System on Chip (SoC):

A Real Time Touch Screen System on Programmable Chip

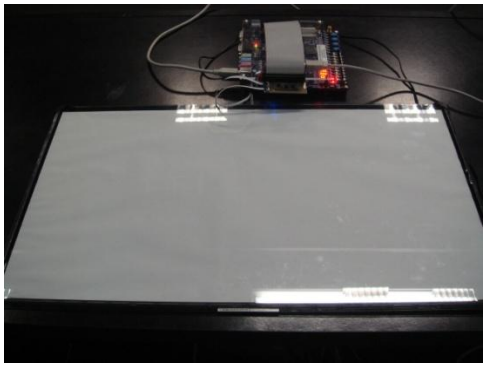
Stephen Xu

A thesis submitted in fulfillment of the requirements for the degree of

Master of Engineering (ME)

In Electrical and Electronic Engineering

Auckland University of Technology, New Zealand



November 2011

School of Engineering

Primary Supervisor: Dr John Collins

Kaore ko au
E kimi ana,
E hahau ana,
I nga pari ra
Piri nga hakoakoa,
E kau oma tera.
Ka toa atu tera
Ka ao mai te ra
Ki tua.
E I a ha-a!

Seeking, searching, peering,
As on those rocky crags
The sea-hawk sits
And watches for his prey,
Soon will the sun
Rise flaming over the world!

Whakaaraara-pa

From Rauparaha's Ngati-Toa warriors

Attestation of Originality

‘I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for qualification of any other degree or diploma of a university or other institution of higher learning, except where certain content is exactly defined in the acknowledgements.’

Stephen Xu

November 2011

Acknowledgement

I am very grateful to both of my academic supervisors: Dr John Collins and Mark Beckerleg. With your patient mentoring, strict requirements and kind encouragement through all these years, I have become a better person with independent research capability, innovative thinking and consistent motivation from inside.

Dr John Collins has provided invaluable guidance and advice on the architectural and directional level with his broad background in embedded system. And Mark Beckerleg has been a technical mentor and friend in the new technology domain with numerous amounts of help and assistance.

I am also very grateful to all my industry supervisors: Simon Bridger, John Newton and Gordon Macdonald. I am grateful for the research platform which is based on unique technology, aiming at investigating and solving real world limitations and having an impact on the outcome. I have benefited a lot from all my supervisors from different perspectives and I would like to express my sincere appreciation again.

Finally, I am very grateful for the help from technicians in the university and engineers inside the company in terms of system understanding, prototype building and trouble shooting. I would like to thank all my friends and family as well since it has been a long and difficult process.

Abstract

This thesis is involved with the investigation, implementation, verification, validation and optimization of a purpose built on-chip solution customized for a real world touch screen application. A Field Programmable Gate Array based application specific controller has been designed and built in this research as a substitute for a general purpose controller to explore the feasibility and capability of meeting the required system performance while maintaining the minimum consumption of system resources. A variety of new mechanisms, approaches and techniques have been evaluated, developed and applied to different design stages at multiple levels to achieve an overall optimized system outcome.

A dedicated optical imaging acquisition system has been developed with a concurrent control mechanism, faster operational speed and lower signal noise; a customized touch information processing unit has been designed to perform edge detection, object positioning, and touch motion indication with low system latency and highly parallelism; and a computer interface has been built to demonstrate the coherent real-time system performance with visualized validation of results. In the optical based touch screen area, this research presents an original and compact on-chip solution with a significant number of algorithm and method improvements in terms of the touch object detection and localization efficiency as well as touch motion analyzing capability.

The system design has been optimized after establishing the desired functionality to minimize logic resource and memory storage consumption, based on a wide range of techniques with a certain amount of architectural restructuring. The overall economic on-chip resource consumption has been achieved in this research with further consideration for migrating the design into a more application specific high integration density chip in the future for large volume manufacture.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
ADC	Analog-to-Digital Converter
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
BBD	Block Based Design
BSE	Binary Search Engine
CPU	Central Processing Unit
DAU	Data Acquisition Unit
DCM	Digital Clock Manager
DSP	Digital Signal Processor
EDU	Edge Detection Unit
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
LED	Light Emitting Diode
LUT	Look up Table
MMU	Memory Management Unit
PBD	Platform Based Design
PC	Personal Computer
PLU	Position Localization Unit
PU	Processing Unit
RAM	Random Access Memory
SAW	Surface Acoustic Wave
SOC	System on Chip
TOC	Touch Screen on Chip
USB	Universal Serial Bus

Chapter 1 Background and Introduction

1.1 Background and Requirements Analysis

Conventional Touch Screen Technologies

Touch screen technology has been developed as a popular interactive component in everyday life to facilitate human-to-machine communication and enrich user experience. There are conventionally four types of touch screen input devices based on different technologies. They have been applied to various applications with their own advantages and disadvantages where:

- Resistive based touch screen solution is relatively cost efficient and supports multiple input means (fingers, gloves, and stylus) but has the drawbacks of image clarity degradation in public environments and the requirement of periodic calibration.
- Capacitive based touch technology is considered durable and reliable when applied in harsh environments (water, dirt and dust) but has disadvantages in input methods capability (most are restricted to finger input) and scalability (the technical difficulty and high cost of applying to large formats).
- Surface Acoustic Wave (SAW) constructed touch device provides high quality image clarity and light transmission however with the same restriction in large scale capability and cost efficiency.
- Infrared oriented touch technology has the unique integration advantage of being a completely sealed in device, but is also regarded as having a relatively poor touch function performance and it is vulnerable to complex environments (contaminants etc).

Optical Imaging Based Touch Screen Technology

The optical imaging based touch screen solution has emerged in recent years as a new competitive technology which has its own distinct characteristics leading to a significant variety of advantages over the aforementioned traditional approaches.

In general, it is a compact and efficient image sensor and reflective retro combined solution designed to realize touch screen functionality with minimum cost and great scalability.

The simplified optical imaging based touch screen system is illustrated below in figure 1:

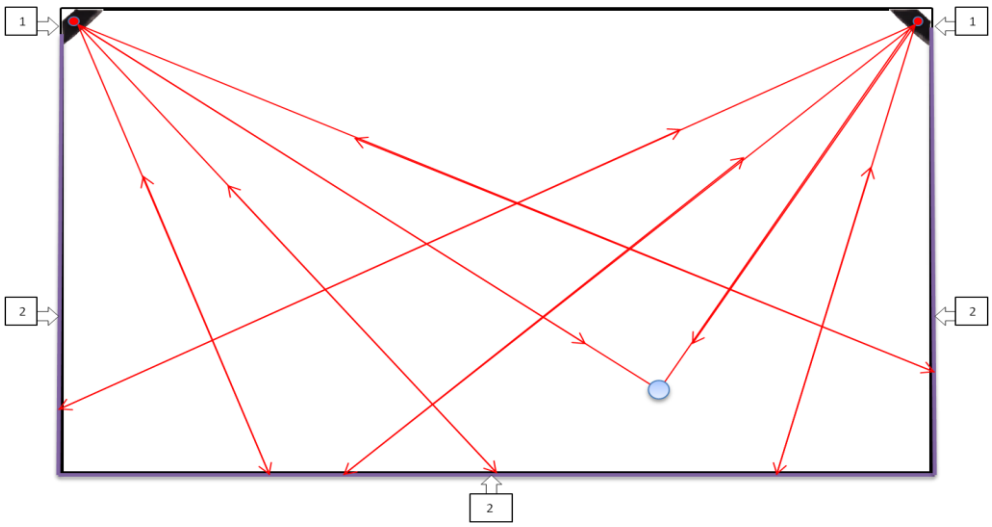


Figure 1 Simplified Optical Imaging Based Touch Screen System

Inside the basic optical touch system, there are two image sensors (each being single line scan based, one at the left top corner and the other at the right top corner) with a LED positioned on top of each sensor chip (1). Specific reflective material has been placed along the screen edge (2). The simplified system mechanism is when there is no object within the active touch screen area all the infrared light emitted by the two LEDs will be reflected back to the image sensors by the retro material at the screen boundary; the image sensors receive the full level light reflection as below in figure 2:

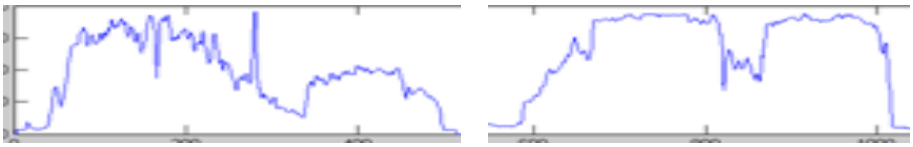


Figure 2 Left and Right Line Scan Sensors Waveform

When an object occurs within the touch screen active area, part of the reflection will be blocked as shown below in figure 3:

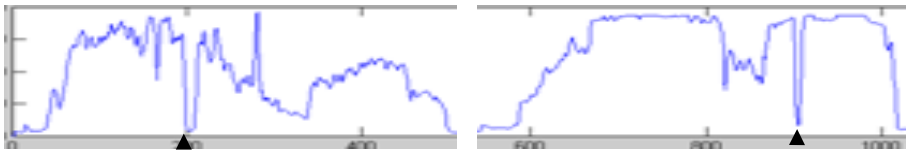


Figure 3 Left and Right Line Scan Sensors Waveform with Single Touch

By converting the blocked waveform locations to actual angles from the sensor view points, and coordinating with the physical screen dimensions, the actual touch point is able to be triangulated and its position calculated. Moreover, a significant number of advanced methods and techniques have been evolved based on the basic mechanism as described above to achieve real time high quality performance. The basic optical imaging control and process unit behind the system is comprised of a microcontroller which directs the timing sequence of the LED based illumination subsystem and image sensor subsystem to acquire and transmit the raw touch information. A PC based processing system abstracts the edges, localizes and finalizes the touch positions, resolves the occlusions etc. The existing system is relatively accurate, robust and meets the current real time requirements reasonably well. Nonetheless, a certain number of system limitations and additional requirements have been identified.

Requirements Analysis

With the ever growing market competition and increasing customer expectations, there are three new design requirements investigated in this research: Low level system acquisition requirement, high level processing requirement and system cost requirement on the resource level.

System acquisition requirement

The general-purpose microcontroller based data acquisition unit is a system bottleneck which relies on a sequential acquisition mechanism with a relatively limited frame rate. Customized acquisition control logic is urgently required, to execute a concurrent acquisition control structure where one image sensor is capturing image pixels while the other sensor is simultaneously digitizing and transmitting the pixels. The significant increase in the touch information frame rate is required for fast touch activity detection and touch history tracking, which will play an important role in the subsequent processing module.

Another major problem in the current optical imaging based system is the touch occlusion issue which is largely caused by lack of sufficient screen imaging information. Furthermore, a fundamental requirement of the next generation operating system is the supporting of multiple touches that demands more accurate coverage of screen information from more different camera view perspectives. Therefore, it is important that the proposed acquisition control logic is not only able to provide a more efficient concurrent mechanism for improving the frame rate but should also have the capability and extensibility for handling a larger camera based bandwidth in a more advanced system.

System processing requirement

In current touch screen systems, the majority of system processing functionality has been placed on the PC processor with a combined latency of sequential acquisition delay and PC algorithm operation time. With the potential increase in the number of acquisition sensors and frame rate, longer latency will occur in the current acquisition engine and more processing load will placed on the general-purpose CPU which is also shared by other applications within the OS. In order to reduce the low level system latency under the increasing data rate and computational pressure, it is necessary that certain processing in the critical path is pipelined and pre-processed with the acquisition process. Pre-processing operations such as edge detection, ambient subtraction and pre-touch indication leads to the process-on-the-fly requirement.

System resource requirement

Market and cost considerations are always involved in real product development. A minimum amount of hardware resources is demanded in this industrial research for realizing the aforementioned concurrent acquisition engine and pre-processing module. Moreover, a further interest has been expressed for this project to explore and analyze the feasibility and resource consumption of a complete on-chip hardware solution achieving basic touch screen functionality to lower the overall manufacturing cost.

1.2 Objective and Methodology

This is investigative research based on real world touch screen system requirements, which aims at creating customized logic to acquire touch information and further explore the feasibility of a complete on-chip hardware solution to realize basic touch screen functionality. There are three objectives in system research and development according to these requirements:

- Create a customized and more advanced data acquisition system compared with the existing platform in terms of its operation mechanism, speed, accuracy and noise level.
- Research and validate the concept of embedding a complete touch system on a chip with an incremental design and verification process.
- Optimize the resource utilization without compromising overall system performance.

For the system design methodology, it is necessary and critical to understand the existing system structure and functionality. Therefore the same functionality is required to be migrated to a new platform (a field programmable gate array) on the system level with the consideration of technology differences and the explicit partitioning of different system modules. After that, more sub-modules are designed, constructed, verified and tested to realize the required system level functionality with constrained timing requirements. Then all top level modules are integrated and synchronized to achieve system timing closure with a well organized control flow. In the end, a number of optimization techniques will be applied to both system level and module level components to minimize the resource consumption while maintaining the same functionality.

During the development process, there are three design phases:
Phase One: Data Acquisition System
Phase one is a performance oriented design process with the purpose of constructing a high speed, concurrent data acquisition engine to replace the existing general purpose microcontroller based acquisition system. It is planned to start by constructing the image sensor timing control logic with the basic capability of configuring and controlling sensors to acquire analog pixels, equipped with a LED based illumination subsystem with more complicated timing control between the shutter and operation modes. After that, an analog-to-digital convertor is added to

digitize the analog signals which will be further transmitted out of the data acquisition system through a USB module. Each module will operate at the highest possible execution speed by taking advantage of the pipeline structure. It is acceptable that extra resources are consumed to reduce overall noise level and a new concurrent timing control mechanism is expected to maximize system operation efficiency. The data acquisition system is aimed initially to achieve the highest possible system performance without consideration of resource consumption. At the end of phase one, a number of tests are required to verify the real time performance is acceptable.

Phase Two: Processing Unit

The development of the processing unit is considered as an investigation and verification process using the incremental design methodology [1] where each new module is constructed and then integrated with the previously verified system. The design objective of the processing unit is to realize basic touch screen functionality with minimum concern for resource use. Based on raw touch information collected by the Data Acquisition System, the Processing Unit design process starts from a proposed memory management unit to store and direct image frames before further processing. Dedicated logic blocks are required to perform initial edge detection, then touch position localization and initial touch level detection, then touch motion tracking. Each block development will go through its own verification and test process [2]. Finally, it is necessary to have a system master controller synchronizing and organizing all processing modules, to respond to both the acquisition system and PC through a customized transmission engine. A software interface is required to demonstrate the correct functionality of the basic touch screen system during real time user testing.

Phase Three: System Optimization

After successfully establishing the proposed touch screen system, a number of optimization techniques are applied to the acquisition system and the processing unit on different levels (from the algorithm level to the logic gate level) to minimize resource consumption without compromising overall system performance. An optimized on-chip hardware solution with proven real time performance will be validated as the result of this research.

1.3 System Abstraction and Overview

This research is based on real world touch screen application requirements with specific investigation, implementation, verification, validation and optimization involved to create a dedicated on-chip solution. Field Programmable Gate Array based application specific control logic is proposed and developed in this research not only to investigate the possibility of a customized acquisition engine with desirable performance but also to explore the feasibility and capability of an on-chip touch information processing unit with the minimum consumption of system resources. The abstraction layer of the system is illustrated below before being expanded and explained in detail in later chapters.

The complete system is comprised of three major modules: Data Acquisition Unit (DAU) which controls and coordinates the image sensor, illumination and ADC subsystems to consistently acquire, digitize and transmit the real time touch information to processing logic; Processing Unit (PU) which performs real time touch detection, object positioning and motion analyzing with low latency characteristics by taking advantage of a pipeline structure; and a PC monitor which continuously updates the touch screen optical imaging scope and user touch positions.

The following is the simplified system block diagram on an architectural level:

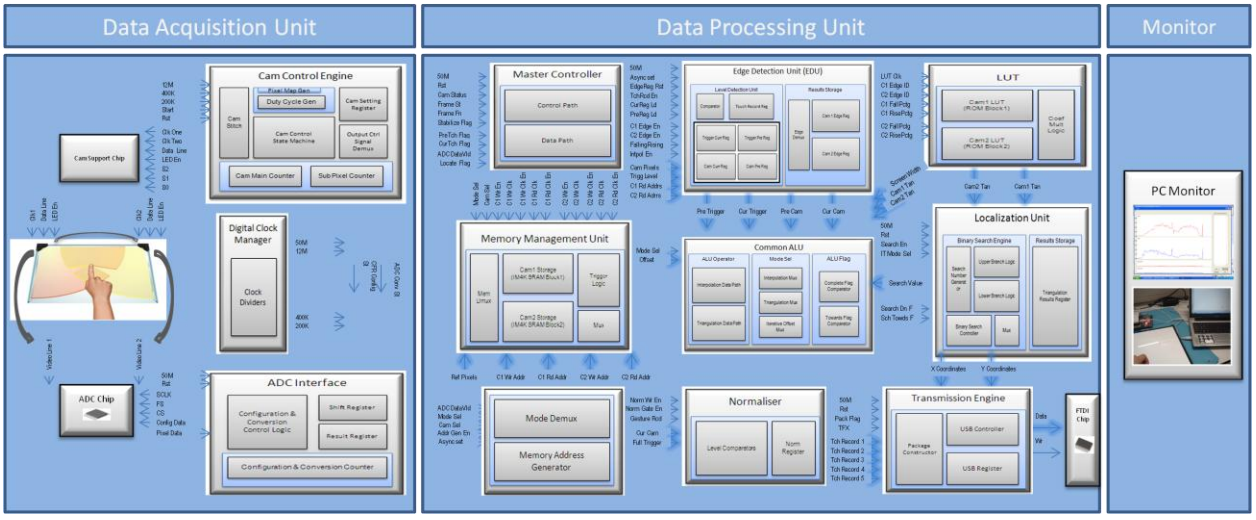


Figure 4 Simplified System Block Diagram

The following paragraphs will concisely introduce the design concept and system flow with an overview of how the system operates. Brief explanations of the main components integrated inside the three main blocks are presented here.

