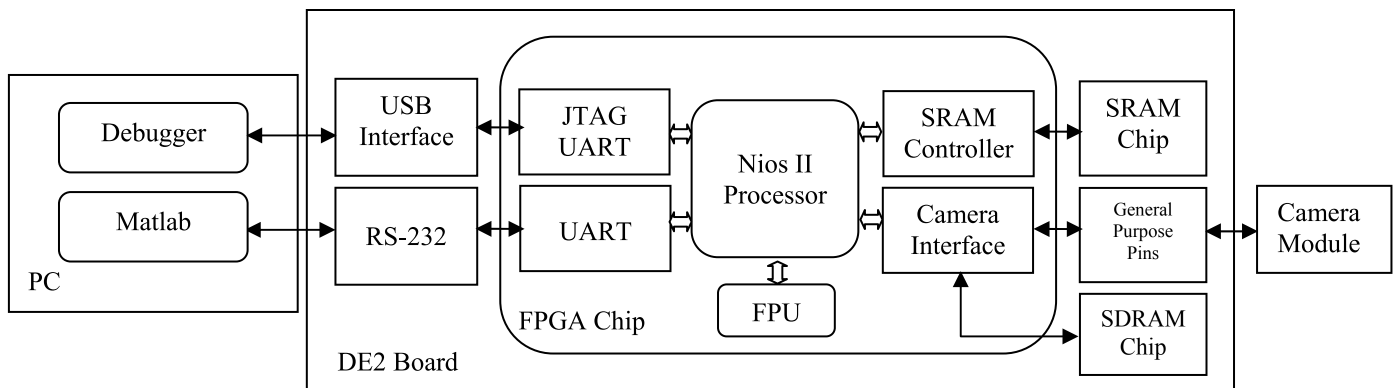


Figure 11: Single Processor Fully Functional System

5.4.2 Software

Both the algorithm and the capture mechanism were merged together to produce the software for this stage. Additionally the UART was used to send debugging data to the PC. Matlab was used as a monitoring platform for the system.

5.4.3 Optimizations via Co-Design and FPGA Flexibility

No significant changes were made other than integrating the UART hardware and adding software drivers and data validation mechanisms for data transmission.

5.5 Multiprocessor System Implementation

The next step is to build a multiprocessor system to accelerate the whole process and produce a system that can perform fast enough to meet the real-time constraints of the application. The multiprocessor system must be able to handle video stream or multiple consecutive frames in order to reach a decision about driver's state of vigilance. This will be the focus of the next chapter.

5.6 Conclusion

In this chapter system development stages and design decisions were covered. It is clear how FPGA flexibility and usage of co-design and dynamic partitioning methods made the task much easier. They allowed the system to be built in a limited time while still maintaining satisfactory performance.

FPGA flexibility and short design cycle time allowed assessing different design alternatives by building rapid prototypes using IP cores and testing different configurations. Moreover hardware/software co-design and dynamic partitioning helped reaching initial and simple design decisions that achieved an overall performance gain in a short time.

It is clear that the approach adopted helped to achieve the goals of facilitating design exploration for the simple case of using a single processor. The advantages of the method will be more apparent when it is exploited to explore the complicated set of design decisions for a multiprocessor system in the next chapter.

Chapter 6
Multiprocessor System Development

6.1 Introduction

This chapter highlights the multiprocessor system development stages and design decisions. After reaching a fully functional system based on a single processor, the need for using a multi-core platform is justified. The long execution time of the algorithm developed to process a single frame on the Nios II processor is causing multiple frames to be lost. This fact renders the system unable to take a decision in real-time about the driver vigilance.

The use of hardware/software co-design as a tool and FPGA flexibility as an asset will form the basis for tackling the problem and exploring the design space as mentioned in chapter 2. The design approach effectiveness and ability to allow the designer to reach a nearly optimal solution in limited time will be assessed by the end of this chapter.

6.2 Design Objectives

Our objective is to develop a system that is able to capture the maximum number of frames possible and process them using the algorithm developed. Moreover, the results obtained after processing each frame should be collected and a final decision about the driver vigilance state should be taken. The frame sequence must also be preserved since it is required to determine the duration of eye closure. This duration is the parameter that reflects whether the driver is vigilant or not.

These objectives will be achieved by using multiple Nios II cores integrated on the FPGA configurable fabric. The cores will collaborate to process the data and produce the required result.

6.3 Design Requirements

Building a multiprocessor system is a non-trivial task that not only requires significant design effort, but also dictates a list of hardware and software requirements. In the following

subsections the different requirements are highlighted. Under each requirement availability and limitations are discussed.

6.3.1 Hardware Requirements

Multiprocessor platforms are characterized by a complex hardware nature. This complexity introduces various requirements that must be satisfied.

6.3.1.1 Area on Chip

The FPGA chip must be able to house more than one processor. So the number of available logic elements must be large enough in order to fit the required number of cores.

The Altera Cyclone II EP2C35 FPGA chip was used for this research and it consists of around 35,000 logic elements. The Nios II full core equipped with a hardware multiplier and a floating point unit uses around 5,000 logic elements. So area is not a limitation for this design, because multiple cores can be built along with helper hardware on the same chip. Helper hardware and interfaces use around 4,000 logic elements. So theoretically we can use six cores on the chip. Practical factors such as ability to place and route the hardware on chip can lower this number.

6.3.1.2 Memory

Each processor in the system requires the presence of program and data memory. Also extra space is required for heap and stack during execution. So the amount of memory needed is directly proportional to the number of cores used in the system.

The DE2 board has only two read and write memories (8MB SDRAM and 512KB SRAM). The previous single core fully functional design used the 512 KB SRAM chip as its memory. It used around 300KB for program and initialized data and the rest for heap and stack. Thus this memory chip cannot be shared with any additional cores. And the SDRAM chip was used to buffer the image data produced by the camera module.

This limitation jeopardized the whole realization of the system at hand. So to work around this, the design was modified. Using FPGA flexibility an SRAM controller was built from

scratch and interfaced to the camera module and the processor. The SRAM only stored the green color component that is used as greyscale directly as mentioned earlier. And thus the 512KB could fit the image successfully. Then the processor used the SDRAM 8MB chip as its program and data memory. Figure (12) shows the modified system.

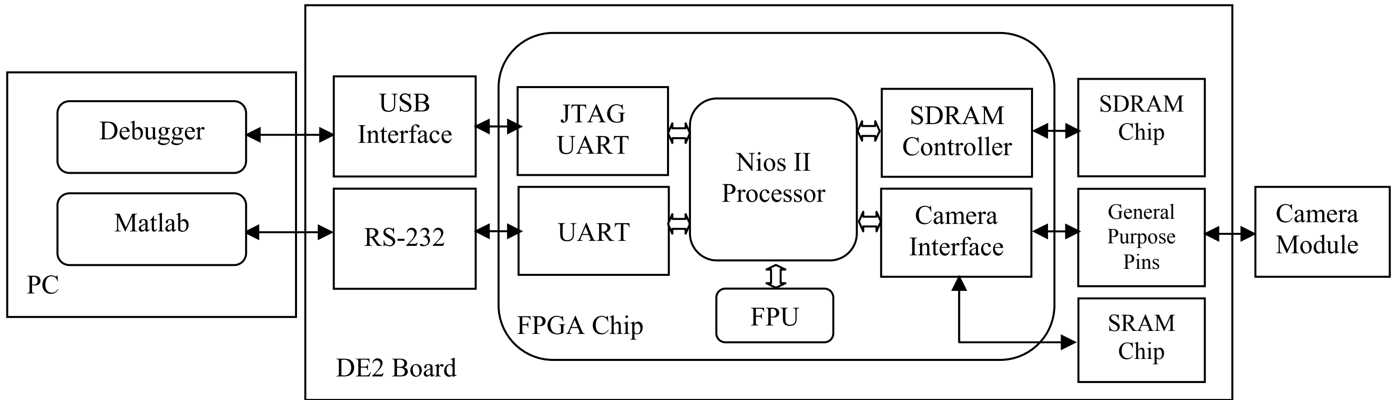


Figure 12: Single Processor Modified Fully Functional System

Sharing the 8MB chip between multiple cores is now possible allowing up to 16 cores to have enough memory to operate. Still the number of cores is limited by available area on chip as indicated in the previous subsection.

6.3.1.3 Communication and Synchronization Logic

The cores added should be able to synchronize in order to access shared resources (data memory or other peripherals). Also they may require communicating with each other to exchange data.

This is achieved via different multiprocessor architectures and topologies. The interconnections and extra hardware required for synchronization and/or communication can be built on the FPGA configurable fabric.

6.3.2 Software Requirements

The software requirements by multiprocessor systems can be divided into hardware and software design tools requirements and additional programming requirements.

6.3.2.1 Hardware and Software Design Tools

- Altera Quartus II 7.2 is used during this research for VHDL/Verilog compilation, analysis and synthesis. It is also used for fitting the hardware design on the Cyclone II FPGA chip and generating the programming file.
- Altera SOPC Builder (System on a Programmable Chip Builder) is used to generate the customized Nios II cores and integrating various interfaces IPs. It is also able to generate bus arbiters and connection logic between the processors and the interfaces using the Altera Avalon bus standard.
- Eclipse Nios II Integrated Development Environment is used for building software to target the different cores. Also it is able to monitor and debug the software running on multiple cores simultaneously.

6.3.2.2 Additional Programming Requirements

- Processors communication and synchronization functions
- Mutual exclusion mechanisms to access shared resources

6.4 Cores Synchronization and Communication

After ensuring that the required hardware and software resources are available, we need to tackle the design problem of building cores synchronization and communication logic. The first step to accomplish this task is analyzing the shared resources and deciding on the implementation method for cores synchronization and communication. The synchronization mechanism should guarantee mutual exclusion of accessing shared resources. Communication mechanism should allow the cores to exchange data when needed.

6.4.1 Shared Resources Analysis

It is easy to notice that the most important resource that should be shared across the processing cores is the camera module. All cores must be able to access the image buffering memory (the 512 KB SRAM in our case). They should be able to retrieve image data by accessing the control, address and data buses of the memory. Mutual exclusion should be guaranteed so that those buses do not get contaminated.

Moreover as mentioned earlier they all need to share the SDRAM and use it as program and data memory. Each processor should be assigned an exclusive location in memory that should not be accessible by the rest of the cores so that program and data corruption is prevented.

This information leads to the next step of determining how to build synchronization and mutual exclusion mechanisms in both hardware and software. FPGA flexibility coupled with hardware/software co-design allows the exploration of different mechanisms and evaluating their performance and reliability.

6.4.2 Cores Synchronization and Mutual Exclusion

Different techniques were evaluated for the system at hand. Reliability is the most important parameter in determining the best candidate.

The following techniques were built and evaluated:

6.4.2.1 Synchronization via a Mutex Core

The first solution tested was to use a mutex core offered by Altera for use in multiprocessor systems. The mutex core provided means for mutual exclusion, so each core has to atomically test and lock the mutex first before accessing the image buffer and release the mutex when done.

The core delivered satisfactory results when it comes to ensuring mutual exclusion but it was unable to provide any control or information about the sequence in which the processors access the camera module. Our application requires controlling the sequence, or at least knowing the sequence in which the camera module was accessed, because the final decision about the driver state of vigilance is determined by checking the eyes state in consecutive frames. In the case of using a mutex core, any processor can access the camera when the mutex core is unlocked thus sequence information was lost.

A software work around was used in which each processor monitors the mutex core to know the current owner and then access the memory next if it is its turn to do so according to

a predefined sequence. Yet this method is unreliable and can cause a dead lock, since the next processor maybe busy when the previous owner was accessing the camera module.

Hence this method proved unreliable and unable to provide the application with the required deterministic sequence of operation.

6.4.2.2 Synchronization via Handshaking

In this technique a token based method is used along with a setup where each core is connected to the next using a strobe and acknowledge signal as shown in figure (13) below.