Inside this vision (image sensor) based real time touch screen system, the **Data Acquisition Unit** (DAU) plays a fundamental role in raw optical imaging information acquisition by controlling and organizing the **Line Scan Sensors**, **LED** based Illumination system, **Sensor Support Chip**, **ADC Chip, ADC Interface Logic** and **Digital Clock Manager** (DCM). The unit is designed and partitioned as a self-contained module with the **Data Acquisition Control Engine** operating all control signals within the unit so there is no interference with other units.

The **Processing Unit** is the key block of the whole touch screen system. The digitized image frames from the Data Acquisition Unit are redirected by **Mode Demux** to be either stored into **RAM** in the **Memory Management Unit** (MMU) in iunitialization mode or fed into the **Edge Detection Unit** (EDU) and **Normaliser** when in operation mode. The **Trigger Logic** inside the MMU adjusts the threshold level for different ambient light environments while the **Memory Address Generator** updates both the reading and writing addresses for the MMU. The Edge Detection Unit (EDU) is constituted of a **Level Detection Unit** which defines and indicates the touch action and the **EDU Result Register** which stores both the rising and falling edges of the touch for both image frames.

More accurate touch edge positions can be obtained using the **Interpolation Data Path** inside the **Common ALU** module catering for more precise requirements. All touch edges are translated from image Pixel Ids into real angles in the detection area from both the top cameras' perspectives, using a pre-calculated **Look up Table** (LUT) in ROM. After that, the two angles are triangulated in the **Position Localization Unit** to calculate the x and y coordinates of the touch on physical screen. The **Triangulation Data Path** is involved in localization processing while the hardware **Binary Search Engine** is used to optimize the search process by accelerating the system critical path to a significant extent.

The Normalizing Process inspects and analyses subtle application actions such as pre-touch using the **Level Comparator and Register** module inside the **Normaliser**. This runs in parallel with the Edge Detection and Localization. As a result, the most recent touch percentage levels will be stored in registers for further touch history tracking analysis. Finally, both the concurrent processed results (touch position and tracking history results) are reconstructed in packages by the **Package Constructor** in the **Transmission Engine** and transmitted outside the block.

The **Master Controller** inside the Processing Unit is the core element of the system operation which not only directs process flow and organizes control signals to and from sub-modules simultaneously, but also coordinates and interfaces with the Data Acquisition Unit and PC monitor to achieve real time system synchronization and coherence.

The **PC Monitor** receives data transmitted from the hardware touch screen system through a **USB2.0** connection. The software interface is designed for high speed USB transmission with two GUIs: one for displaying the raw touch screen optical imaging information for development debugging and data acquisition process validation; and the other for updating actual real time touch position results based on the hardware processing unit in a visual form for system demonstration purposes.

All units and modules abstractly aforementioned will be further discussed and illustrated in detail in the next few chapters.

Chapter 2 -- Literature Review

2.1 Hardware Accelerated Hybrid System

In modern electronic system design, with increasing demands in processing capability and more stringent requirements for system latency, it is difficult for a real time application to fulfill design specifications if it is only implemented in software. The hardware accelerated hybrid system [3] emerged due to this reason in the early 1990s to describe a process where certain parts of the program's critical functions are constructed in hardware to meet performance target and timing constraints while other parts are implemented in software in the processor . It is informed by multiple design disciplines and approaches based on the characteristics of an application which can be classified as a software-oriented hybrid system, hardware-oriented hybrid system or software/hardware co-design system.

2.1.1 Software-Oriented Hybrid System

For software-oriented heterogeneous system design, without violating any application performance constraints, a significant proportion of system functionality has been implemented in software with a minimum amount of hardware processing. Most often, software-oriented design takes advantage of the existing processor or microprocessor structure (either general purpose or special purpose) with dedicated software programs that allow the user to define the desired functionality using a specialized language. On the instruction level, as indicated by [4], programming is achieved by executing on the hardware supported by the existing architecture. It is considered to be more flexible with a relatively high level of software and lower cost by implementing major system design into the available structure with very limited extra hardware cost. Compared with a pure software solution, the software-oriented hybrid system is more efficient with a hardware acceleration of a small part of the system.

The software-oriented hybrid system structure is illustrated below redrawn from [3]:

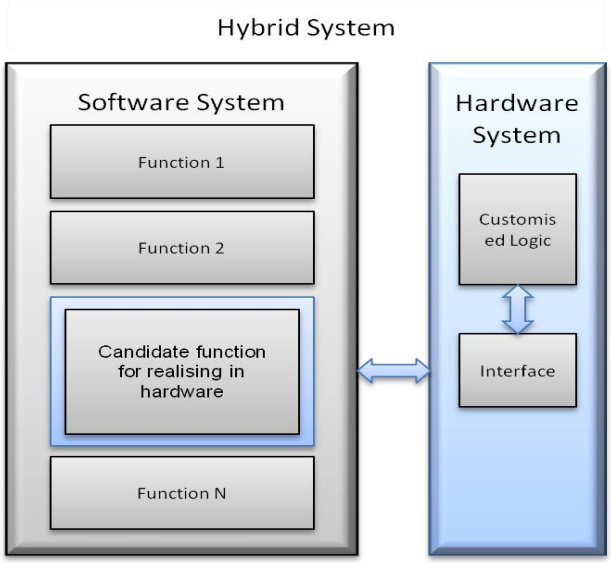


Figure 5 Software-Oriented Hybrid System

Low Level Latency Requirements

In most software-oriented hybrid system design, the key processing timing and latency requirements are less demanding which makes it easy and cost-efficient to implement the major proportion of system functionality into general processor based software. For example, in Garufi and Acernese's hybrid modular control and acquisition system [5], with low sampling and performance requirements, the complete distributed control algorithm was processed by PC based software with low cost hardware logic used in the critical acquisition path. Another example is the control design for swing scanning infrared earth sensor application [6]. The entire earth wave algorithm has been executed in an embedded ARM processor with necessary acceleration in obtaining raw data by taking advantage of the hardware structure.

Flexibility and Extendibility

For some other software-oriented systems, the specification requires flexibility and adaptability because a certain amount of system functionality is to be executed unpredictably and sporadically. Thus, it is natural to implement the majority of the proposed design into a standard software core to allow rapid change to a new algorithm. The real-time dark environment vehicle detection system [7] is a good example. With basic hardware acceleration on pixel-level

operation, the main vehicle searching and decision making algorithm has been executed on processor based software because of the broad variety of possibilities and uncertainties.

Computational Cost and Execution Convenience

Sometimes the system functionality is technically difficult and computationally expensive to be implemented into pipelined hardware or the required function is already an existing component of a processor based system. For instance, floating-point arithmetic operations are a basic element inside numerous algorithms which are widely supported by existing instruction sets inside conversional PC or processor based systems. And for modern software-oriented platforms (physical or soft [8] on chip processor core based system) it is also commonly supported. Thus, from the cost perspective, some applications are best suited to processor based design. Meanwhile, software-oriented design is considered efficient and convenient in terms of implementation cycle and requirements for specialized human resources compared with a fully customized hardware design.

2.1.2 Hardware-Oriented Hybrid System

The aforementioned software-oriented hybrid system design usually has a high speed clock frequency and relatively powerful sequential computation processing ability. However the hardware-oriented design is practically application-specific with the characteristic of inherent parallelism to meet stringent user requirements. Usually, the operational frequency in hardware dominated design is much slower than PC or processor based system but with a capability to provide better performance through paralleling more operations, minimizing memory access and optimizing critical paths.

The hardware-oriented hybrid system structure is illustrated below: the majority of system functionality has been implemented in hardware logic with tight and multi-level interconnections between system peripherals and the subsequent processing block.

Hardware-oriented hybrid system structure:

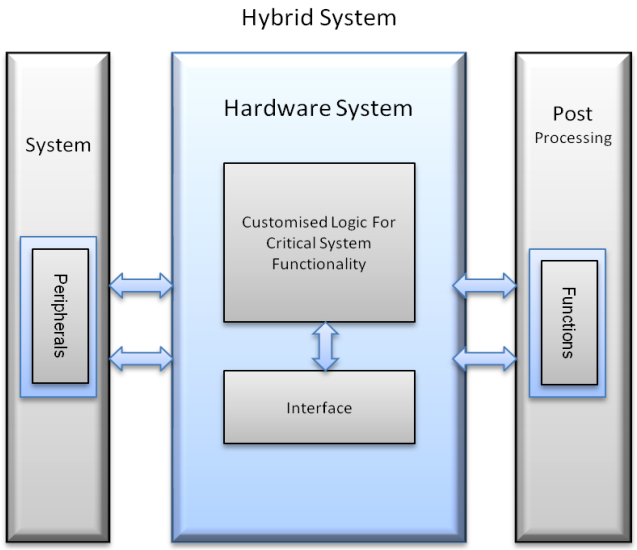


Figure 6 Hardware-Oriented Hybrid System

High Throughput and Low Latency

In most high performance real-time applications, throughput and latency are two significant factors affecting the performance and outcome of the system. Morris, Thomas and Luk's [9] financial trading system is an example. "The combination of increased message rate and more complex market feed data format " naturally requires a system which has high bandwidth message processing capability and desired latency in a situation where large bursts of trading activity occurs. Software-oriented solutions are often unable to keep up with the input data rate and also have difficulty meeting the latency requirements of sub-millisecond response times. Under such circumstances, there is a shift during the application design towards a more hardware-oriented system with an accelerated propagation structure.

Functionality Effectiveness, Compactness and Robustness

With an explicit design specification or clear problem definition, the architecture of a hardware-oriented system is often constructed in a compact and application-specific manner such as [10] compared with a general processor software solution. In detail, the hardware dominated solution usually starts with a straightforward design approach which resolves the problem in a direct way

that leads to better efficiency. The hardware oriented structure is usually more tightly interconnected and integrated with target requirements, which leads to a better solution because the hardware solution is more convenient for adding concurrent features inside each block and it is more versatile because it is more easily extensible.

Architecture Parallelism and Reduced Response Time

In safety-critical areas, such as medical instruments, the efficiency and efficacy of equipment design plays a crucial role in accurate and appropriate problem identification and localization, which sometimes reduces the instrumentation operation and response time, helping minimize unintended damage (such as to healthy tissue etc). Minimally invasive image-guided intervention (IGIs) is a typical application that benefits from the hardware-oriented design method with a significant improvement in execution time from hours to a few minutes compared with the conventional open and invasive procedure. Inside the IGIs workflow, the deformable image registration is the fundamental and major step, which in Dandekar and Shekhar's system [11] has been implemented into a highly pipelined multimodal based hardware architecture to reduce the procedure time.

In order to maintain the high level calculation accuracy, which is equally important in hardware occupied medical instrument design to minimize the possibility of complications, there are commonly two execution alternatives. One is constructing a dedicated floating processor unit at a high cost of hardware resources, such as plane-sphere intersection in a hardware ray tracing system [12] having an independent pipeline arithmetic unit for meeting accuracy requirements. The other alternative often used is the adoption of multiple look up table (LUT) based polynomial approximation. In Castro-Pareja, Jagadeesh and Shekhar's system [13], two LUTs (Interpolation Weights and corresponding floating coordinates) have been utilized to replace a standard arithmetic unit while achieving the same operational precision.

Security and Adaptability

For certain areas, such as military communications, there is a growing concern not only in traffic speed and bandwidth, but also in network security where a device is required to protect the intellectual property of a design from attempts to reverse engineer or replicate operation. The general-purpose processor based software solution is considered relatively unsecure in terms of concealing internal algorithms since "their reliance on external instruction streams means that they are susceptible to instruction bus monitoring attacks while operational" [14]. In order to maintain network security, hardware based solutions are used where the critical function has been separated and implemented into hard-wired logic in a self-contained manner without requiring external instruction data. At the same time, by taking advantage of the capability of run-time reconfiguration of certain hardware devices, new protocols and algorithms are able to be processed and adopted without compromising performance. The information exchange security system based on specialized hardware has demonstrated a more capable security monitoring mechanism and in addition the "reconfigurability and expandability of the solution provides an apparatus for further improvement and elimination of potential threats that are still to arise" [15].

2.1.3 Hardware/Software Co-Design System

The hybrid hardware/software co-design [16] emerged to embrace new design requirements in the early 1990s to describe a process where part of the program's functions are constructed in hardware while the others are implemented in software in a processor. The increasing use of the co-design method for complex systems stems from a variety of factors. One is "the need for multi-formalism specification" [17] where most modern embedded systems consist of different components (analog, digital etc.) and there is no common description or language to cope with specification variations within the system. Another is the market pressure for real-time products which demands shorter development cycles and effective design cost. Last but not least, design reuse is a key issue in complex system design convenience and coherence.

The hardware/software co-design is the net result of a significant number of research activities in the embedded area. It is informed by multiple disciplines including software/hardware partition design, application-oriented hardware circuits design and hardware/software domain interface design. Compared with the aforementioned hardware-oriented design and software-oriented design, there is neither an explicit overall performance constraint demanding a complete hardware solution nor a pure software solution fully satisfying the specification without acceleration.

Hardware/Software Co-Design Approach:

There are a number of fundamental stages for establishing a complex system following the co-design approach: from system specification, system profiling, architecture partitioning, concurrent hardware and software development to the actual run-time system. The general approach route is illustrated below redrawn from[18] with minor changes:

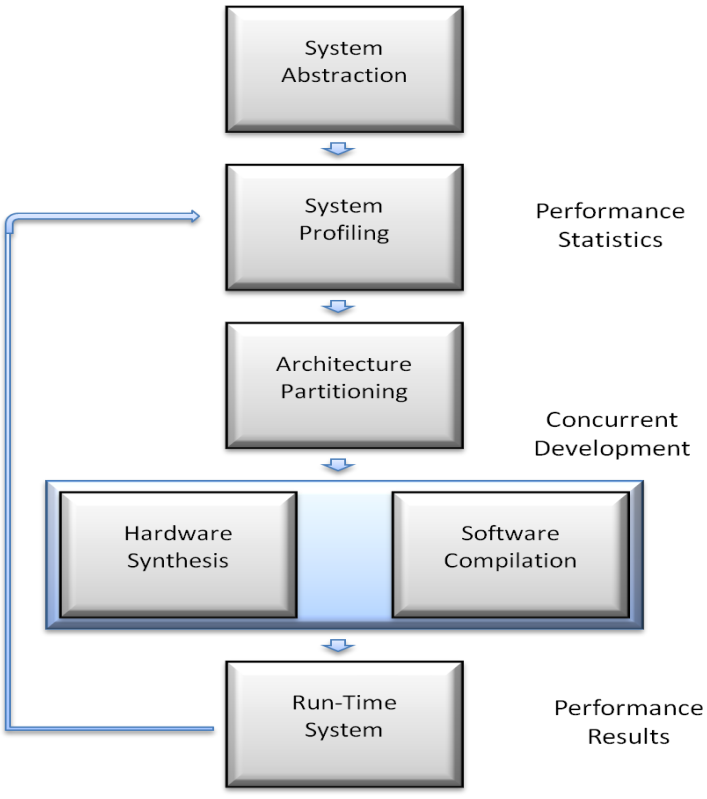


Figure 7 Hardware Software Co-design Approach

System Abstraction

The abstraction of system functionality based on heterogeneous specifications can also be described as “modeling system functionality and constraints " [18]. It is usually considered as a high level behavioral description regardless of implementation details.

System Profiling

System profiling refers to the identification of performance in critical regions and the timing estimation for real time operation using software (for instance C code) without real execution on the candidate architecture.

Architecture Partitioning

Hardware/Software partitioning aims to determine which parts of the system should be implemented in hardware and which in software by applying various trade-offs and spectrums [19]. As further explained by Ismail and Jerraya [20] the architecture is partitioned by assigning the whole system functionality to concrete parts of the physical system, which will be either compiled into machine code in a processor or synthesized into a hardware circuit description. The final system architecture formalizes the result of “several successive trial and error iterations of this step" [17] based on the detailed specification, timing constraints and designer’s experience.

Concurrent Development

During the implementation stage, the hardware compilation and software compilation are executed simultaneously after the system evaluation and partitioning from previous stages. The concurrent development follows different design processes where hardware compilation is composed of hardware behavior description, optimization, synthesis, placing and routing, linking etc using a modern reconfigurable platform while software compilation generates machine code for the processor structure.

Run-Time System

The run-time system is the final operation platform which varies according to applications, such as a general-purpose CPU with Field Programmable Gate Array, or an application-specific processor with an application-specific-integrated-circuit.

To conclude, In order to achieve hardware/software co-design success, it is not only critical that the initial partitioning be made appropriately into hardware and software, but also that the ensuing system level development proceeds along lines that benefit from an integration of both hardware and software perspectives. Compared with traditional design, it is more efficient to plan the design with a more detailed specification available before selecting the final architecture. It is less costly to modify parts of the system between different partitions at later stages of development and it is easier to facilitate the final integration between different technology domains. Alternative system implementations are considered with respect to performance, physical aspects, reliability, modifiability, maintainability and manufacturing cost as emphasized by Purvis and Franke [21].

2.2 Image Acquisition and Processing

Vision based systems have been a popular research topic for the last decade where a significant number of areas have benefited directly or indirectly from it, such as robotics and autonomous systems [22], surveillance [23] and navigation systems [24]. It is also considered to be the general research and knowledge background for the proposed touch screen system. Vision systems aim at enhancing and improving the visual image characteristics for a human viewer through a variety of techniques and processes depending on applications. The acquisition capability which refers to large volume data handling, memory bandwidth and real-time constraints [25] and the processing ability which depends on the efficiency and parallelism of algorithms, arithmetic and logic operations are two major factors that determine the success of a vision based system.

There are three critical stages during the vision system design which includes data acquisition, system pre-processing and high-level processing.