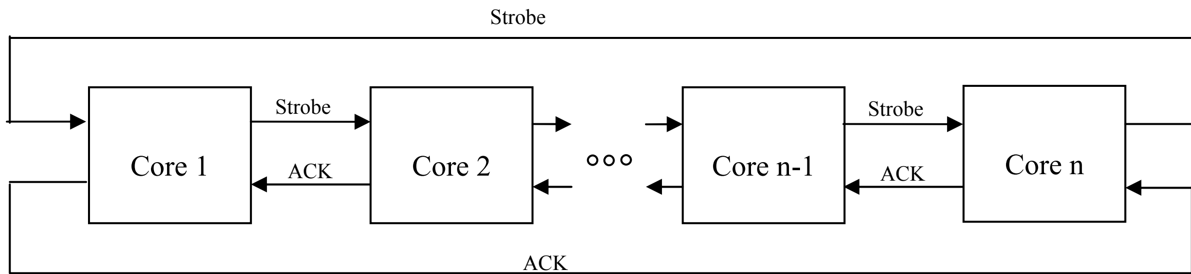


Figure 13: Synchronization via Handshaking

The first core initially possesses the token and so it has the right to access the camera module. After it is done it then signals the next processor by raising a strobe signal high. The next processor then acknowledges that it received the token by raising the acknowledge signal high. The first processor then lowers the strobe signal. Each two consecutive cores in the setup behave the same way to pass the token (or the right to access the camera) till the last core is reached. The last core hands the token back to the first processor the same way using the loop connection shown in figure (13).

This method is able to achieve the mutual exclusion condition since the token cannot be possessed except by one processor at a certain time. It is worth mentioning that the token is just a software variable and this condition must be ensured when writing each of the cores program. This method is able to realize a deterministic sequence of operation since the system makes sure that the camera module is accessed in turns starting with core 1 and ending with core n then looping over again according to the wiring diagram shown in figure (13).

The only drawback of this technique is that the previous processor has to wait for the acknowledge signal from the next one, halting its operation and wasting CPU cycles. So a better technique was implemented using interrupts.

6.4.2.3 Synchronization via Interrupts

This method is a modified version of the previous method. It uses the same token passing mechanism but instead of using a strobe and acknowledge line it uses interrupts as shown in figure (14) below.

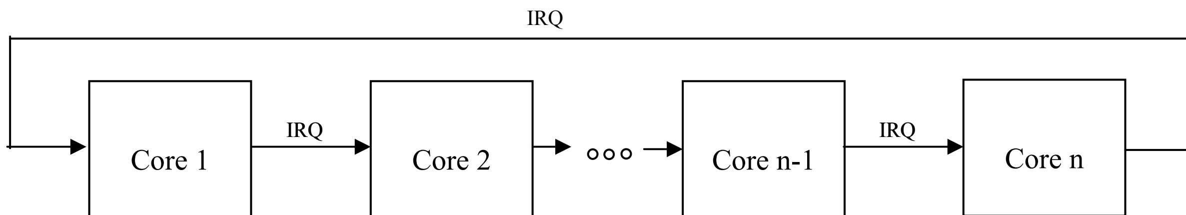


Figure 14: Synchronization via Interrupts

Similar to the previous method core 1 initially has the token and when it is done accessing the camera buffer it signals an interrupt to the next processor. This method ensures mutual exclusion and a deterministic access sequence. Moreover, when a CPU is done it just signals the interrupt and continues processing. The next processor will receive the interrupt and uses the token when it is ready to do so without tying the previous processor up.

Thus in this research work the former synchronization method was selected because it can satisfy all the requirements.

6.4.3 Cores Program and Data Memory Management

As mentioned earlier the SDRAM chip is shared by all cores to act as program and data memory. Luckily the Avalon bus arbiter can prevent simultaneous access to the SDRAM chip by multiple cores. And using different addresses that are separated by 1 MB to link the program of each core, we ensure that the processors will not access each other's memory space. So corruption is prevented using these two measures.

6.4.4 Cores Communication

According to the previous model of synchronization and the fact that the only input for the algorithm developed is a single frame, there is no data or instructions dependence between cores that requires them to communicate. But it is worth mentioning that to deduce the final result the output from each core is needed.

Cores can operate separately because each one can capture a single frame and start processing it. And then the final results can be collected from all cores by a simple hardware unit to determine the period of eye closure and produce a final decision about the driver's state of vigilance.

This dictates a single program multiple data (SPMD) multiprocessor architecture. Since the data is different (different frames) but the program executed on each frame is the same. The model is depicted in figure (15) below.

Of course, analysis can be applied on the algorithm itself to divide the processing of each frame on more than one processor, yet this initial model can exploit the apparent parallelism that exists because of independence between frames and also accessing the camera buffer is limited by a single processor at any instance of time.