Image Acquisition

Inside a vision based system, data acquisition is the fundamental image-sensor based integration control and read-out process to “set the mode of operation, address the pixels and transfer the data to high-speed local storage" [26]. And the difficulty of designing such an acquisition system usually lies on the efficiency of the interface which is the bridge connecting the system’s front end and back end processing [27]. Usually, the image acquisition storage structure is developed by consideration of the related processing unit which has an impact on data collection.

System Pre-Processing

In recent real-time application development, there is an increasing trend towards the system pre-processing unit design because of the pressure of larger volume data from multi-inputs and the stringent performance and latency requirements. In the image processing area, the pre-processing is usually referred to three major components: image enhancement, feature extraction and interest segmentation. **Image enhancement** is a basic restoration approach to make subsequent analysis easier over environmental variation [25] through a number of techniques such as noise reduction, contrast or color correction [28] and lens distortion. The second important component in image pre-processing is **feature extraction** where typical properties like lines, edges [29], texture and shapes [30] can be extracted to reduce raw data size in order to reduce the processing burden on the main system. Last but not least, the **interest segmentation** step refers to “ the selection of a specific set of points or regions of interest which are the subject for further processing" [31]. On the pixel level, the image segmentation task functions on the spatial domain “grouping together neighboured pixels or voxels to homogeneous regions if they can be considered to be similar according to a common feature" [32]. In general, the purpose of pre-processing is to robustly abstract typical characteristics and information useful for subsequent processing from the image to further optimize overall system processing distribution.

High-Level Processing

High-level processing is the critical stage of collecting, analyzing and generating the final system outputs. The results after advanced processing are likely to be route and movement decisions from a vision based navigation system [24], or the identification of a person from a face authentication system [33]. The high-level processing is usually performed by a general or

specific processor structure because of the variety, flexibility and complexity of the dedicated algorithms.

On the implementation level, a number of studies have investigated the optimum system structure partitioning between hardware and software in different applications.

In **two dimensional image based vision detection**, Alt, Claus and Stechele [7] have built a vehicle detection system where the image acquisition, spotlight detection and region labelling are partitioned in a hardware structure while light pair searching and plate searching are executed in an embedded processor. Pei, Chun and Li [34] have a similar design partition in their departure warning system with frame acquisition and liquid-crystal display and other peripherals controlled by hardware. The implementation of a fundamental Gaussian smoothing filter based global edge detector has been proposed in hardware from their research. In order to meet increasing system performance requirements in real time, more and more researches have shifted intensive image processing units from software-executed processors and customised this into more application-specific logic structures. For example, Bonato, Marques and Constantinides [35] describe a feature detection system, in which major system processing such as pixel streaming, magnitude calculation, orientation computing, and key points detection and classifying are constructed in logic structures and the embedded processor is used to generate descriptors for future feature extension. Vicente and Munoz [36] describe an object counting and tracking system which is constituted of critical object and contour detection algorithms constructed in hardware structures and more flexible classifying and tracking modules implemented in a processor.

Furthermore, in other performance-oriented research applications, more hardware resources are consumed to achieve design goals. For example, Ishii, Taniguchi and Yamamoto [37] have developed a high speed (1000fps) real time vision platform where most image processing algorithms (multi-target colour tracking, feature point tracking, optical flow detection, and pattern recognition) are accelerated by a hardware pipeline structure. Another example is Yean and Yu's [38] Smart Camera system for Gesture Recognition in HCI Applications The complete Harris keypoint detection algorithm and Kanade-Lucas-Tomasi (KLT) feature tracker algorithm have been implemented in a logic structure.

The proposed touch screen system is a **line scan image sensor based specific object detecting** application. Ohtani, Baba and Notohara [39] have built an optical measurement system with the analogue scan unit and ADC conversion controlled by software in a processor while hardware is responsible for actual spot position calculation from digitized information. In Hussmann and Ho's real-time edge detecting system [40], hardware was used for most elements inside the system including sensor scan circuits, ADC conversion, and low-pass filter. In addition, a simple edge localization algorithm was constructed in hardware. The processor was used for further advanced post-processing. From their work, it is suggested that the fundamental architectural, data-path critical and computationally less complex functions are more efficiently implemented in hardware while software is preferred to handle more flexible features. These design considerations will be applied in constructing the touch screen system.

2.3 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are reprogrammable hardware devices that can be utilized to implement logical functions through a hierarchy of reconfigurable interconnects among digital blocks. It is the advanced technology having both the execution speed by taking advantage of the parallel hardware architecture and the implementation flexibility through the programming approach. In modern designs, there is a growing trend in applying FPGAs as development and product platforms. There are three major types of FPGAs among different vendors: SRAM based FPGA, Flash based FPGA and Anti-Fuse based FPGA[41].

SRAM FPGA

Static memory technology based Field Programmable Gate Arrays are widely used by major vendors such as Altera and Xilinx where power is required to be sustained to retain the internal data during the operation. Usually an external boot device is needed to provide in-system programming through bit streams.

The following is the basic logic element structure from an SRAM based Altera Cyclone II device [42]:

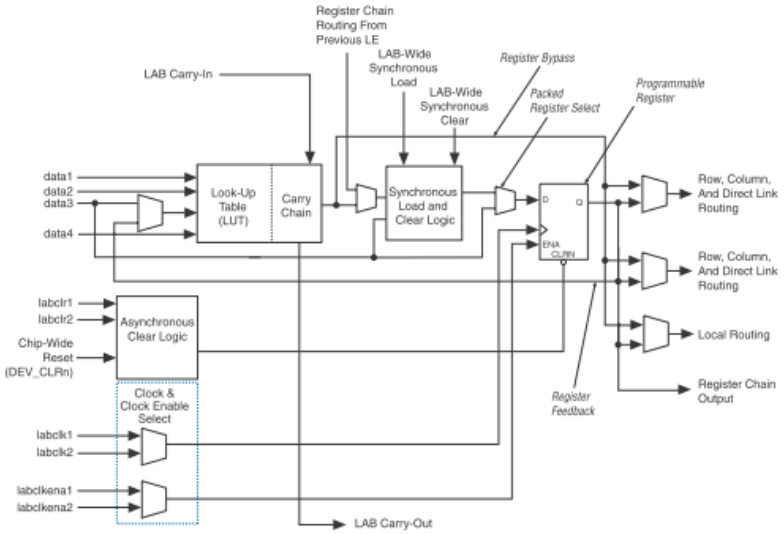


Figure 8 SRAM Based Logic Element Structure

The basic logic element is composed of a four-input look-up table (LUT), which is a function generator that can execute any function of four variables: a programmable register which can be configured as different logical operations such as D flip flop or T flip flop; a carry chain for function directing; a register chain for unit cascading; and some other components for controlling and linking signals.

Flash based FPGA:

Flash-erase EPROM based Field Programmable Gate Arrays are reprogrammable and low-cost hardware devices. The FLASH based FPGA is able to retain the value after power-off and will always leave residual charges after being erased.

The following is the logic cell of a Flash based Actel ProASIC FPGA device[43]:

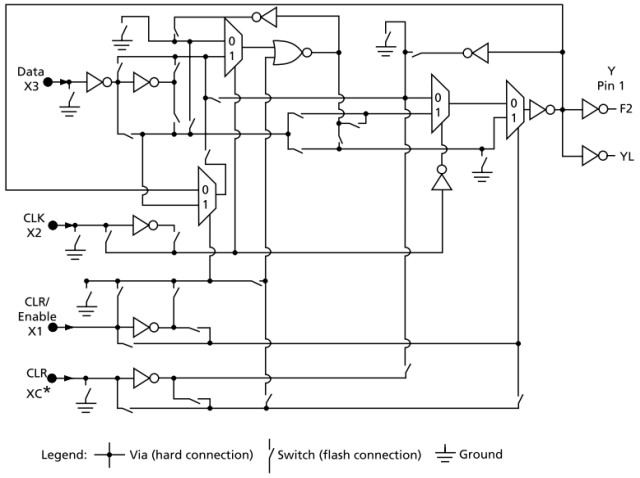


Figure 9 Flash Based Logic Cell Structure

The basic cell structure also has a look-up-table (which is a common component in most FPGA devices) with three inputs that can implement a three-input combinatorial gate or flip-flop with enable. This flash based Actel device is similar to basic ASIC cells in the sense of its fine granularity.

Anti-Fuse based FPGA

Anti-Fuse based Field Programmable Gate Arrays are less used due to their limitation of one-time programmability. However, its less flexible physical structure has relatively high security that prevents illegal etching and design retrieving.

FPGA Verses ASIC

After establishing a proposed hybrid system with proper considerations on system level partitioning and module level integration, it is usually required to convert all or part of the system into an application specific design (ASIC) to lower product cost and increase system reliability, especially in high volume applications. The ASIC design process and semiconductor process performance characteristics vary according to different manufacturers. The efficiency of resource consumption is normally measured as the number of standard cells. Compared with the traditional ASIC design process which usually has a long design cycle and relatively high NRC (Nor-Recurring Cost), recently there is a growing trend to create hardware embedded simulations

to facilitate incremental design verification of ASICs by using FPGA (Field Programmable Gate Array) devices. With its high flexibility and short time-to-market, as indicated by Selvaraj, Sapiecha and Dhavlikar [44], this FPGA emulation has emerged as a major ASIC verification technology.

FPGA Verses DSP and Mobile PC

Compared with a Digital Signal Processor (specialized microprocessor for signal processing) and a Mobile PC (general purpose processing unit), parallelism is the most significant advantage in an FPGA which enables Process-On-The-Fly (processing during data acquisition), multiple operations within one cycle and multiple algorithms processed in a pipeline. The drawback is the relatively slow operational clock, relatively more expensive cost and much higher design and verification effort compared with an equivalent implementation running on a DSP or PC [31]. Powerful arithmetic operation processing capability is one well known feature of Digital Signal Processors, such as multiply accumulate (MACs) which is ideal for mathematical algorithms. Recent DSP's supporting of SIMD [45] (single instruction multiple data) featuring the instruction-level-parallelism through very long instruction words (VLIW) improves execution time to a great extent. The general processor is renowned for its super fast clock speed and multi-threading for potential multi-cores. The following figure shows the results of a performance benchmark of selected vision based algorithms implemented on different high-speed embedded platforms: Field-Programmable-Gate Array (FPGA) (Altera Stratix- II families with 133MHz core), digital signal processor (DSP) (Texas Instruments TMS320C6414 with 1GHz system clock) as well as mobile PC processor (Intel Mobile Core 2 Duo T7200 with 2GHz clock) [31]:

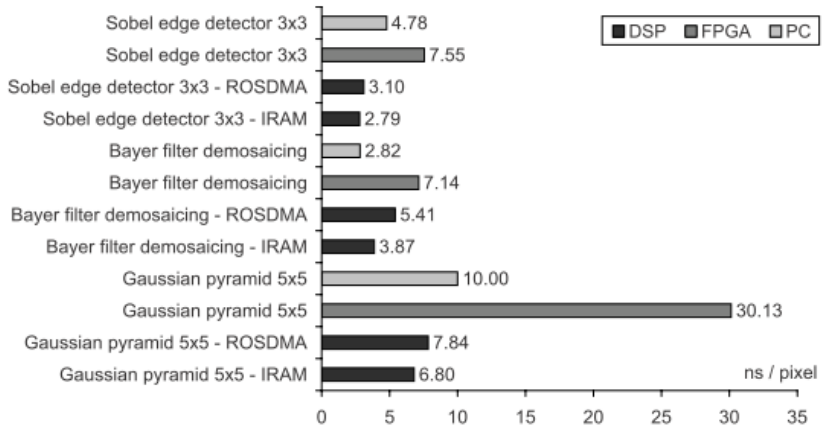
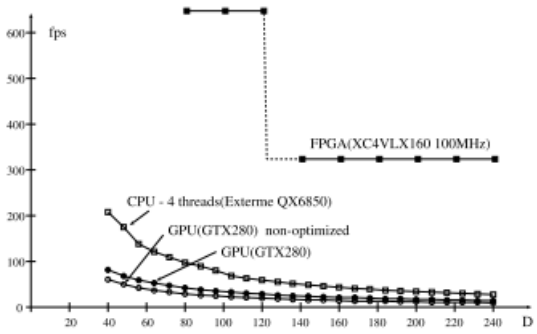


Figure 10 Performance Benchmark of Vision Based Algorithms on Different Platforms

FPGA Verses GPU and CPU

FPGAs have shown very high performance in many applications in image processing. However, recent GPUs and CPUs also have the potential for high performance for similar applications. A Graphics Processing Unit (GPU) is designed particularly for intense graphic processing which usually has a much faster clock than a FPGA and a little slower clock than a CPU, but supports a significant number of cores (for instance, 240 cores in Nvidia GTX280) running in parallel and outperforms a CPU. The disadvantage of the GPU is the relatively slow memory access speed which affects the efficiency of communication and the limited data transfer between grouped cores fixed by its structure [46]. The following figure shows the relative performance for stereo-vision and k-means clustering algorithm. As computational complexity increases the performance of all GPU, FPGA and CPU based solutions decreases. However, the FPGA based design consistently outperforms both GPU and CPU solutions with the stepwise curve [46].

Performance of the stereo-vision



Performance of the k-means clustering algorithm

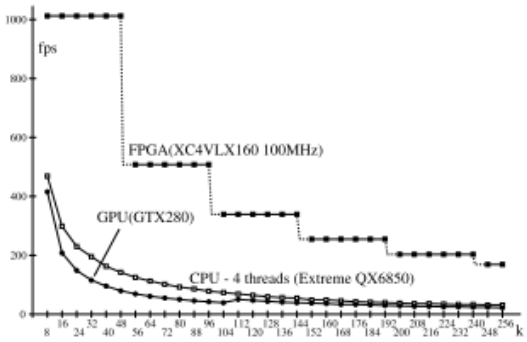


Figure 11 Stereo-vision and K-means Clustering Algorithms on Different Platforms

2.4 System on Chip (SoC)

With the rapid advance of electronic design automation and IC fabrication technology, System-On-Chip (SOC) has become more and more popular in complex system design which integrates data acquisition, signal conversion, application specific processing and various peripherals into a single chip. It refers to integrating all system components into a single circuit or chip to reduce manufacture cost and enable more compact systems.

Compared with traditional board level design, the System-On-Chip approach has a number of advantages in terms of higher system integrity, higher operation speed, lower development cost, more compact chip size and diminishing time to market. A typical SoC device can be a hybrid module with embedded processors, on-chip memory, versatile general purpose IO ports, high speed serial links, standard peripherals and other digital or analogue components[47]. In practice, external memory chips and various specific peripherals are often needed for different application requirements. The purpose of System-on-Chip development is to realize the required system functionality through a single chip resource with minimum external peripherals.

There are two main SoC development methods: **Block Based Design** (BBD) method and **Platform Based Design** (PBD) approach. For block based SOC design, the system is constructed based on assembling and coordinating a number of hardware blocks under sub-system functionalities to achieve an overall performance, while for platform based design, the system is a device integrating a predetermined collection of resources with a more system and model level focus [48] . Extensive research has explored methodologies and techniques on both design methods to improve the SOC design overall efficiency. Carloni and Li [49] proposed a latency-insensitive design (LID) to increase global performance in Block-Based Design. In the platform based approach, the reuse of intellectual property [50] and pre-designed blocks or virtual components (VC) [51] have been applied to boost design productivity.

A number of performance criteria have been required from modern industry system applications according to [52]. These are the demand for high-level integration and density of the target for the control module; the capability of supporting high-performance control algorithms with flexibility and modifiability; and "reliability, accuracy, and safety in a harsh environment". In order to meet most of the above criteria, recent research studies show that the digital hardware solution, such as the field-programmable gate array (FPGA), is an appropriate SoC development and implementation platform compared with other solutions.

The proposed touch screen system is an image sensor based design, using a Field Programmable Gate Array as a fast and advanced prototyping and implementation platform. In the image sensor based field-programmable-gate-array-executed detection and tracing system on chip area, within

the platform based design domain, Kim, Park and Lee's outdoor scene tracking system[53] has been designed relying on a platform based approach with a large amount of component reuse, such as the image encoder. Eun Tack and Kwang Sung have demonstrated an object and distance tracking SoC system based on field programmable gate array prototyping [54]. The on-chip system is comprised of processor core (ARM core), I2C control, SRAM control for stereo image capturing, DMA control, LCD driver and other basic peripherals. The distance and object trackng algorithm has been partitioned in processor based software. The development is under the guidance of the platform based approach where a variety of IPs have been adopted from previous designs such as LCD control IP and SRAM accessing IP. Moreover, the system has been further organized as a new IP for future platform development purposes.

Since the proposed research aims at providing a system solution for a specific application in a unique technology domain, it is estimated that most system functions and components are required to be constructed and customized according to specific application requirements and are unlikely to be reused as IPs in other designs. Therefore, block based design (BBD) is considered the most appropriate approach to initiate the design. A block based face detection system[55] is an appropriate example where the development process has been organized in three phases from a pre-processing block, detection block to post-processing block design. All key components have been constructed and verified on the module level, such as the Haar feature generator and ANN classifier, before being integrated incrementally into the whole system. A PowerPC based processor was used for memory control and future extension.

The System on Chip design is still a relatively new research field for system development in terms of high level system partitioning, design methodology and specific hardware level requirements. The design varies significantly according to different applications and knowledge of both architecture design and module verification is critical for achieving an optimum result.

Chapter 3 -- Touch Screen System on Chip Design and Implementation

3.1 Introduction

The proposed touch screen system was originally specified to perform two major system functionalities, acquiring touch information and processing touch results, with a number of sub-modules specified by the industry partner, such as an image sensor module and illumination support chip. A number of other sub-units were considered to be investigative, such as most of the blocks in the processing part of the system. Therefore, the traditional block based system on chip design method has been applied to the construction process with each module researched, implemented and tested separately and integrated incrementally as a complete system in the end.

The system has been partitioned into three main parts defined by the functionality on the top level:

- A Data Acquisition System which is designed to configure and control acquisition sub-modules to consistently collect and update touch screen based vision data.
- A Processing Unit which is designed to calculate and analyse the object position and touch motions based on the previous acquisition result.
- A PC module which is planned to handle and update touch positioning and motion results from the hardware block with data then passed to shared memory for future extension.

Each sub-module inside the acquisition and processing systems is dedicated to implementing one function. In addition there is a customized Acquisition Controller organizing all sub-units in the Data Acquisition System and a Master Controller coordinating all sub-modules in the Processing Unit for system synchronization. More details are presented in following paragraphs.

3.2 Data Acquisition System

3.2.1 Data Acquisition System Overview

The Data Acquisition System is the fundamental module inside the touch screen system to continuously acquire and transmit touch information to the next stage, the Processing Unit. The acquisition system is comprised of three major parts: Acquisition Controller which configures and directs the cameras and the LED based illumination subsystem through camera support circuits; ADC Interface which controls the analogue-to-digital converter chip to digitize analogue pixels from the cameras into digital values; and USB Interface which constructs digital pixels into packages and drives the USB chip to transmit them correctly.

Inside the Data Acquisition System, all the control units have been constructed in the FPGA with I/O signals interfacing with outside physical components such as camera support circuits, two image sensors mounted on top of the touch screen, the ADC chip (Texas Instrument) and the USB chip (FTDI) through ribbon cables. The initial block diagram (before optimization) of the Data Acquisition System is illustrated below:

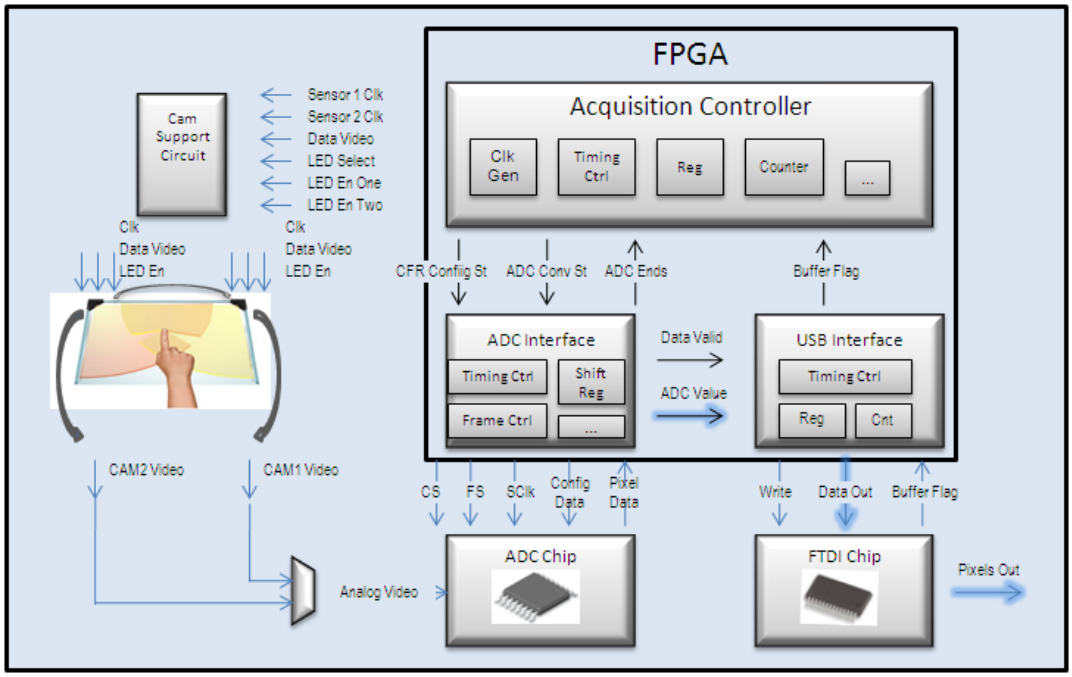


Figure 12 Data Acquisition System Block Diagram

3.2.2 Acquisition Controller

The Acquisition Controller is the central sub-module of the Data Acquisition System, which not only controls the camera and illumination system but also organizes the communication with both the ADC and USB modules. Timing control is one of the most significant parts inside the Acquisition Controller, affecting the system coherence and correct handshaking between sub-modules. Thus, timing control flow is decoded into a state machine with clear transitions among different states. Counter modules are constructed for more accurate timing control inside states in different operations. Meanwhile, pixel and frame counting are required from camera communication. There is also a group of clock dividers in the Acquisition Controller generating clocks at different frequencies for different operation modes. As a basic controller design, the register unit is a necessary element for initialization purposes or restoring intermediate results. The initial design of the Acquisition Controller is illustrated below with the sub-modules mentioned above and other separate logic elements:

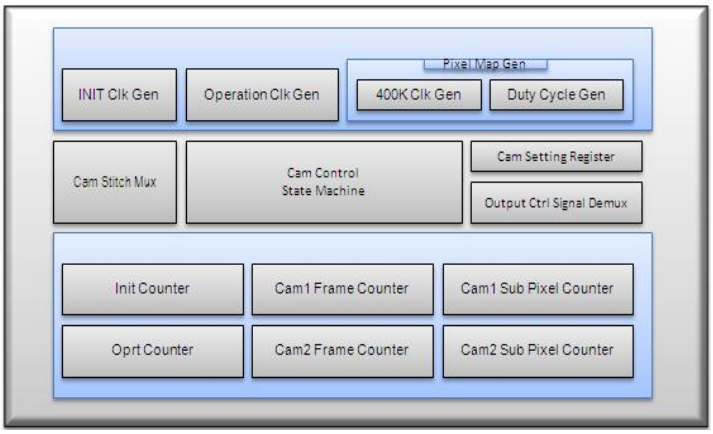


Figure 13 Acquisition Controller Block Diagram

3.2.2.1 Acquisition Controller Timing Control Flow

Acquisition timing control has been organized in four phases: Idle, Configuration, Operation and Transmission with a total number of 23 states describing the flow of the acquisition process. Two additional states (CAM_Setup_Switch and CAM_Tx_Switch) are decoded to switch between the two cameras. Each camera goes through the 23 states before the system switches to the other camera.

The following table lists all states encoded in the Acquisition Controller state machine:

	Name	
1	Idle	Phase 1
2	CAM_Int_Tsu	
3	CAM_Int_Clk_Sync	
4	CAM_Int_Mode_Sel	
5	CAM_Int_Pixel_Map_Tsu	Phase 2
6	CAM_Int_Pixel_Map_Config	
7	CAM_Int_Volt_Curr_Setup	
8	CAM_Int_Reset_To_Idle	
9	CAM_Int_Idle_1	
10	CAM_Int_Idle_2	
11	CAM_Operation_Tsu	
12	CAM_Operation_Clk_Sync	Phase 3
13	CAM_Operation_Mode_Sel_Tsu	
14	CAM_Operation_Mode_Sel	
15	CAM_Operation_LED_En	
16	CAM_Operation_Exposure	
17	CAM_Operation_Exposure_CFR_Config	
18	CAM_Operation_E2	
19	CAM_Operation_E2_Idle	
20	CAM_One_Transmission	Phase 4
21	CAM_One_Transmission_Idle	
22	CAM_Two_Transmission	
23	CAM_Two_Transmission_Idle	
24	CAM_Setup_Switch	
25	CAM_Tx_Switch	

Table 1 Acquisition Controller State transition Table

Phase 1 (Idle): All internal and external control signals are set to their default values.

Phase 2 (Configuration): In the configuration phase, the camera clock is switched to the configuration clock frequency (200 kHz) first, and then the camera mode, camera pixel map, camera voltage and current level are all set up under the configuration clock.

Phase 3 (Operation): In the operation phase, the camera clock is changed from the configuration clock to the operation clock and the camera mode is switched to operation mode. The illumination subsystem is enabled for the camera exposure time (1ms typical), and during this period, the controller is concurrently configuring the ADC register before the transmission phase starts.

Phase 4 (Transmission): In the transmission phase, the camera pixels are transmitted at the transmission clock frequency.

Camera Switching Phase: Camera one and camera two are switched in both configuration and transmission.

3.2.2.2 Clock Divider Modules

The system clock is 50 MHz, however, there are three different frequencies required for different parts of the Data Acquisition System: 200 kHz, 500 kHz and 12.5 MHz. Counter based clock divider modules are designed to derive lower frequencies from the system clock.

For example, the system clock needs to be divided by 100 in order to generate a 500 kHz clock from the 50 MHz system clock. A counter is applied to count from 0 to 49 (50 counts) and then reverses the output clock signal from either 0 to 1 or 1 to 0.

The following is a circuit level design of the clock divider (from 50 MHz to 500 kHz) which is mainly made of multiplexers, an adder and a comparator (for incrementing and checking the count value), and D flip flops used to store counts and synchronize with the reset signal:

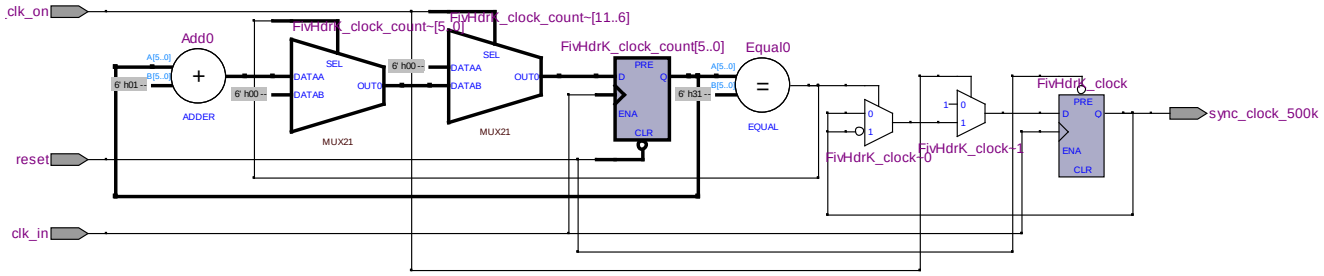


Figure 14 Clock Divider RTL Structure

The same principle has been applied to the 200 kHz and 12.5 MHz dividers which use 125 count and 2 count based designs respectively.

3.2.2.3 Acquisition Engine Counter Modules

In the original design before optimization, there are six counters in the Data Acquisition System: two process counters, two frame counters and two pixel counters. The use of counters is common inside most digital systems and the following is the circuit level design of one process counter (operation counter) which is 14 bits wide:

14 bit operation counter design:

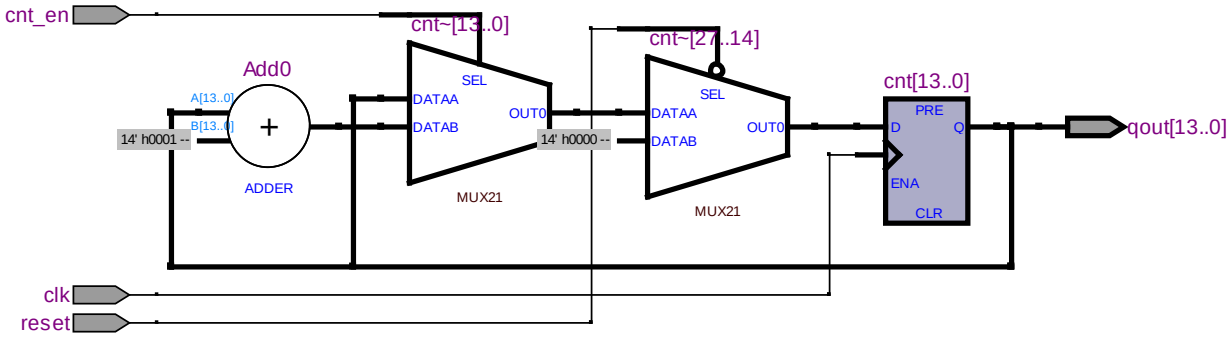


Figure 15 Acquisition Engine Counter Module RTL Structure

In the counter structure, one adder is used to increment the count starting from 14'h0001. When the counter is enabled, the new incremented result will be selected by the first multiplexer, otherwise the old result which was previously stored in the D flip flops will be chosen. The second multiplexer is used to reset the counter value to the default value of zero.

3.2.2.4 Reconfigurable Pixel Map Generator

One important task in the camera configuration process, which is also managed by the Data Acquisition System, is selecting the active imaging area by configuring pixel maps inside both cameras. The active imaging area is different for each camera. Thus it is required that the Data Acquisition System is able to generate all possible combinations in the pixel map to suit each camera's optimal imaging performance. There are ten cycles to configure the pixel map with each cycle having ten bits for activating certain vertical rows in imaging area. The design idea of the Pixel Map Generator is to selectively produce any duty cycle waveforms inside each cycle. From the hardware point of view, the design idea has been translated into a core having two comparators: one for indicating the start the duty cycle with the other indicating the end as well as a counter for adjusting duty cycle accuracy inside the cycle. The initial design is to configure the cameras by a default setting (bit5 and bit6 high with the other bits all low in the 10 bits) which represents only the central two rows are activated out of whole ten rows. So in the hardware structure, a 20% duty cycle is supposed to be generated with adjustable position. The following structure is designed for default pixel map configuration:

Pixel Map Generator Structure for default setting:

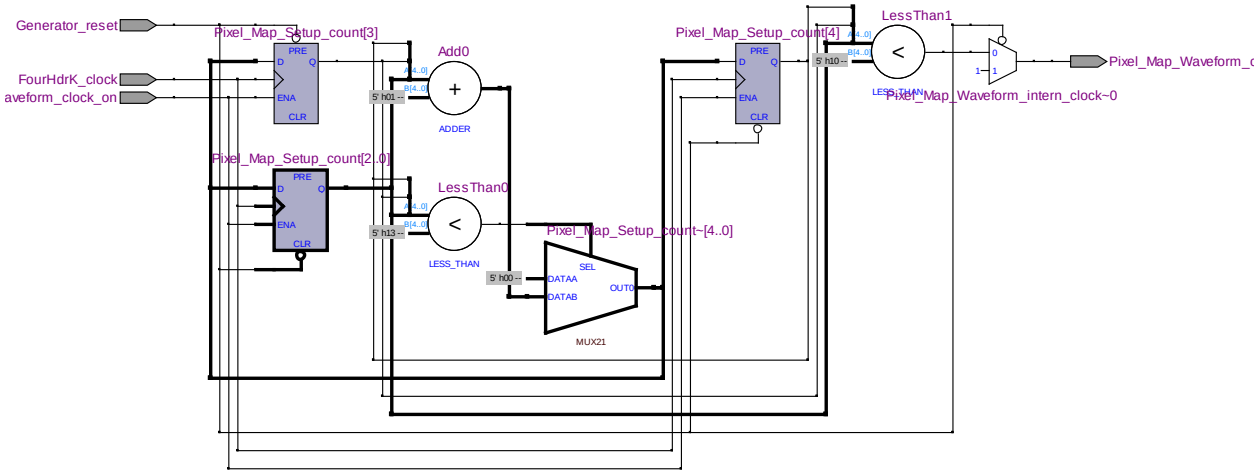


Figure 16 Pixel Map Generator RTL Structure

For the default setting, the counter has 20 counts and the first comparator (left) is compared with 19 (H'13) while the second comparator (right) is compared with 16 (H'10). This produces a duty cycle of 20 % (4 out of 20). The start value of the counter is set to 8 to position the pulse in the middle of the period. Both the start position and duty cycle can be adjusted by modifying the initial counter value and the two compare values.

3.2.2.5 Shift Register Module

In the Data Acquisition System design, a shift register is used to load voltage and current configuration settings at the beginning and these are shifted to the camera bit by bit during the configuration. The following is the D Flip Flop based configuration shift register circuit:

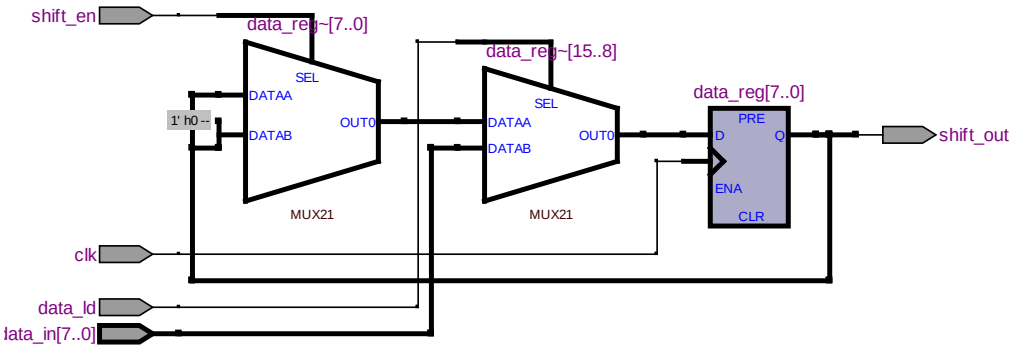


Figure 17 Acquisition Engine Shift Register RTL Structure

3.2.4 Analog to Digital Convertor

The analog pixels output from the camera module are required to be converted into digital values for further processing. An analog-to-digital converter is used to convert the analog values to digital numbers. In our touch screen system, the analog pixels represented by voltage levels are to be converted into a digital format so they can be displayed as binary numbers.

3.2.4.1 ADC Chip Selection and Circuit Configuration

There are three considerations in selecting an appropriate ADC chip. The first one is speed. The camera module is able to transmit analog pixels at a rate up to 1 MHz, so the ideal sampling rate of the ADC chip is greater than this frequency. The second consideration is resolution. The previous microcontroller based acquisition system operated at 10-bit resolution, but a higher resolution is expected in the new system to provide more touch information. The final consideration is synchronization. The ADC conversion process must be able to synchronize with an external clock (from the system clock, outside the ADC module) to achieve complete data acquisition system synchronization.