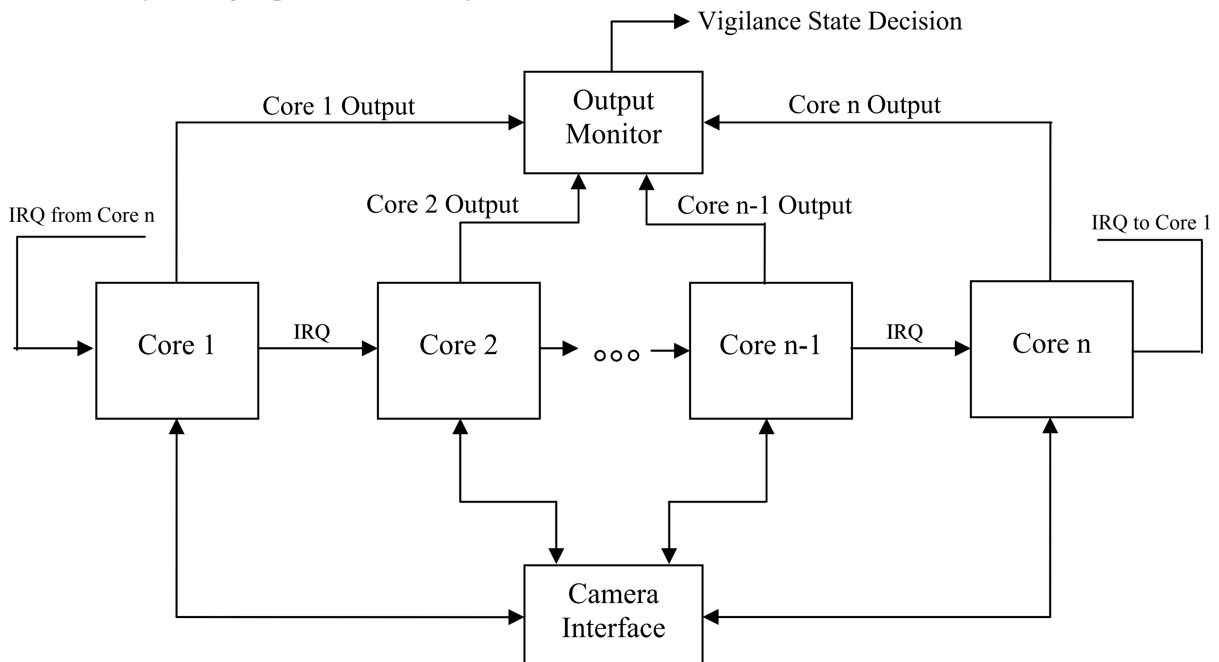


Figure 15: Synchronization and Communication Model

6.5 Design Space Exploration

By now we settled down on the cores communication and synchronization methods and the multiprocessor architecture. The next step is to determine the optimal number of cores that can satisfy the real-time constraints of our application.

In order to reach that optimal number of cores and to verify whether the resulting system can satisfy the application constraints given the decided architecture, we have to perform a software timing analysis.

The timing analysis gives the designer information about the execution time of the created program with different levels of granularity. Timing analysis can be done on the instruction level (fine grain analysis), function level or task level (coarse grain analysis).

In the upcoming section, the timing extraction method deployed in this research will be discussed along with the results obtained. Then, an analysis will be applied on the timing information extracted in order to reach an optimal decision about the number of cores that need to be used. The criteria of optimization will be area on chip and performance.

6.5.1 Software Timing Extraction

For timing extraction a performance timer hardware core was used in the system. It counts the number of clock cycles elapsed since calling a timer start command and till a timer stop command is issued. The number of cycles can be easily converted to time in seconds given that the operating frequency is known.

Using this technique the frame capture time and processing time were calculated (coarse grain timing extraction). Results are shown in table (1) below:

Operation	Elapsed Time
Frame Capture	1.6 sec
Frame Processing	23 sec

Table 1: Timing Analysis

6.5.2 Software Timing Analysis

The frame capture time shown in table (1) is constrained by the processor ability to retrieve data from the camera module so this timing is lowest achievable time. Yet this capture time is satisfactory for the application, since higher frame processing rate would produce redundant data.

As for the frame processing time, it is the time taken to run the developed algorithm and obtain the binary result of whether the eye is opened or closed.

The system will possess multiple cores and each core can handle a different frame. So the number of cores should ensure that frames are always retrieved from the camera module continuously in steady state. Given the token passing synchronization architecture discussed earlier that means that the token should be returned back to the first processor as soon as it is done processing to ensure the continuous capture condition is satisfied. This is illustrated by figure (16) below.

Examining the figure it can be easily deduced that the total number of cores required to satisfy the continuity condition is:

$$n = \text{ceiling}\,(\text{Processing time}/\text{Capture time}) + 1 \quad (1)$$

Using the numbers from table (1):

$$n = \text{ceiling}\,(23/1.6) + 1$$

$$n = 16$$

The available area allows only six cores, thus we have to resort to further timing analysis and deploying hardware accelerators. This will decrease the processing time and lower the number of required cores.

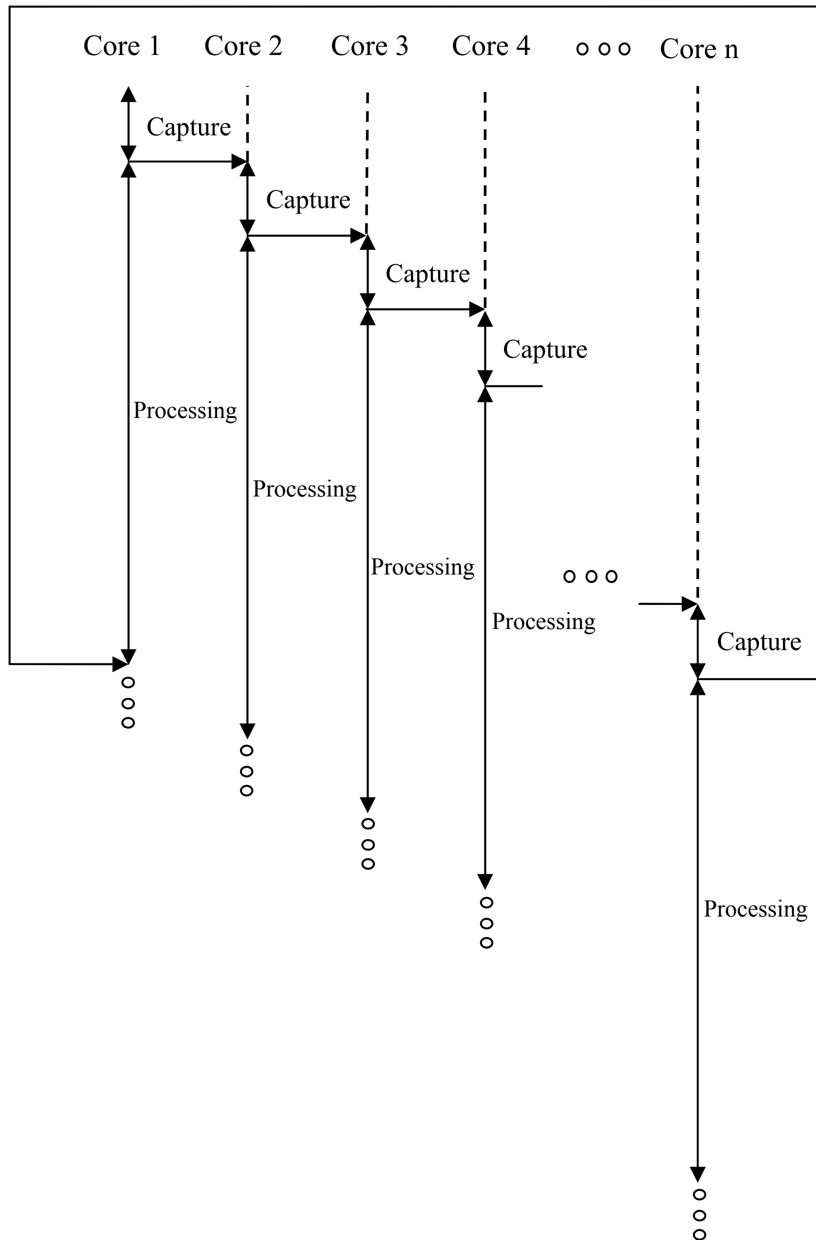


Figure 16: Timing of a System Ensuring Continuous Frame Capturing

6.5.3 Building Hardware Accelerators

In the previous section, the optimal number of cores was calculated and given the available resources it is impossible to build such a system. In this section the possibility of accelerating the processing time by deploying hardware acceleration will be discussed. It is essential to