The ADS 7229 [56] from Texas Instrument has been selected as the ADC chip. This is a low-power, SAR (Successive-approximation) based analog-to-digital converter. It is able to operate at a 1MHz sampling rate which meets speed requirement, it is 12-bit which provides more resolution and the conversion can be programmed to run based on an external clock to achieve synchronization with the system controller. The internal conversion clock logic is illustrated below, with options for using either an internal oscillator or external synchronised clock:

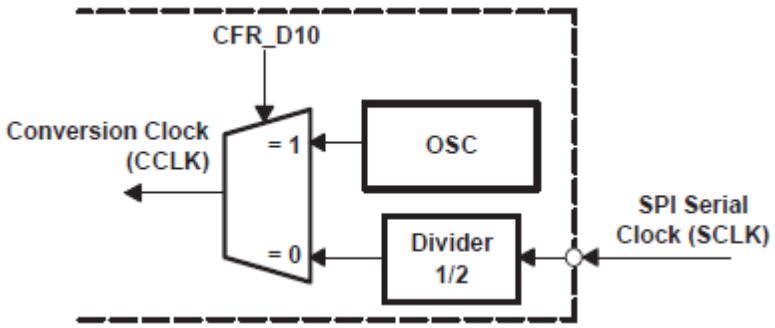


Figure 18 ADC Chip Conversion Clock Block Diagram

Configuration circuit design for ADC 7229:

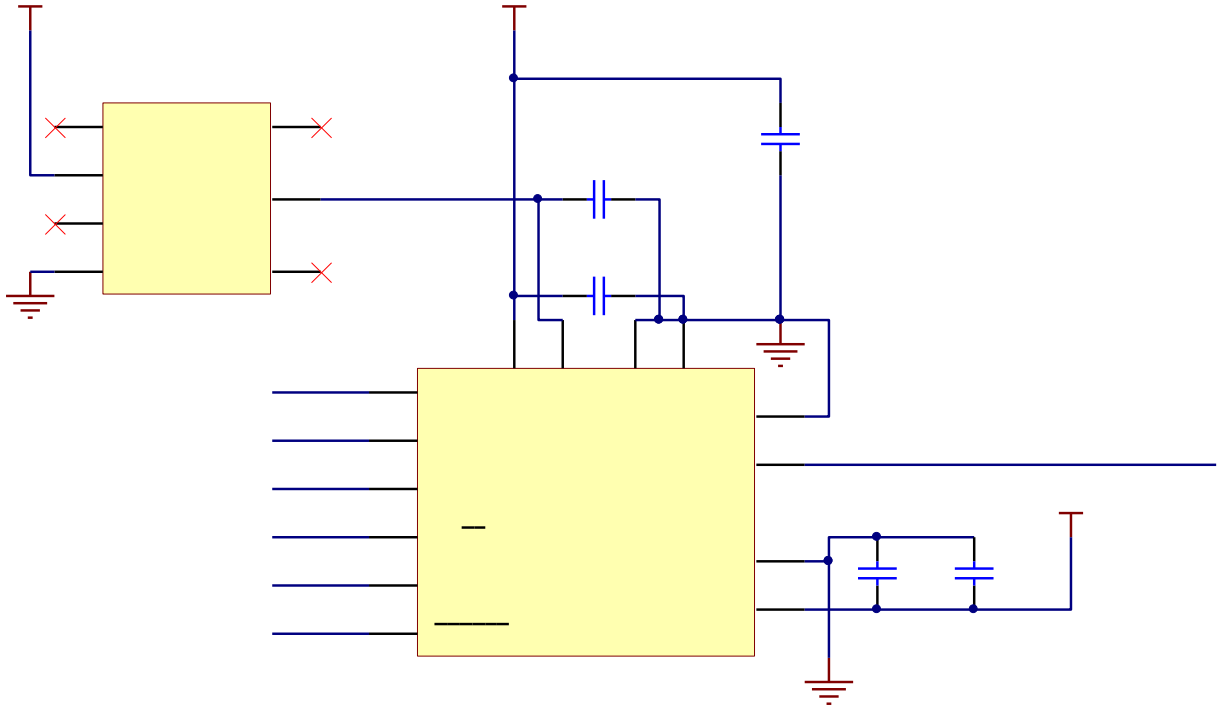


Figure 19 ADC Chip Configuration Circuit

Two pairs of decoupling capacitors are added separately for the analog power supply and the digital power supply to shunt the noise and reduce the effect of the power supplies on each other. An extra low noise, low drift precision voltage reference (REF5025 [57]) is applied to the ADC chip to provide a highly accurate conversion reference. Excellent temperature drift (3ppm/degree) and high accuracy (0.05%) are two main reasons that make this chip ideal for use in high-precision data acquisition system design. The complete ADC circuit (with reference) is able to convert analog pixels which range from 0 to 2.5 volts to two byte binary results (12 bits of value plus 4 bits of extra information). The conversion in the ADC system is synchronized by the Acquisition Controller through SCLK, with CONVST and FS signals sent from the Acquisition Controller to indicate start conversion and start transmission of digitized results. The SDI input is used by the controller to configure the ADC registers and the SDO output is used to reading digitized data. EOC is the feedback signal to indicate the end of conversion to the controller.

3.2.4.2 ADC Interface Overview

In the Data Acquisition system, the ADC Interface sub-module plays an important role in controlling the ADC chip during the digitization. It responds to the Acquisition Controller for initiating and completing the configuration or conversion processes. In the initial design before optimization, the ADC Interface consists of three parts: Control Logic for directing process flow for configuration and conversion, ADC Counter system for handling accurate SPI transfer timing, and ADC Register system for shifting configuration settings and shifting out digitized results.

The ADC Interface block diagram is illustrated below:

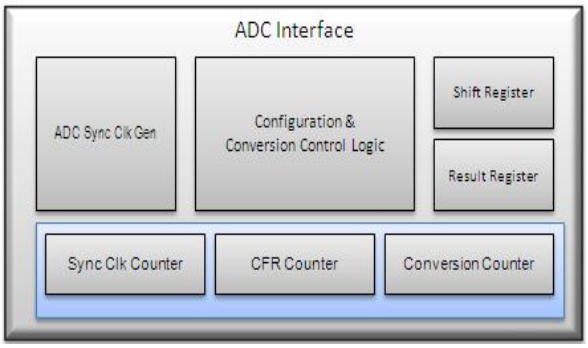


Figure 20 ADC Interface Block Diagram

3.2.4.3 ADC Control Flow

	Name
1	CTRL_CONFIG_ST
2	CTRL_Idle
3	CTRL_CONFIG_STABLZ
4	CTRL_CONFIG_CFR
5	CTRL_CONFIG_CFR_ED
6	CTRL_CONFIG_CFR_ED_2
7	CTRL_CONVERSION_TSU

Phase 1

8	CTRL_CONVERSION_ST
9	CTRL_CONVERSION_EOC
10	CTRL_CONVERSION_FRM_SETUP_1
11	CTRL_CONVERSION_FRM_SETUP_2
12	CTRL_CONVERSION_RD
13	CTRL_CONVERSION_STAB
14	CTRL_CONVERSION_VALID
15	CTRL_CONVERSION_END
16	CTRL_CONVERSION_END_2

Phase 2

Table 2 ADC Interface State Transition Table

ADC control flow has been organized in two phases: configuration phase and conversion phase.

Phase 1 (Configuration): Configuration phase is the process of ADC register configuration (which selects power mode, operation mode and others). It operates at the same time as the

Acquisition Controller exposure which means the Configuration phase does not cost extra system time.

Phase 2 (Conversion): Conversion phase starts when the analog pixel is available from the camera and is signaled by a start command from the Acquisition Controller. It takes 36 system clocks until the conversion process is complete. Then the shift register will be enabled to shift the 16 bit digitized value from the SDO pin of the ADC chip. The data valid signal will be generated at the end of shift to indicate completion.

3.2.4.4 ADC Counter Modules

There are two counters in the ADC Interface: one is for configuration timing counts and the other is for conversion timing control. The counter structure is the same as the operation counter inside the Acquisition Controller design, apart from differences in the count numbers.

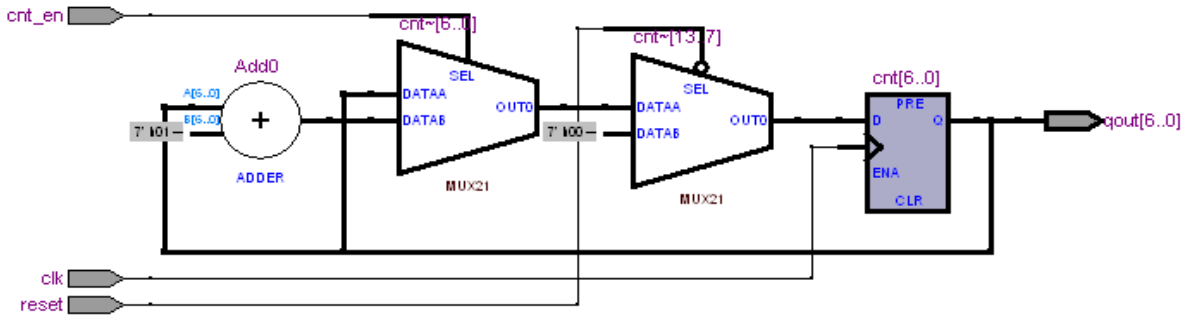


Figure 21 ADC Counter Module RTL Structure

3.2.4.5 ADC Register Modules

Two registers are used in the ADC Interface: one is a configuration register that loads default settings at the beginning and shifts the settings out in serial; the other is a result register which reads the digitised value bit by bit in serial and stores the result temporarilly. A Synchronous Parallel Load, Serial In, and Serial Out shift register is customized for ADC requirements.

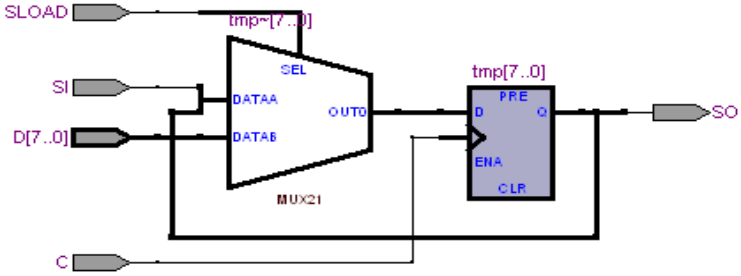


Figure 22 ADC Register RTL Structure

3.2.5 Universal Serial Bus 2.0

Universal Serial Bus (USB) is a common communication method for interfacing external devices with a host controller. In the Data Acquisition System design, a USB2.0 chip FT 245R[58] from FTDI is selected to consistently transmit camera frames to the real time monitor on the PC side for testing purposes. The block diagram of the FT 245R is illustrated below:

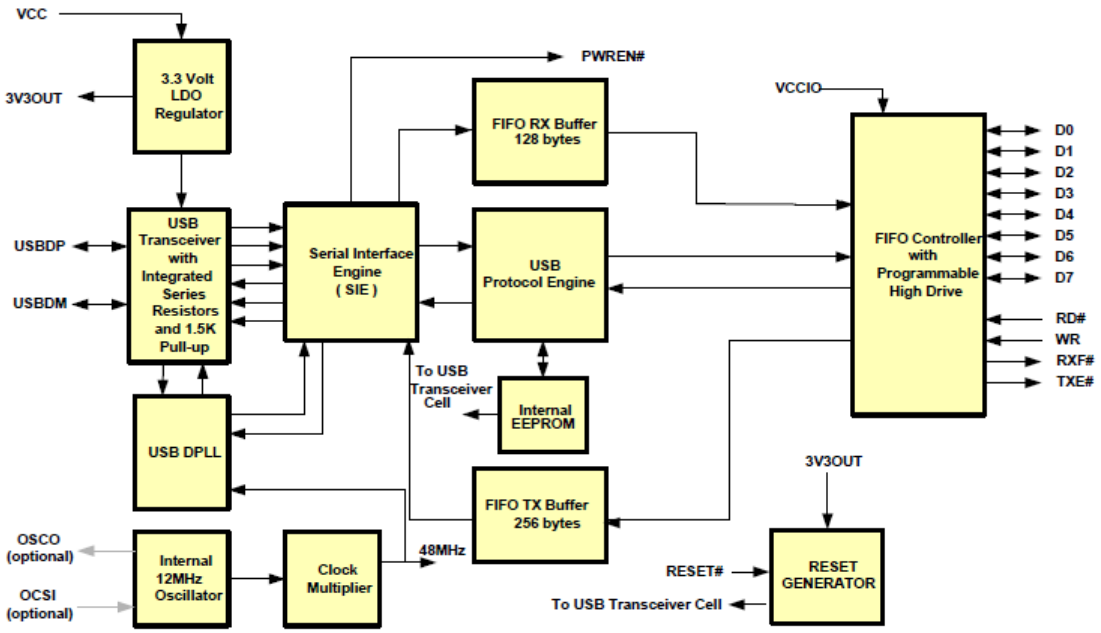


Figure 23 USB Chip Block Diagram

The FT 245R handles the entire USB protocol on the chip with integrated EEPROM storing device descriptions and I/O configuration. The USB Protocol Engine is the key unit inside the FT 245R chip which manages the data stream from the device USB control endpoint and also handles the low level USB protocol requests generated by the USB host controller and the commands for controlling the functional parameters. The Serial Interface Engine (SIE) block is the core data conversion engine which performs the parallel to serial and serial to parallel conversion of the USB data. The USB DPLL cell locks on to the incoming NRZI USB data and generates recovered clock and data signals for the Serial Interface Engine (SIE) block. One important feature of this USB structure is the internal buffers on both the transmission and receiving sides to handle the speed differences between hardware acquisition and the host processor.

3.2.5.1 USB Configuration Circuits Design

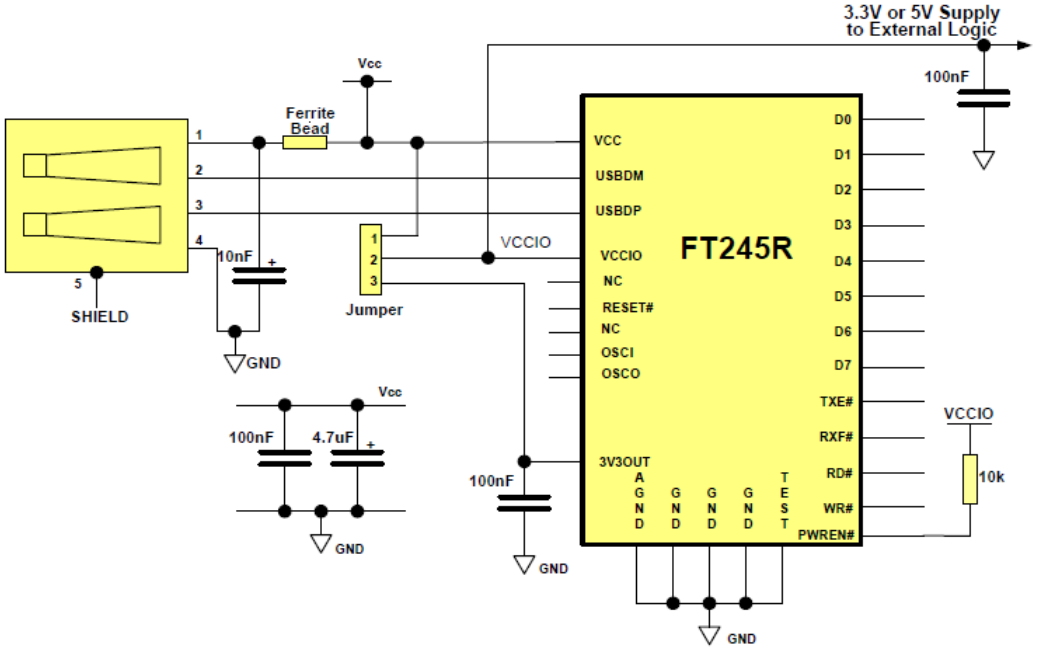


Figure 24 USB Chip Configuration Circuit

The circuit configuration for the FT 245R chip is simple because the chip integrates the external EEPROM, clock circuit and USB resistors onto the device already. The USB circuit is configured as self powered with decoupling capacitors added for noise reduction.

3.2.5.2 USB Transmission Engine Overview

There are two design requirements for the USB Transmission Engine: One is constructing the digitized pixel (12-bit value) into a package which has a header structure indicating the beginning and end of the camera frame to facilitate the synchronization with the software monitor for display purposes. The other is the control of the USB chip to achieve consistent transmission. A simple logic structure is designed to meet both specifications.

The Transmission Engine block diagram is illustrated below:

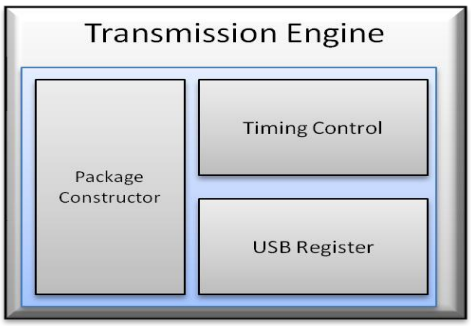


Figure 25 Transmission Engine Block Diagram

3.2.5.3 USB Interface Package Constructor

There are 528 pixels (including dummy pixels) in one camera frame with each pixel comprised of 12 bits. The constructor packs 12 bits into 2 bytes (16 bits) with 4 spare bits providing sync information for the software monitor.

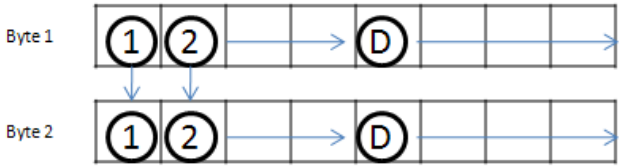


Figure 26 USB Package Constructor Structure

The least significant 12 bits of the 2 bytes (6 bits from each byte) are used to store the 12 bit pixel value. The most significant bits of Byte 1 and Byte 2 are used to represent the beginning and end of frame: 01 means beginning (first pixel), 10 means end (pixel 528), and 00 is used for all pixels in between. The other bit in both Byte 1 and Byte 2 is used to indicate Byte sequence: 0 means Byte 1 and 1 means Byte 2.