mention that the hardware accelerator built will require area on chip and thus area constraints should be integrated in our optimization problem.

6.5.3.1 Area Considerations

Currently the area usage can be modeled using equation (2) below:

Area on chip = Number of processors * (Area of each processor + Extra peripherals)
+
Area of helper hardware (2)

Where extra peripherals stand for processor interfaces to each other and custom hardware blocks (for instance floating point unit). Helper hardware consists of shared resources such as the camera interface, memory interface, output monitoring logic and arbitration logic.

Using post-fitting data for a single core system the following logic elements usage in table (2) is observed:

Hardware Block	Sub blocks	Area (in LE)
Processor	CPU Core	2663
Extra Peripherals	FPU Unit	2009
	Inter-processor Communication	14
	Sub Total	2023
Helper Hardware	Camera Interface	655
	Memory Interface	875
	Output Monitoring	1
	Arbitration	2147
	Sub Total	3678

Table 2: Area Usage of Each Hardware Block

After adding the hardware acceleration block to each processor the total area can be calculated using equation (3) below:

Area on chip = Number of processors * (Area of each processor + Extra peripherals + Area of accelerator) + Area of helper hardware (3)

Hence in the process of building the accelerator even though it is going to decrease the processing execution time and thus the number of required cores, still hardware area on chip should not exceed the number of available logic elements (35,000 LEs).

Currently the maximum number of cores that can fit on the chip is six cores using equation (2) above.

6.5.3.2 Hardware Accelerator Design

Now, after laying down the area constraint and the required number of cores (which is inversely proportional to execution time), the hardware accelerator should minimize the execution time leading to lowering the number of cores required. On the other hand the area saved by lowering the number of cores should not be wasted by using a bulky acceleration block which might lead to a dead end.

The design process proceeded first by applying a fine grain timing analysis (on the function level). Then second step is selecting the function that consumes most of the processing time and implementing a hardware block able to perform this computation. From there the hardware accelerator is deployed and functionality is verified. Finally area information is extracted and new maximum number of cores is calculated based on the new area usage information. Also new timing is extracted and required number of cores is obtained.

1- Timing Analysis on the Function Level:
   Taking time measurements for various functions in the algorithm, the edge detection function proves to be the most time consuming part. The 2d convolution process involved consumes 75% of the processing time (~17 sec).

2- Hardware Acceleration Implementation:
   In this step an industry proven automatic ANSI C to hardware creation tool was used. Interested reader can refer to [10] in order to know more details about Altera Nios II C2H tool. One difficulty was faced is that the tool does not support floating point variables which were used in the convolution process. Thus scaling and rounding was applied to convolution kernel on the Matlab model and similar edge detection final results were obtained. The scaling and rounding method was then applied to the code and the C2H tool was able to generate custom hardware that applies 2d convolution.

3- Hardware Accelerator Deployment:
   In this step the hardware accelerator was deployed and the system was tested to ensure that results did not change after replacing the software function with hardware accelerator driver calls.

4- Area Information Extraction:
   Table (3) below summarizes the area usage for the single processor system after deploying the hardware accelerator. Notice that the FPU was removed because it is no longer needed and it was replaced with the 2d convolution hardware accelerator.

Hardware Block	Sub blocks	Area (in LE)
Processor	CPU Core	2663
Extra Peripherals	Hardware Acceleration Unit	1578
	Inter-processor Communication	14
	Sub Total	1592
Helper Hardware	Camera Interface	655
	Memory Interface	875
	Output Monitoring	1
	Arbitration	2147
	Sub Total	3678

Table 3: Area Usage Summary after Adding Hardware Accelerator

Replacing the FPU with the hardware accelerator decreased the area required by extra peripherals thus overall the area decreased and the number of cores that can fit on the chip using equation (3) is now seven cores.

So the addition of the accelerator actually had a positive effect on the area usage of the previous design. But it is fair to mention that the FPU could be removed from the previous implementation using the scaling technique. Thus maximum number of cores could have been 11 cores. Still even using 11 cores without a hardware accelerator cannot satisfy the continuous capturing condition (since 16 cores are required). The actual benefit of the accelerator should be minimizing the processing time.

5- Timing Information Extraction with HW Accelerator:
   The results obtained after applying timing extraction again with coarse granularity with hardware accelerator deployed are summarized in table (4) below:

Operation	Elapsed Time
Frame Capture	1.6 sec
Frame Processing	9.3 sec

Table 4: Timing Analysis after Adding Hardware Accelerator

A dramatic drop in frame processing time can be noticed from 23 to 9.3 seconds which is around 60 %.
Now reusing equation (1) to calculate the required number of cores to satisfy continuous the frame capturing condition, we obtain seven cores which can be fitted on the chip.

Thus from the first iteration adding a hardware accelerator block a feasible design was obtained.

6.6 Final Multiprocessor Design

The final system designed consists of seven cores and satisfies the continuous capturing constraints. The capture time is sufficient for the application, when two consecutive frames results are both eyes closed the driver is considered drowsy.

Each processor is equipped with a hardware acceleration block that decreases processing time significantly. Also all processors share a single camera module and memory.

The system uses token passing synchronization method to ensure mutual exclusion and deterministic sequence of operation. Output is forwarded to a simple dedicated output monitoring device that reaches a final decision by detecting a pattern of two consecutive closed eyes frames.