3.2.5.4 USB Interface Control Flow

The USB Interface Control Flow is relatively straightforward compared with the Acquisition Controller and the ADC Interface. When pixel data is available from the ADC Interface, a valid flag will be set and the USB Interface will be activated. It will take two cycles to construct the package and then check the buffer is not full before pulling the write signal high to the USB chip to start transmission.

3.2.6 System FIFO

The FIFO (First In First Out) is a common queue processing technique widely applied to hybrid system design which has both hardware accelerate logic and a processor. It is used to buffer and balance the difference in traffic speed between the hardware domain and the host processor domain. In this touch screen system, a hardware synchronous FIFO is integrated between the hardware Data Acquisition System and the PC monitor for real time operating purposes. The hardware FIFO structure is usually made of three parts: Control Logic, Pointer Logic (Read and Write Pointers) and Storage. The synchronous FIFO [59] integrated in the Data Acquisition System is illustrated below on the top level:

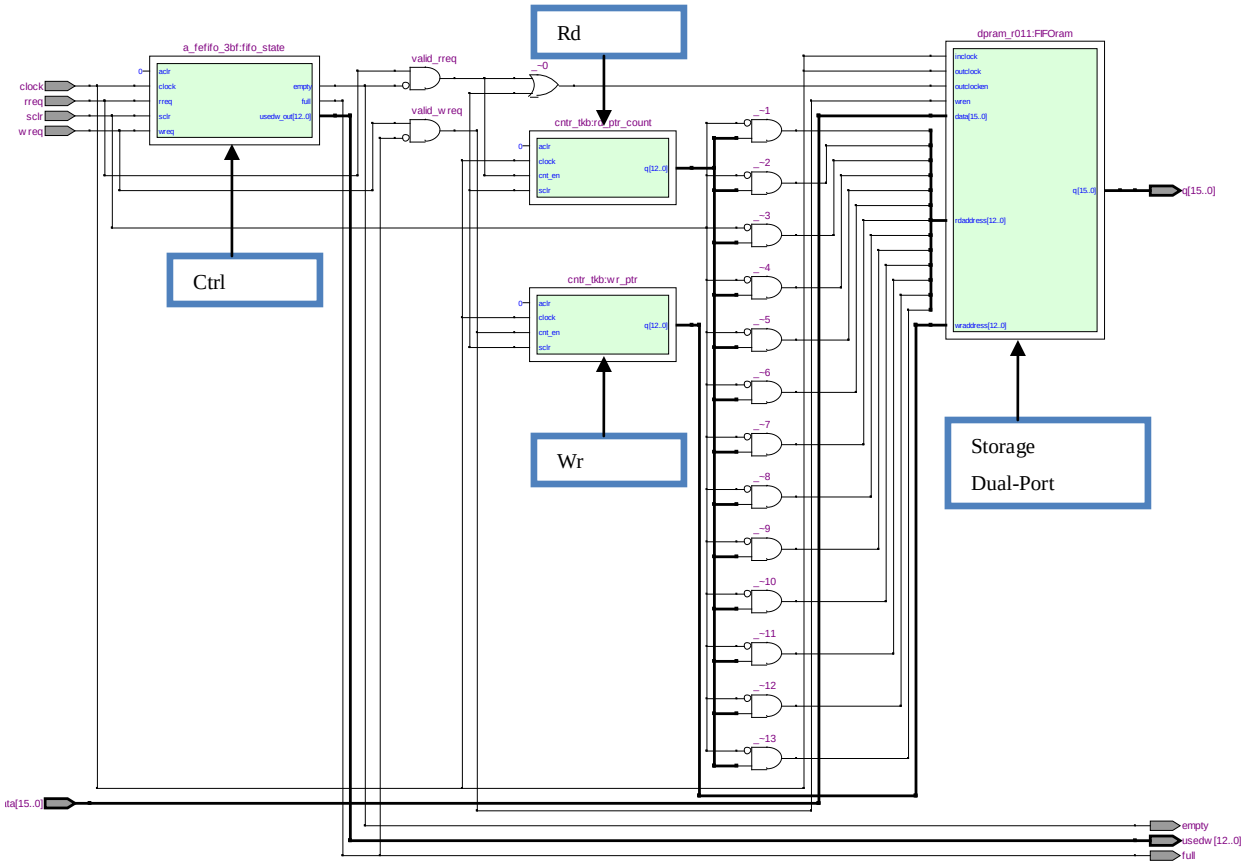


Figure 27 System FIFO RTL Structure

Control Logic:

Control logic has four inputs: clock, synchronous clear, read request and write request. The availability of the FIFO buffer is evaluated inside the control logic with a FIFO Full flag or FIFO Empty flag generated as outputs. The unused width of the FIFO is also tracked by the control logic.

Pointer System:

The counter based pointer system has a read pointer and a write pointer, where one pointer is incrementing and the other is decrementing, depending on whether a read or write process is being performed.

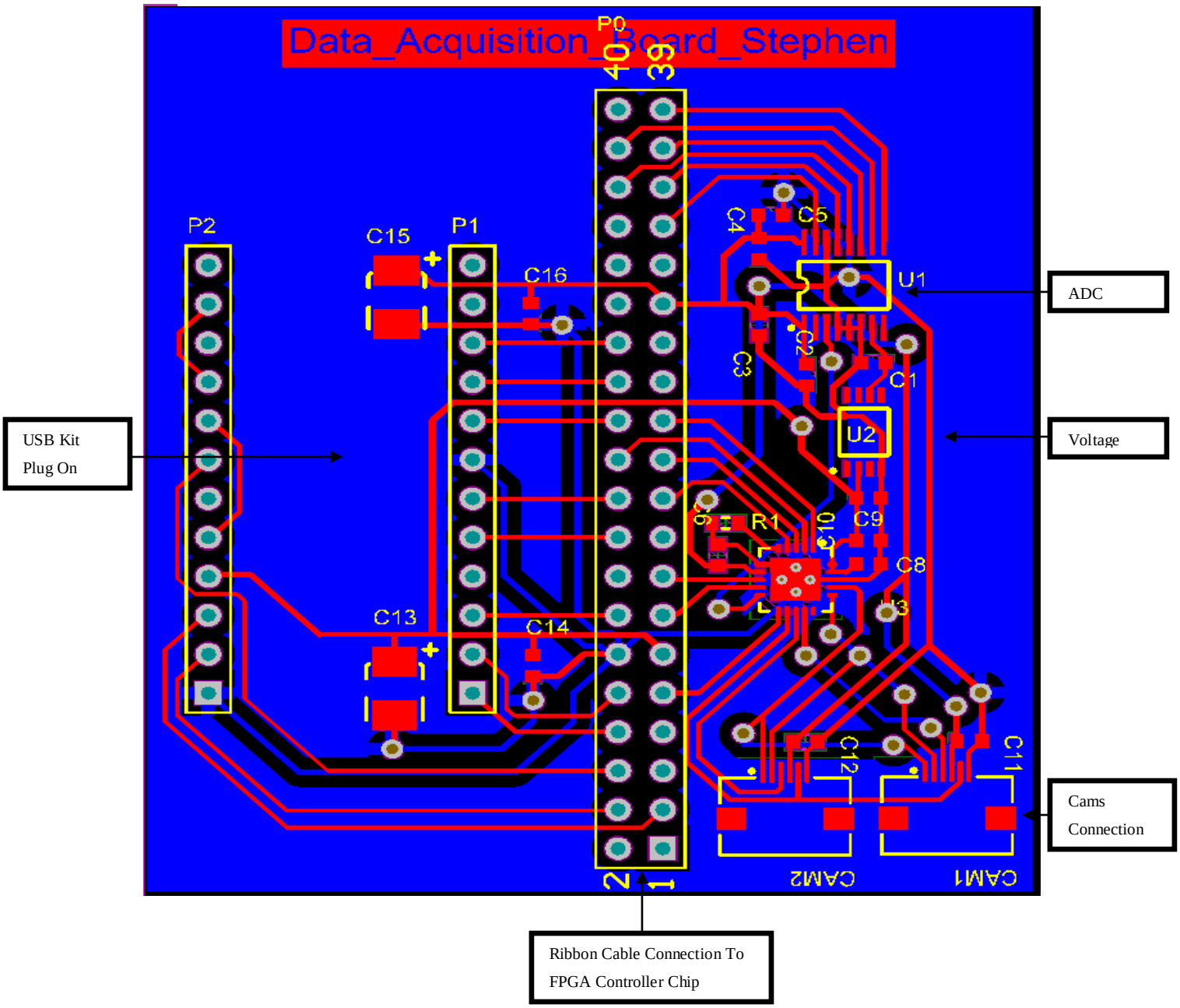
Storage:

In FIFO storage, SRAM, flip-flops or latches are frequently used. In this FIFO design, a dual-port SRAM is created where one port is used for writing and the other is used for reading at the writing or reading addresses generated from the pointer system.

3.2.7 PCB Finalization

The following is the finalized PCB layout for the Data Acquisition Board:

Figure 28 Data Acquisition Board PCB Layout



The following is the physical view for the Data Acquisition Board:

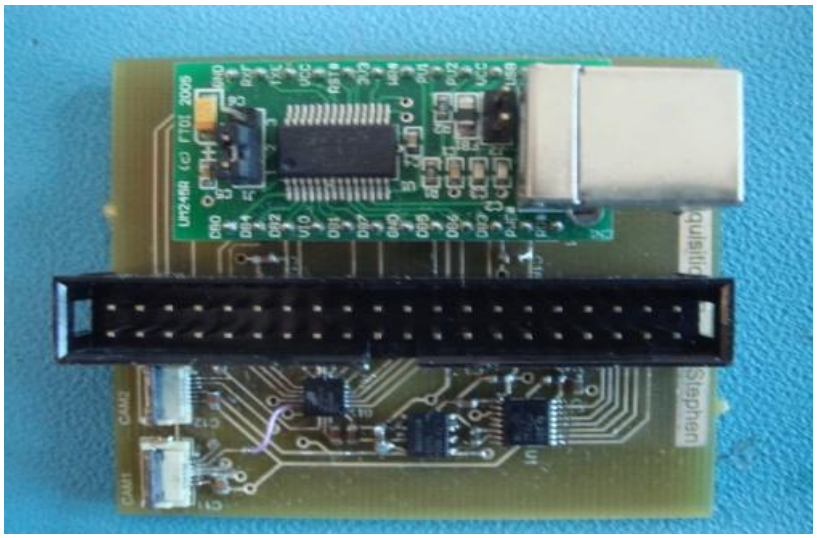


Figure 29 Data Acquisition Board Physical View

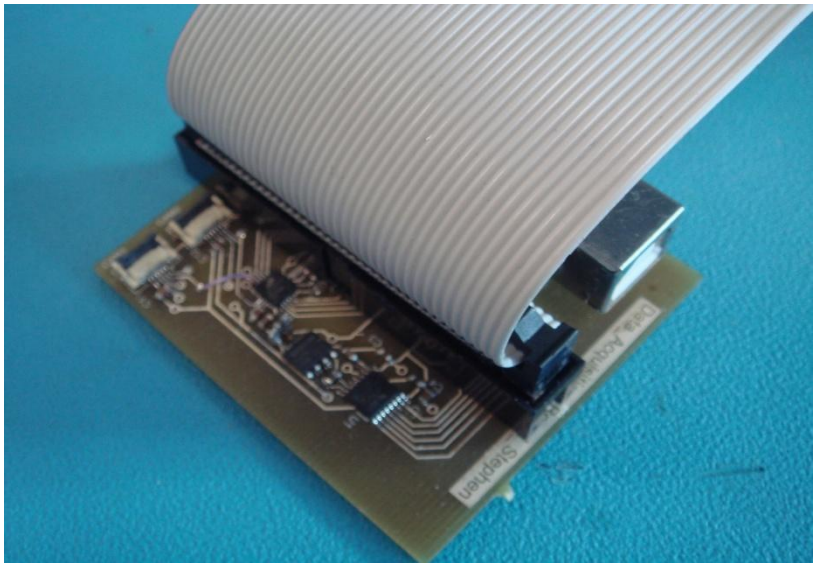


Figure 30 Data Acquisition Board Connected to Dev Board

3.3 Memory Management Unit (MMU)

3.3.1 Memory Elements Overview

In any large digital system design, the memory unit is always one of the most important parts during the system operation. The basic concept of memory is the organization of stored information. Bits are stored in locations specified by an address which is a unique code telling a digital system how to find data that has been previously stored. The simplest memory device, a D-type latch, can store one bit. A 0 or 1 is stored in the latch and remains there until changed. There are a number of types of storage catalogued in terms of volatility, mutability and accessibility.

One of the most common memory types is random access memory (RAM) which can be written to and read from in random order. M4K based Block RAM is the main memory used to store camera pixels in this touch screen system. In read only memory (ROM) data is only able to be read. In the touch screen system design, ROM is used to store look up tables for triangulation. These will be introduced later.

There are two other types of memories considered in the first phase of system design, in case the on-chip memory is insufficient for dealing with extra camera data for potential extended processing. One is SRAM (Static Random Access Memory) which is bistable latching circuitry based storage and can operate up to 167 MHz in the current Cyclone II based design platform[60]. The other one is DRAM (Dynamic Random Access Memory) which is cell (one capacitor and transistor) based storage.

As mentioned, M4k Block based on-chip RAM is selected as the main touch screen system memory to store a total number of 1056 pixels (528*2). An M4K Memory Block is able to operate up to 250 MHz with true dual-port operation and supports four modes of operation: single clock, shared clock, separate clock, and asynchronous. In single clock mode, the read and write operations are synchronous with the same clock while in shared clock mode, the read and write operations are synchronous with the same clock but also a separate clock for the output port.

In separate clock mode, there are two independent clocks (read clock and write clock) for the read/write operations respectively. In asynchronous mode, no clock is required. Both the write operation and the read operation are dependent only on the enable signal.

The basic Read/Write Operation structure is illustrated below:

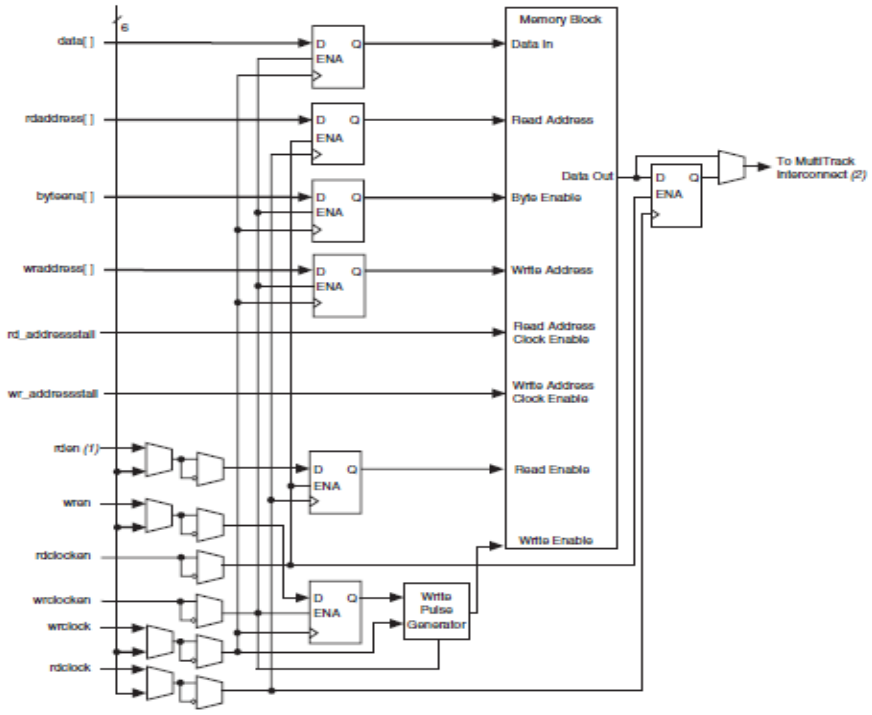


Figure 31 Dual-Port Ram Read/Write Operation Structure

3.3.2 Memory Management Unit Control Structure

After reviewing all the storage elements considered in the initial design planning, a memory management unit control structure is presented below which responds to the master controller to store a reference camera frame, update the latest camera frame and derive the trigger level. The core M4K storage operates in asynchronous mode with separate clock control for memory reading and writing, coordinating with a multiplexer and demultiplexer to switch between the different camera data and propagating through trigger logic to generate a trigger level for next step Edge Detection Unit.

Below is the block diagram of the Memory Management Unit without Updating Logic:

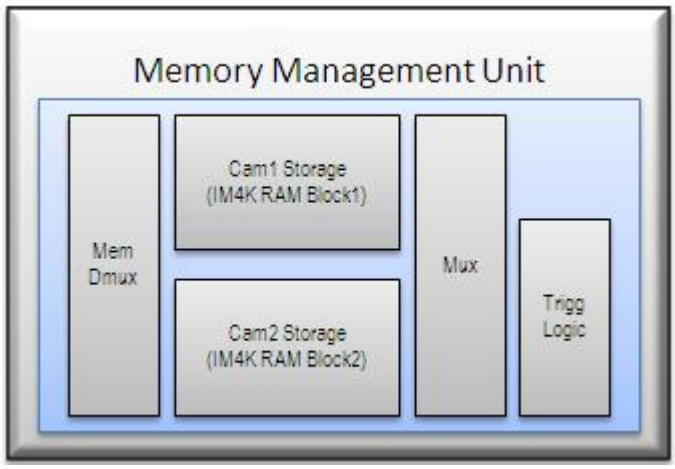


Figure 32 Memory Management Unit Block Diagram

Since a small number of frames are required to be stored (one frame for nor-constrained MMU, three frames for MMU with constrained logic), on-chip M4K RAM blocks are capable of storing the camera pixels for this touch screen system. This RAM also has a high execution speed, using three processing cycles for a write operation and two processing cycles for a memory read operation. The abstract circuit level of the MMU structure is illustrated below:

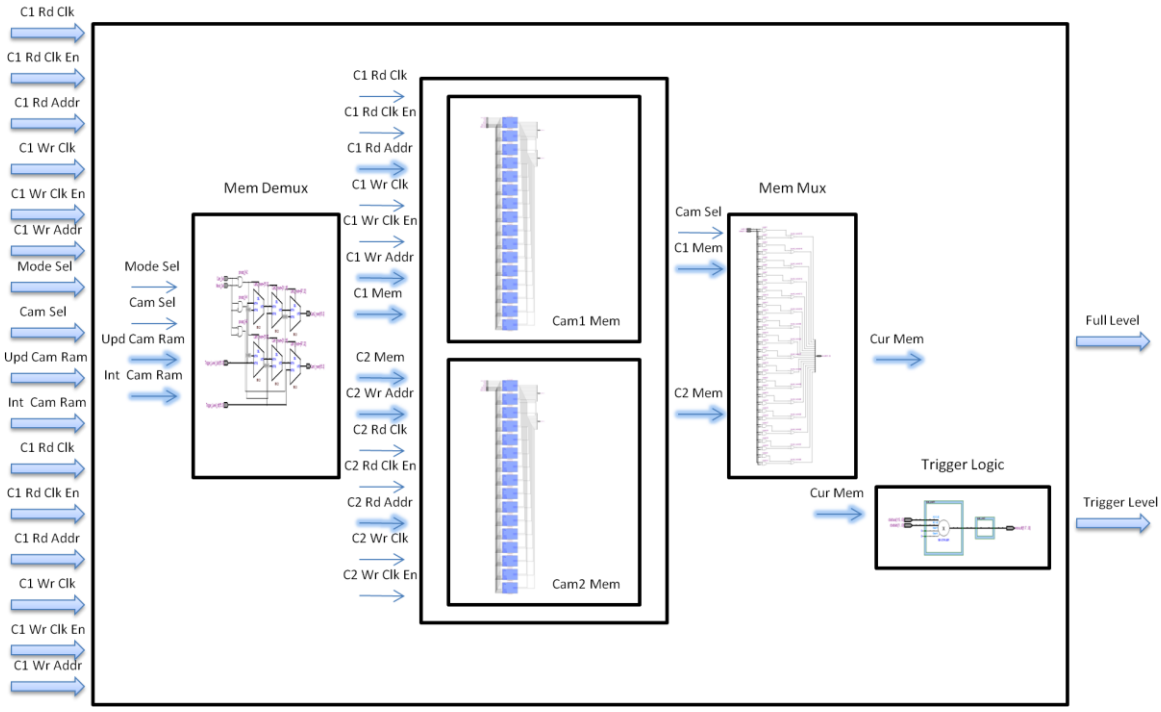


Figure 33 Memory Management Unit Abstracted Circuit Structure

The Memory Management Unit (MMU) is involved in the operation of the Initialization mode and the Operation mode, which are organized by the Master Controller. The access to each camera's memory block is switched by Mem Demux and the output of the memory block is swapped by Mem Mux, both conrolled by the Cam-Sel signal from the Master Controller.

In the initializing process, both cameras' memories are configured by digitized data from the Data Acquisition System after stabilization. All writing related control signals (camera one and two writing clock enable, writing clocks and writing addresses) are enabled, while all reading related control signals are deactivated. The initialization camera frame is the full camera level with the ambient light level subtracted to remove environmental effects.

In operation mode, the Memory Management Unit starts functioning first and finishes last. In detail, when system operation begins, the previous camera pixels and previous trigger values are required to be read from memory for edge detection purpose. It takes three cycles to retrieve these from memory with all the reading related control signals (camera one and two reading clock enable, reading clocks and reading addresses) enabled and all writing related control signals disabled. At the same time, trigger values are generated through Trigger Logic based on the full level pixel values from the memory block.

The design of the Trigger Logic is also a highlight of the MMU. The trigger is specified to be either 75% or 62.5% depending on experimental experience. Creating a floating point unit processor to calculate the percentage would consume a large number of arithmetic logic elements, so a resource-efficient adder-register based solution is used with equivalent results. To generate 75% logic, which is equivalent to ¾, the execution adds itself three times (multiply by 3) with the result shifted 2 bits to the right (divide by 4). To generate 62.5% logic, which is equivalent to 5/8, the execution adds itself five times (multiply by 5) with the result shifted 3 bits to the right (divide by 8).

3.3.3 Constrained Memory Management Unit Structure

During the operation of the real time touch screen system, the trigger level should be updated in a relatively smooth way without significant increases or decreases. Therefore, a customized constrain logic is required to set reasonable limits on the top and bottom range of trigger variations. The design specifications for this constrain logic are that it is one self-contained unit that does not affect other components and then it is as compact and low latency as possible.

The abstract circuit level of the constrained MMU is shown below:

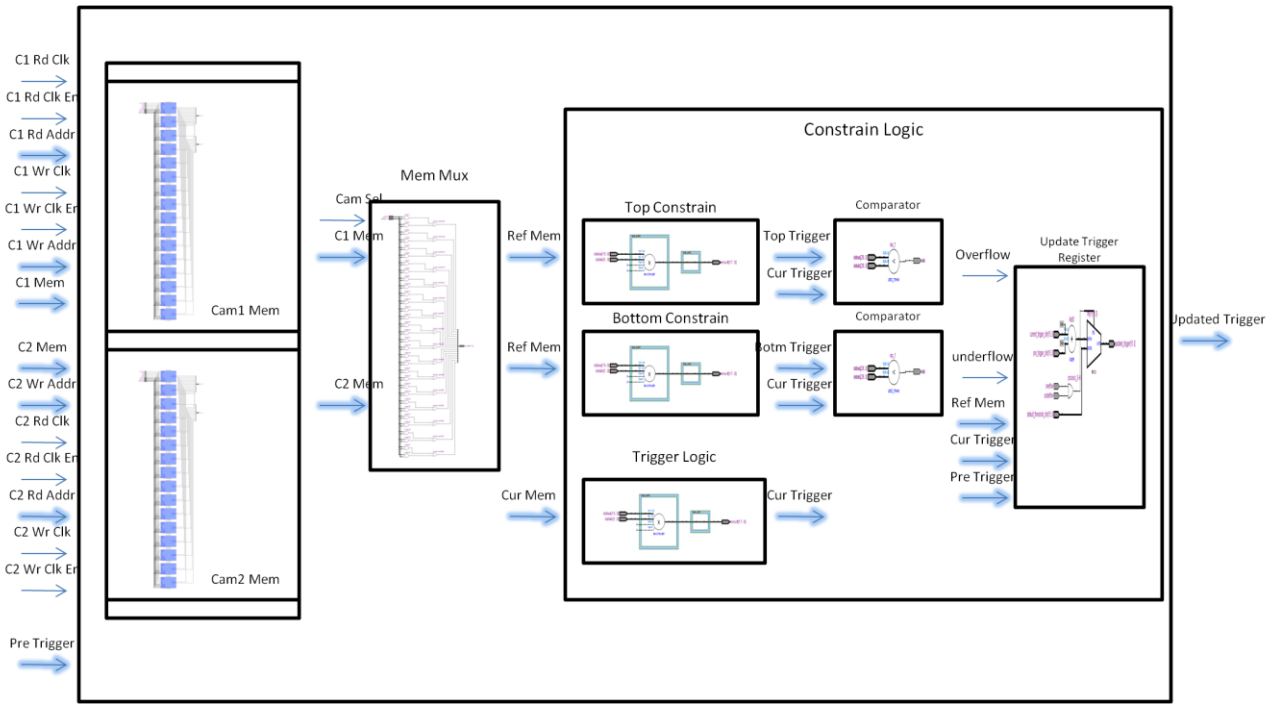


Figure 34 Constrained Memory Management Unit Abstracted Circuit Structure

The Top Constrain and Bottom Constrain components are set to make sure the current trigger level is within a certain range. If the current trigger level is beyond the top limit, an overflow flag will be generated and an underflow flag is indicated by the comparator if the new trigger level is below the bottom limit. The Trigger Update Register adjusts the new trigger level based on the overflow and underflow flags. If either flag is high, which means it is out of normal operation range, the trigger level will be reset to the reference default trigger value. If the current trigger value is within the safe range, the trigger level will be updated by averaging previous and current trigger values.

3.4 Edge Localization Unit (ELU)

Once a touch event happens, the touch object blocks part of the reflection seen from one image sensor while it blocks a different part of the reflection seen from the other sensor:

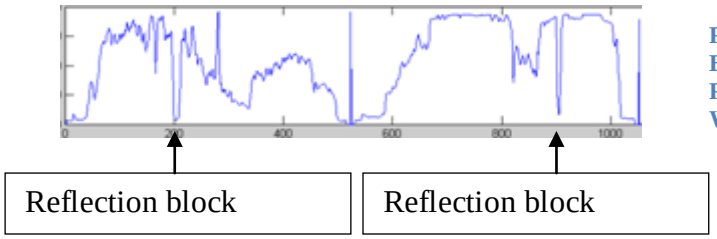


Figure 35 Reflection Block on Left and Right Image Waveforms

The purpose of edge localization is to capture the pixel ID for both the rising and falling edges of the touch object, to localize the object on each camera before further triangulated in the next processing unit.

3.4.1 Edge Localization Unit Overview

Gradient-based edge detection is one of the most well known methods in the field of image processing. It s a discrete differentiation based technique with the position of the local maximum of the first derivative considered as the edge point as defined in Yasri and Hamid's paper [61]. Mathematically, for a 2D image function f(x, y), the gradient magnitude g(x, y) and the gradient direction Θ (x, y) are computed as:

$$G(x,y) = \sqrt{\Delta x^2 + \Delta y^2}$$

$$\theta(x,y) = \arctan\left(\frac{\Delta y}{\Delta x}\right)$$

The edge gradient is calculated from the difference of the pixels in the horizontal and vertical directions. G(x, y) (magnitude) is the sum of the magnitudes of the differences, while the gradient's direction is the arctangent of the ratio of the differences.

The basic concept of gradient based edge detection is originally designed for two dimensional image processing. In order to modify the basic detection idea to this touch screen system using a single line scan camera, simplification and transformation have been applied to create a one dimensional gradient based hardware edge detection structure.

In a one dimensional image, gradient based detection can be simplified to searching for the maximum of the derivative of y in the x direction, which can be further reduced to finding largest vertical difference since all pixels are equally distributed on the horizontal axis.

The design specification requires localizing and storing the touch edge pixel IDs into registers on both the rising and falling edges, by locating the largest vertical magnitude changes.

Gradient Based Edge Detection Logic block diagram is illustrated below:

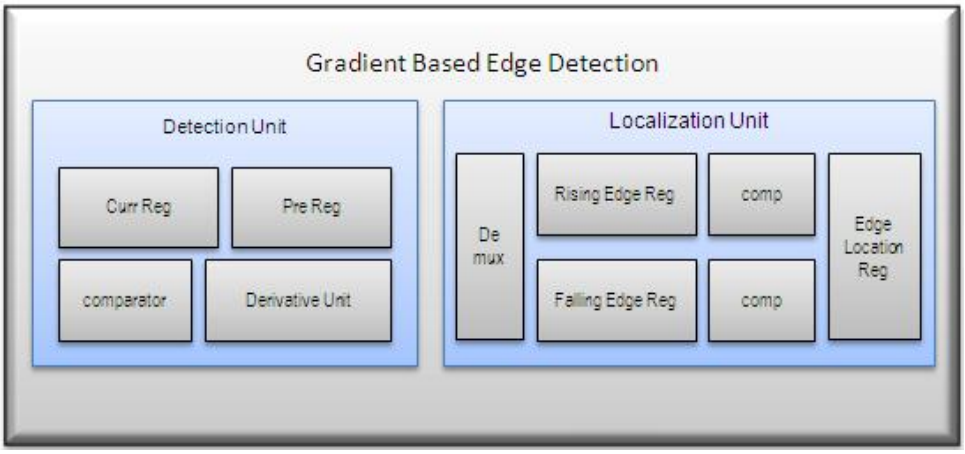


Figure 36 Gradient Based Edge Detection Block Diagram

The Gradient Edge Detection Logic is made up of a Detection Unit and a Localization Unit:

3.4.2 Edge Detection Unit

The Edge Detection Unit is used to generate the flag when a touch occurs. The derivative is calculated by subtracting two registers.

3.4.3 Edge Localization Unit

The Localization Unit is used to capture the pixel ID at both the maximum rising derivative and the maximum falling derivative.

The abstracted circuit level is depicted below:

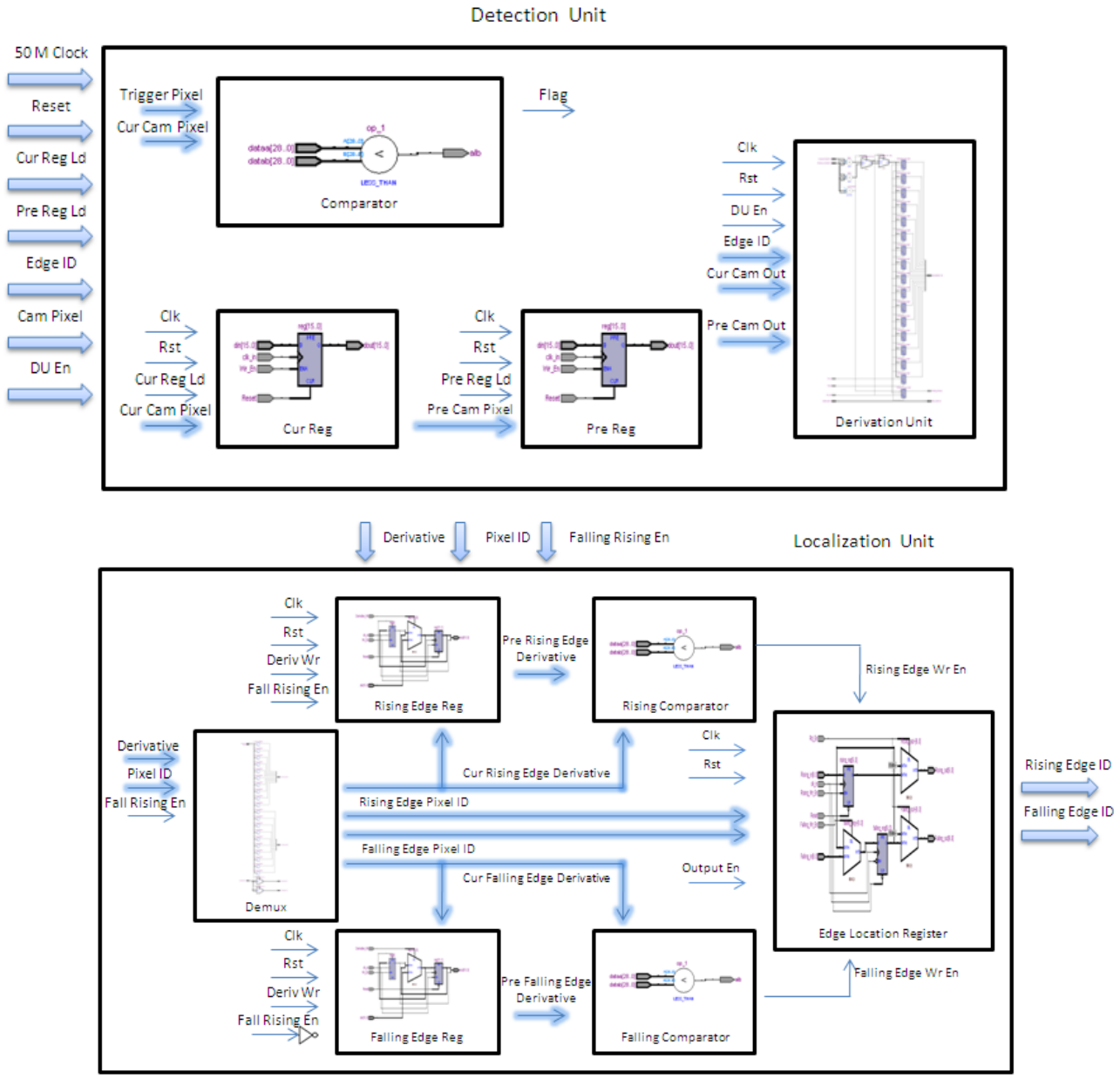


Figure 37 Gradient Based Detection and Localization Units Abstracted Circuit Structure

Process Description:

There are two major units for locating the edge positions on both the rising and falling event: the Detection Unit and the Localization Unit. Inside the Detection Unit, a comparator is used to indicate the touch event by issuing a flag calculated on a preset trigger value and the current camera value. Once the current camera value is below the trigger value, a flag will be generated. Two registers are involved in the DU module: Cur Reg is applied to record the current camera value while the other Pre Reg is used to store the previous camera reading. Both register results will be fetched into the Derivative Unit when triggered by the Master Controller to calculate the current derivative value.

When the current derivative is available at the Detection Unit, it will be directed into either the rising branch or the falling branch, determined by the Falling_Rising flag. In the LU (Localization Unit), both the maximum rising edge and the maximum falling edge derivatives are stored in the Rising Edge Reg and the Falling Edge Reg respectively. After that, either the rising or falling comparator may be activated to indicate the current derivative value is the largest rising or falling value so far in this frame. If either comparator is activated, the largest derivative value will be maintained with its pixel ID captured simultaneously in the Edge Localization Register. In the end, both the falling edge pixel ID and the rising edge pixel ID will be output to the subsequent processing unit for further processing.

The edge detection and localization unit has a conventional basis, customized for the line scan based optical touch screen. The architecture is resource oriented with a small amount of parallel operation. A total of 10 cycles are consumed in the worst case to process a complete edge detection and obtain localization function results.

3.5 Position Localization Unit (PLU)

The Position Localization Unit is a unique and fundamental part of the touch screen system, which processes the final touch coordinates based on the edge positions from the previous Edge Detection Unit. The Position Localization Unit is customized for the physical layout of the touch screen which has image sensors (two in a basic system) on the top corners with the reflective retro glued on the screen frame. Once an object is detected, the middle of the object is calculated by the Edge Detection Unit, and then the tangent values of the camera view angles are read from the LUT. Finally the Position Localization Unit processes and finalizes the whole operation using the results from all the previous modules.