Finally the system uses 33463 logic elements out of the 35000 available on chip which can be further reduced by using lighter processing cores and removing debugging logic for the production cycle. The final design is shown in figure (17) below.

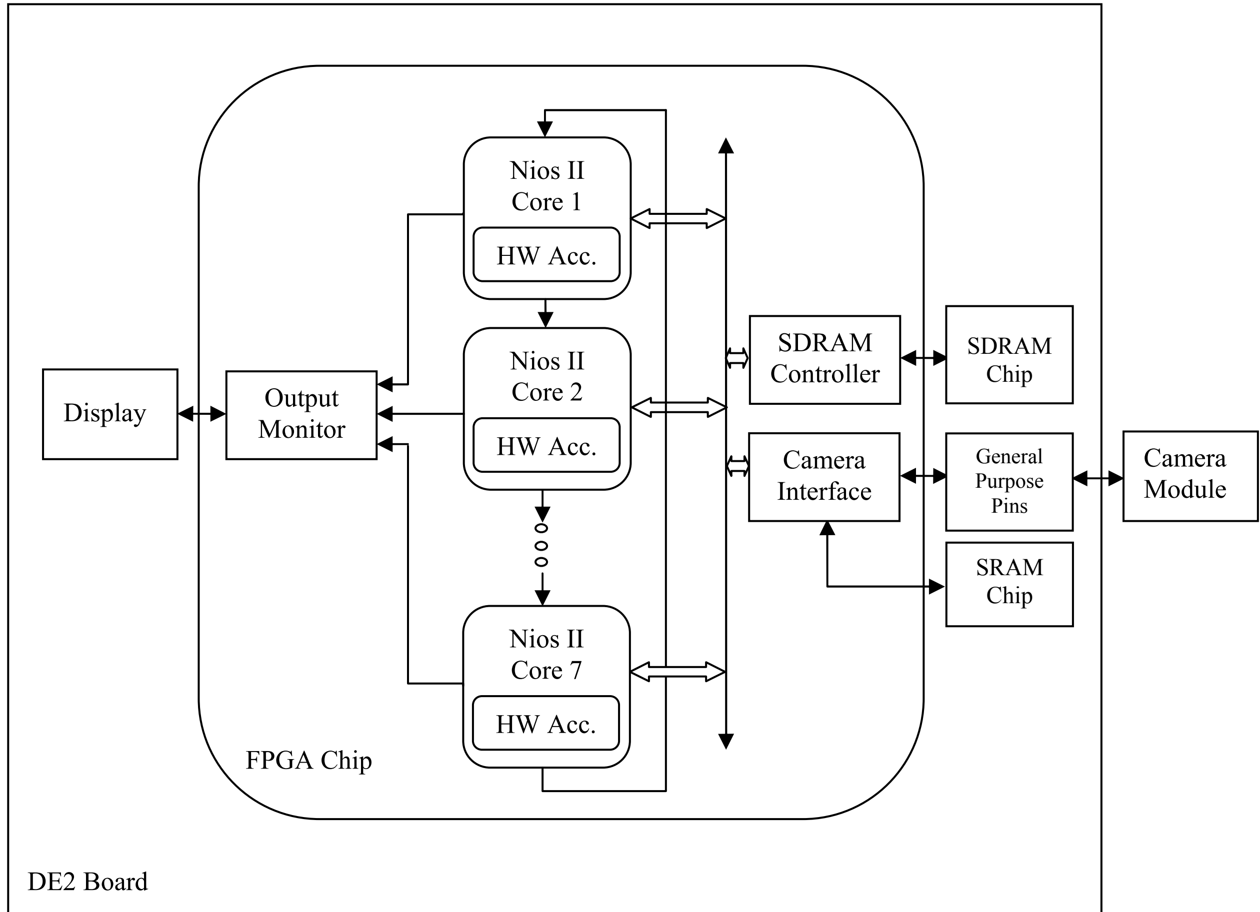


Figure 17: Full Multiprocessor System

6.7 Conclusion

In this chapter the different design requirements for a multiprocessor system were discussed. Moreover, the practical system prototype at hand was used to demonstrate those requirements. The problem of choosing the multiprocessor synchronization and communication architectures was practically tackled for the target system. Different methods were evaluated using the flexibility of FPGA technology and the best candidate was picked based on the target application.

This chapter also covered different design space exploration aspects. We had constrained area and target performance requirements that need to be satisfied. Optimal number of components could not fit on the available area. Thus again FPGA technology flexibility and

leading edge design tools were used to apply dynamic partitioning and converting a time consuming software function to an equivalent and much faster hardware block. The steps of this process were discussed in details.

By the end of the chapter, the final design was described and it successfully complies with the area constraints and satisfies the required performance for the application.

Chapter 7
Results and Future Work

7.1 Introduction

In this chapter all the proposed methods and suggested approaches will be evaluated according to the results obtained in the final system.

Each one of the three main aspects of this research will be discussed in a section highlighting the results associated, results indications and future research direction.

7.2 Driver's Vigilance Monitoring Algorithm

In this research a new vigilance monitoring algorithm based on visual cues was implemented. It mainly depended on extracting the state of the eye from a set of consecutive frames and then reaching a decision based on eye closure duration.

The main advantage of the algorithm proposed is that it is light and does not depend on machine learning techniques. So it does not depend on the vehicle driver facial features and does not require training.

However, the algorithm has some shortcomings; for instance it neither can handle tilted faces nor drivers wearing sun glasses. The problem of tilted face was overcome using the fact that only a vigilant driver is able to maintain such a posture. Another solution was to rotate the face to compensate for the tilt, yet that would increase the processing load significantly. As for the sunglasses problem, it requires special hardware to deal with it (which was not discussed in this research work).

Results obtained were satisfactory using the Grimace Face Database [45] and when operating the system on my face. On operating the device, a false positive of eye closure must take place two consecutive times in order to get an alarm, which decreases the probability of failure.

Future work on that algorithm consists of making it more robust and adapting it to different lighting conditions. Extra processing using special hardware for night mode and for drivers wearing sunglasses could also be integrated.

7.3 Proposed Multiprocessor System Design Methodology

In this work, a systematic design methodology for multiprocessor systems design was proposed. The methodology depends on FPGA technology to reinforce it with flexibility and cheap short design iterations.

The proposed method was created to be generic, yet it was only verified by building a system targeting our driver's vigilance monitoring application. It achieved satisfactory performance in a short time and allowed for optimization throughout the different stages of design as highlighted in chapter 5 and chapter 6.

Future work on this method consists of assessing its ability to implement any required multiprocessor system. This can be done by trying to utilize it to build different systems targeted for different applications. Additionally, its results need to be compared against other methods targeting FPGA technology. Yet, to accomplish these goals a dedicated research should be conducted.

7.4 Implemented Driver's Vigilance Monitoring Device

By the end of this research work a multiprocessor driver's vigilance monitoring device was implemented. The device was based on the proposed algorithm and was built using the proposed methodology. Throughout the development process hardware/software co-design coupled with FPGA flexibility allowed easier design space exploration and reaching satisfactory performance in a relatively short time.

The system satisfies the functional and performance requirements constrained under area and clock frequency limits allowed by the FPGA chip.

Future work on the device consists of incorporating more capabilities to it such as using more visual cues or integrating other sensors to increase accuracy. Also a practical step

towards marketing the device is to migrate it from the development board to a custom board with only the required peripherals.

7.5 Conclusion

We can conclude the work done throughout the research in the following points:

- FPGA technology proved to be a very powerful prototyping tool for complex multiprocessor systems design
- Driver vigilance algorithm developed was able to achieve satisfactory results
- Using multiple processors in the device enabled it to provide performance that meets real-time constraints
- The proposed design methodology was applied throughout the whole process and proved to be effective
- The flexible FPGA technology coupled with hardware/software co-design provided means to explore the design space and reach decisions that satisfy the design constraints with minimum time investment and cost

References

[1] Compton, K., and Hauck, S.: *Reconfigurable Computing: A Survey of Systems and Software*. ACM Computing Surveys, Vol. 34, No. 2, pp. 171-210, June 2002.

[2] Altera Corp., Quartus II Version 7.2 Handbook.

[3] Lysecky, R. and Vahid, F.: A Study of the Speedups and Competitiveness of FPGA Soft Processor Cores using Dynamic Hardware/Software Partitioning. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1* (March 07 - 11, 2005). Design, Automation, and Test in Europe. IEEE Computer Society, Washington, DC, 18-23.

[4] Xilinx Corp., MicroBlaze Processor Reference Guide.

[5] Altera Corp., Nios II Processor Reference Handbook.

[6] De Micheli, G.; Gupta, R. Hardware/software co-design. Proceedings of the IEEE, 85(3), March, 1997. pp 349-365.

[7] Stechele, W.; Carcel, L.A.; Herrmann, S.; Simon, J.L., "A coprocessor for accelerating visual information processing," *Design, Automation and Test in Europe, 2005. Proceedings* , vol., no., pp. 26-31 Vol. 3, 7-11 March 2005

[8] Coric, S., Leeser, M., Miller, E., and Trepanier, M. 2002. Parallel-beam backprojection: an FPGA implementation optimized for medical imaging. In *Proceedings of the 2002 ACM/SIGDA Tenth international Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA, February 24 - 26, 2002). FPGA '02. ACM, New York, NY, 217-226.

[9] Yu, J., Lemieux, G., and Eagleston, C. 2008. Vector processing as a soft-core CPU accelerator. In *Proceedings of the 16th international ACM/SIGDA Symposium on Field Programmable Gate Arrays* (Monterey, California, USA, February 24 - 26, 2008). FPGA '08. ACM, New York, NY, 222-232.

[10] Altera Corp., Nios II C2H Compiler User Guide.

[11] Sheldon, D., Kumar, R., Lysecky, R., Vahid, F., and Tullsen, D. 2006. Application-specific customization of parameterized FPGA soft-core processors. In *Proceedings of the 2006 IEEE/ACM international Conference on Computer-Aided Design* (San Jose, California, November 05 - 09, 2006). ICCAD '06. ACM, New York, NY, 261-268.

[12] Yiannacouras, P., Steffan, J. G., and Rose, J. 2006. Application-specific customization of soft processor microarchitecture. In *Proceedings of the 2006 ACM/SIGDA 14th*

international Symposium on Field Programmable Gate Arrays (Monterey, California, USA, February 22 - 24, 2006). FPGA '06. ACM, New York, NY, 201-210.

[13] Lysecky, R. and Vahid, F. 2005. A Study of the Speedups and Competitiveness of FPGA Soft Processor Cores using Dynamic Hardware/Software Partitioning. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1* (March 07 - 11, 2005). Design, Automation, and Test in Europe. IEEE Computer Society, Washington, DC, 18-23.

[14] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, K. Yelik. "The Landscape of Parallel Computing Research: A View from Berkeley". Technical report, EECS Department, University of California at Berkeley, UCB/EECS-2006-183, December, 2006.

[15] David Geer, "For Programmers, Multicore Chips Mean Multiple Challenges," *Computer*, vol. 40, no. 9, pp. 17-19, Sept., 2007.

[16] Intel Corporation, www.intel.com

[17] AMD Corporation, www.amd.com

[18] Altera Corporation, www.altera.com

[19] Xilinx Corporation, www.xilinx.com

[20] Ou, J. and Prasanna, V. K. 2006. Design space exploration using arithmetic-level hardware--software cosimulation for configurable multiprocessor platforms. *Trans. on Embedded Computing Sys.* 5, 2 (May. 2006), 355-382.

[21] Thompson, M., Nikolov, H., Stefanov, T., Pimentel, A. D., Erbas, C., Polstra, S., and Deprettere, E. F. 2007. A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In *Proceedings of the 5th IEEE/ACM international Conference on Hardware/Software Codesign and System Synthesis* (Salzburg, Austria, September 30 - October 03, 2007). CODES+ISSS '07. ACM, New York, NY, 9-14.

[22] A. Kumar, A. Hansson, J. Huisken, H. Corporaal, "An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip," *date*, p. 33, 2007 Design, Automation & Test in Europe Conference & Exhibition, 2007.

[23] Oliveira, M. F., Brião, E. W., Nascimento, F. A., and Wagner, F. R. 2007. Model driven engineering for MPSOC design space exploration. In *Proceedings of the 20th Annual*

Conference on integrated Circuits and Systems Design (Copacabana, Rio de Janeiro, September 03 - 06, 2007). SBCCI '07. ACM, New York, NY, 81-86.

[24] Y. Elzohairy, "Fatal and Injury Fatigue-Related Crashes on Ontario’s Roads: A 5-year Review", Working Together to Understand Driver Fatigue: Report on Symposium Proceedings, February 2008.

[25] D. J. Beirness, H. M. Simpson, K. Desmond. "The Road Safety Monitor 2004 Drowsy Driving", The Traffic Injury Research Foundation, Ottawa, Ontario, Canada, February 2005.

[26] A.V. Desai, M.A. Haque, Vigilance monitoring for operator safety: A simulation study on highway driving, Journal of Safety Research Volume 37, Issue 2, 2006, Pages 139-147.

[27] Santana Diaz, A.; Jammes, B.; Esteve, D.; Gonzalez Mendoza, M., "Driver hypovigilance diagnosis using wavelets and statistical analysis," *Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on* , vol., no., pp. 162-167, 2002.

[28] Gonzalez-Mendoza, M.; Santana-Diaz, A.; Hernandez-Gress, N.; Titli, A., "Driver vigilance monitoring, a new approach," *Intelligent Vehicle Symposium, 2002. IEEE* , vol.2, no., pp. 358-363 vol.2, 17-21 June 2002.

[29] Khalifa, K.B.; Bedoui, M.H.; Raytchev, R.; Dogui, M., "A portable device for alertness detection," *Microtechnologies in Medicine and Biology, 1st Annual International, Conference On. 2000* , vol., no., pp.584-586, 2000.

[30] Schmidt EA, Kincses WE, Schrauf M, Haufe S, Schubert R, Curio G. Assessing Drivers' Vigilance State During Monotonous Driving. Proceedings of the Fourth International Symposium on Human Factors in Driving Assessment, Training, and Vehicle Design, pp. 138-145, 2007.

[31] Batista, J., "A Drowsiness and Point of Attention Monitoring System for Driver Vigilance," *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE* , vol., no., pp.702-708, Sept. 30 2007-Oct. 3 2007.

[32] D'Orazio, T.; Leo, M.; Spagnolo, P.; Guaragnella, C., "A neural system for eye detection in a driver vigilance application," *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on* , vol., no., pp. 320-325, 3-6 Oct. 2004.

[33] D'Orazio, T.; Leo, M.; Distante, A., "Eye detection in face images for a driver vigilance system," *Intelligent Vehicles Symposium, 2004 IEEE* , vol., no., pp. 95-98, 14-17 June 2004.

[34] He Huang; You-Sheng Zhou; Fan Zhang; Feng-Chen Liu, "An optimized eye locating and tracking system for driver fatigue monitoring," *Wavelet Analysis and Pattern Recognition, 2007. ICWAPR '07. International Conference on* , vol.3, no., pp.1144-1149, 2-4 Nov. 2007.

[35] McCall, J. and Trivedi, M. M. 2006. Driver monitoring for a human-centered driver assistance system. In *Proceedings of the 1st ACM international Workshop on Human-Centered Multimedia* (Santa Barbara, California, USA, October 27 - 27, 2006). HCM '06. ACM, New York, NY, 115-122.

[36] Bao, D., Yang, Z., and Song, Y. 2007. Projection function for driver fatigue monitoring with monocular camera. In *Proceedings of the 2007 ACM Symposium on Applied Computing* (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY, 82-83.

[37] Ji, Q. and Yang, X. 2001. Real Time Visual Cues Extraction for Monitoring Driver Vigilance. In *Proceedings of the Second international Workshop on Computer Vision Systems* (July 07 - 08, 2001). B. Schiele and G. Sagerer, Eds. Lecture Notes In Computer Science, vol. 2095. Springer-Verlag, London, 107-124.

[38] Z. Zhu, Q. Ji, and P. Lan, ”Real Time Non-intrusive Monitoring and Prediction of Driver Fatigue,” IEEE transactions on vehicular technology, vol. 53, pp. 657-662, July 2004.

[39] Ji, Q. and Yang, X. 2001. Real Time Visual Cues Extraction for Monitoring Driver Vigilance. In *Proceedings of the Second international Workshop on Computer Vision Systems* (July 07 - 08, 2001). B. Schiele and G. Sagerer, Eds. Lecture Notes In Computer Science, vol. 2095. Springer-Verlag, London, 107-124.

[40] Dikkers, H.J.; Spaans, M.A.; Datcu, D.; Novak, M.; Rothkrantz, L.J.M., "Facial recognition system for driver vigilance monitoring," *Systems, Man and Cybernetics, 2004 IEEE International Conference on* , vol.4, no., pp. 3787-3792 vol.4, 10-13 Oct. 2004.

[41] Bergasa, L.M.; Nuevo, J.; Sotelo, M.A.; Barea, R.; Lopez, M.E., "Real-time system for monitoring driver vigilance," *Intelligent Transportation Systems, IEEE Transactions on* , vol.7, no.1, pp. 63-77, March 2006.

[42] Bergasa, L.M.; Nuevo, J., "Real-time system for monitoring driver vigilance," *Industrial Electronics, 2005. ISIE 2005. Proceedings of the IEEE International Symposium on* , vol.3, no., pp. 1303-1308 vol. 3, 20-23 June 2005.

[43] Amditis, A.; Polychronopoulos, A.; Bekiaris, E.; Antonello, P.C., "System architecture of a driver's monitoring and hypovigilance warning system," *Intelligent Vehicle Symposium, 2002. IEEE* , vol.2, no., pp. 527-532 vol.2, 17-21 June 2002.

[44] The Mathworks Inc. Matlab, http://www.mathworks.com.

[45] Grimace Face Database, Computer Vision Science Research Group, University of Essex, UK, http://cswww.essex.ac.uk/mv/allfaces/grimace.html.

[46] Altera Corp., DE2 Development and Education Board User Manual.

[47] Terasic Technologies, TRDB_D5M 5 Mega Pixel Digital Camera Development Kit.