3.5.1 Position Localization Mechanism

Since the Position Localization Unit is constructed in hardware, its mechanism should be efficient and simple. However, a high level of accuracy is required to meet the minimum design specification. The object localization mechanism is illustrated below:

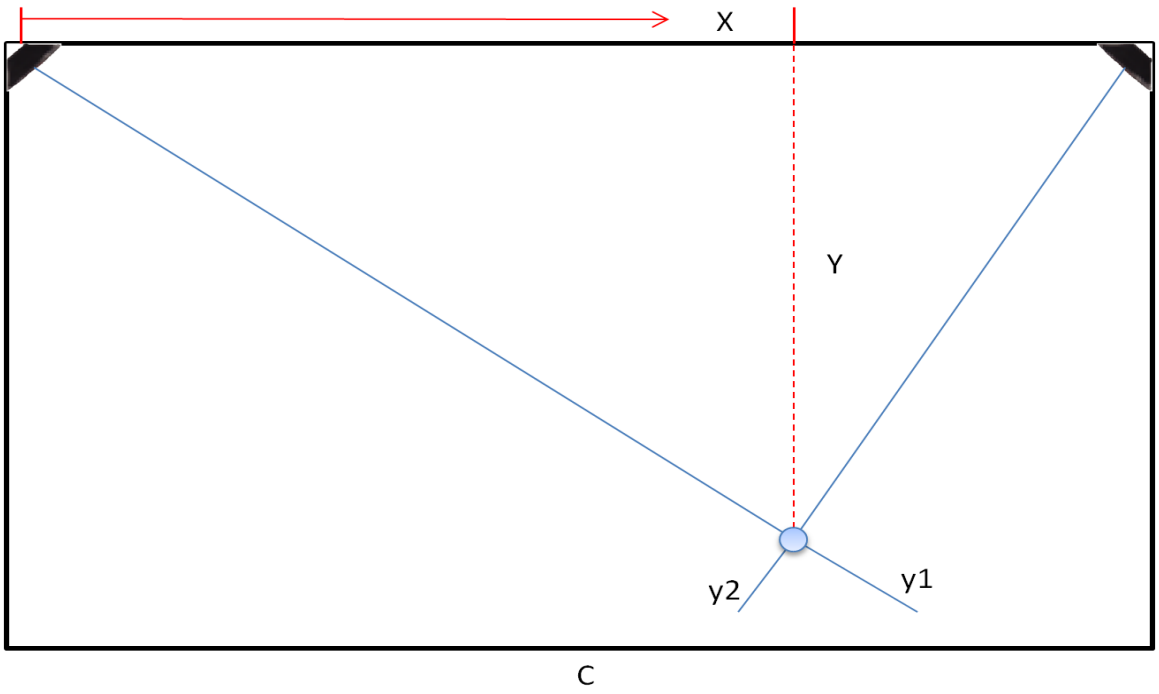


Figure 38 Position Localization Mechanism Illustration

When a touch occurs, the object can be seen by the two cameras along the lines represented by two linear equations in x and y:

$$y = \tan((pi/2 * Num_{active}) * PixEdge1 - offset) * x1$$

$$y = \tan((pi/2 * Num_{active}) * PixEdge2 - offset) * (C - x2)$$

where Num_{active} is the number of active pixels between the start and end pixels of the camera,
$(pi/2 * Num_{active})$ assumes the 90 degree view is equally distributed among the active pixels,
$PixEdge1$ is the middle pixel of the touch position seen from camera one,
$PixEdge2$ is the middle pixel of the touch position seen from camera two,
$offset$ allows calibration for the actual camera placement,
C is the screen width.

After the touch object is seen by both cameras, the coordinates of the touch point are found by solving these two simultaneous linear equations. In hardware, the object localization can be performed by searching for the x value that makes these two linear equations have the same value for y. In a real implementation, the required accuracy is a maximum difference of 1mm between the two calculated y values.

3.5.2 Position Localization Unit Structure Transformation

In order to design the hardware structure, the localization mechanism has to be converted into a hardware style description. The transformation process from the original mechanism to the proposed hardware structure is presented in following stages:

Stage one:

The localization mechanism can be simplified as finding an x value that makes y1 equal to y2 in the following equations:

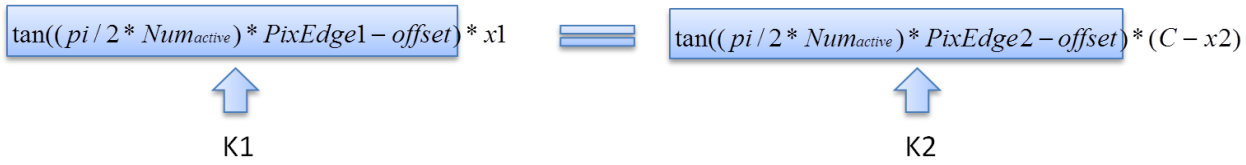
$$y1 = \tan((pi/2 * Num_{active}) * PixEdge1 - offset) * x1$$

$$=$$

$$y2 = \tan((pi/2 * Num_{active}) * PixEdge2 - offset) * (C - x2)$$

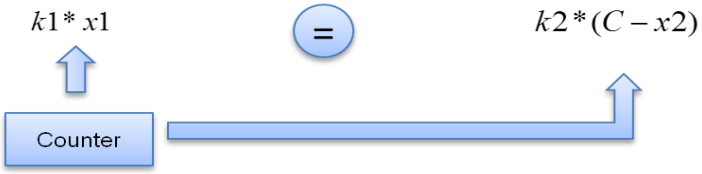
Stage two:

The calculation of the tangent value is computationally expensive and is not suitable to be implemented directly into a hardware structure. As explained in a previous section, a look up table will be used to perform the tangent calculation. For the position localization structure, both tangent terms can be replaced by coefficients k1 and k2 from the LUT:

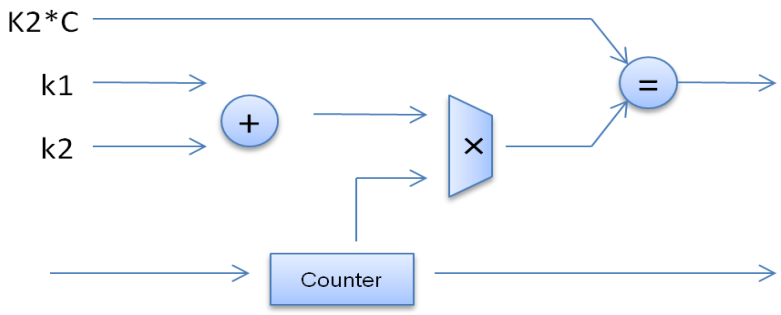


Stage three:

The searching for the x value solution is realized by constructing a counter incrementing from zero to the screen width to find the x value that makes the two equations produce the same value for y.



After stage three, the basic localization idea is translated into an efficient hardware core with one multiplier, one adder and one comparator:



Stage four:

With a normal up counter, using a 24 inch (532mm width) touch screen as an example, in the worst case, the localization unit will take 532 system cycles to locate the x and y coordinates. From the system point of view, 532 cycles does not satisfy the timing specification for the Position Localization Unit. Thus, customized logic is required to accelerate the x searching process. In this chapter, a hardware binary search engine based Position Localization Unit solution is developed for the touch screen application. The main idea is illustrated below:

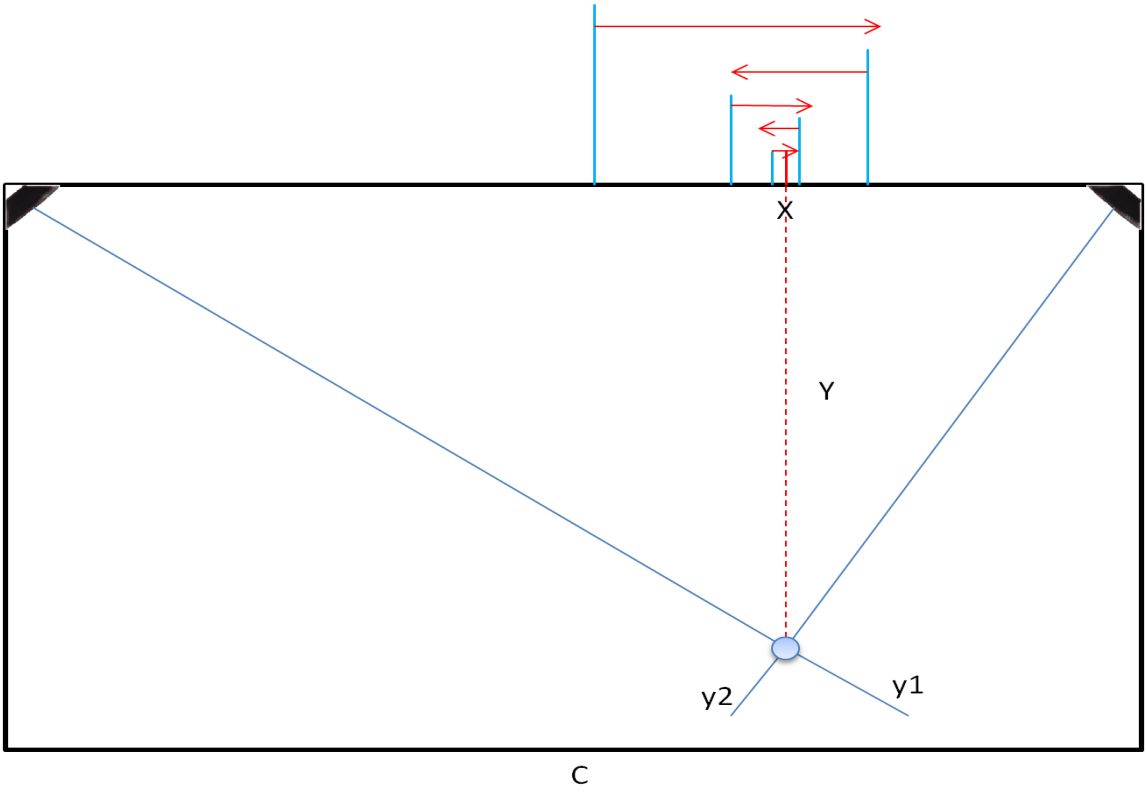


Figure 39 Binary Search Based Object Localization Mechanism

Instead of searching from the beginning of the screen to the end, the search always starts from the middle of the search region. Then, the next search value will be evaluated using arithmetic logic to calculate the optimal next search direction. If the search goal is greater than the current search value, the next search will start from middle of the upper region based on the current value, otherwise it will start from middle of the lower region. Still using the 24 inch screen as an example, the binary search based structure will require only 10 cycles in the worst case (because $log_2^{532} < 10)$ which satisfies the system timing requirements.

3.5.4 Position Localization Unit Structure Overview

The Object Localization Unit was implemented after the transformation from the localization concept to the hardware description. The three major modules are the Binary Search Engine, Triangulation operator and Results Register. They have been coordinated to calculate the touch position in an efficient way by taking advantage of the hardware structure.

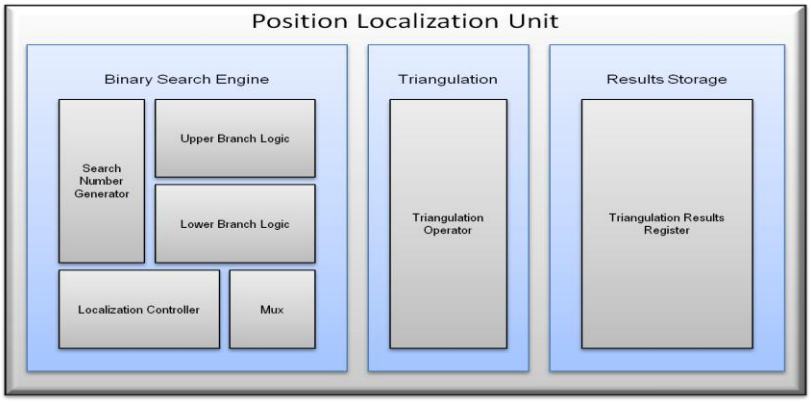


Figure 40 Position Localization Unit Block Diagram

At the beginning of the search process, the Search Number Generator produces the first search candidate which is the middle value of the default screen width (x axis). The candidate is fed into the Triangulation module which calculates the corresponding y values from the left and right cameras' perspectives. The difference between the two y values is evaluated inside the Triangulation module. If the difference is within the required accuracy, a touch found flag is raised to finish the search routine, otherwise a direction flag guides the next search number generation (the middle of the lower range or the middle of the higher range) by propagating through the dedicated logic controlled by the localization controller. The Results register is used to store the final touch location result. This continues until the touch found flag is raised. The Position Localization Unit (PLU) is designed in a self-contained manner with its own independent module controller (Localization Controller) coordinated with the system Master Controller.

3.5.3 Binary Search Engine

The binary search or half-interval search is a conventional and efficient position localization process that divides the search space in half on each iteration. The hardware Binary Search Engine is designed based on the binary search algorithm to optimize the touch position searching process in order to meet the system timing requirements. It has five sub components contributing to the binary search process: a search number generator, upper and lower branch logic, a multiplexer and the binary search controller. The abstracted circuit level is illustrated below:

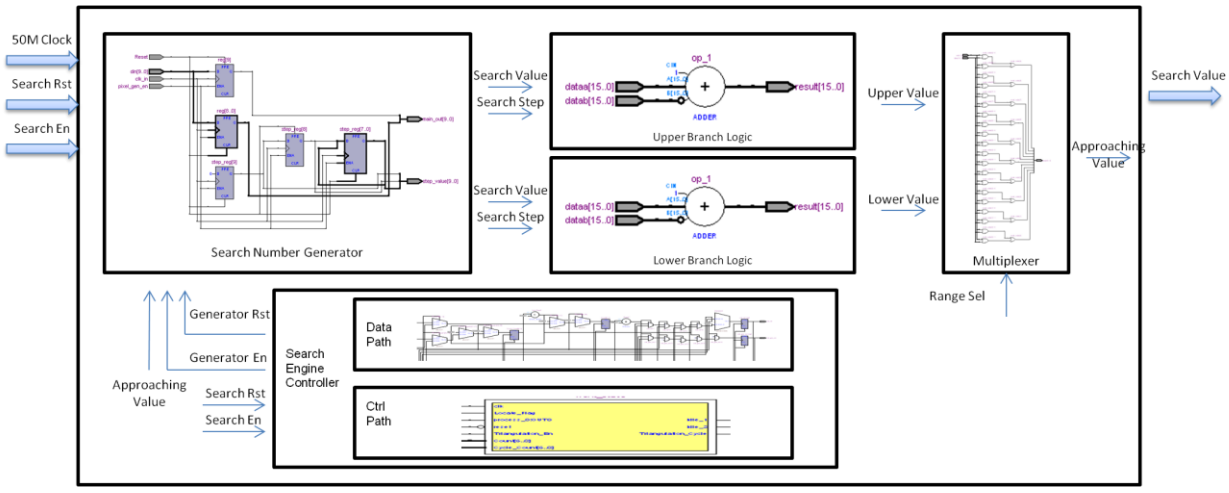


Figure 41 Binary Search Engine Abstracted Circuit Structure

When the Binary Search Engine is activated by the Master Controller, the Search Engine Controller will configure the search value and the search step value inside the Search Number Generator to the default starting values, which are determined by the application screen size. The search value is the search attempt to locate the correct x value. If the current search value does not make the two camera linear functions match, a new optimal search value is derived from the current value through either the Upper Branch Logic or Lower Branch Logic guided by the Search Controller. In the hardware design, the Upper Branch Logic is basically an adder that adds the search step value to the current search value, while the Lower Branch Logic is a subtractor that subtracts the search step value from the current search value, to rapidly approach required solution. The search step value is originally set to ¼ of the screen width and is then right shifted on each search cycle (so its magnitude is half the value of previous step) to generate

a more and more accurate search attempt. The Multiplexer is directed by the operator evaluation result to select either the upper search value or the lower search value for next cycle, until the search goal is achieved.

3.5.4 Triangulation Operator

The Triangulation Operator is the arithmetic unit constructed inside the hardware localization core and consists of a number of arithmetic operators (introduced in the aforementioned transformation stage three with one multiplier, one adder and one comparator) to complete the localization process. Extra arithmetic resources are used here to compare the two y values and generate a Region-Toward-Flag for the Binary Search Engine. A small difference is allowed in the linear function results matching, and implementing this also consumes extra resources.

3.5.5 Position Localization Unit Registers

The Position Localization Unit Register is used to capture the current x value when the current search routine is accomplished. The Search-Finish-Flag and Search-Towards-Flag (which causes the multiplexer to select either the upper or lower branch value) are both temporarily stored in the OLU Register which will be read by the Search Engine Controller during the step of the search operation.

3.6 System Look Up Table

In system design, a Look Up Table (LUT) is usually built to perform some critical task such as providing a mathematical function or conversion. Some simple arithmetic operations such as addition, subtraction or even multiplication can be constructed straightforwardly at a relatively low computational cost and latency without implementing a look up table. More complicated arithmetic operations such as division, square root, sine/cosine, and tangent/arc tangent are both resource-consuming and cycle-consuming and not economical to be implemented directly under the constrained hardware resources and required latency of this application.

For this touch screen system, both touch edges on the two cameras are required to be converted to tangent values ranging from 0 to $\pi/2$ for the next stage position localization processing. This conversion is not only computationally complex but also on the system critical path with maximum processing cycle limitation. Therefore, a Look Up Table (LUT) is considered to be an appropriate solution, using a trade-off between logic cells and memory blocks.

Read only memory (ROM) is selected as the memory component for storing a look up table, since the contents of the LUT are read by the Position Localization Unit (PLU) only after memory initialization. The block diagram of the look up table is illustrated below:

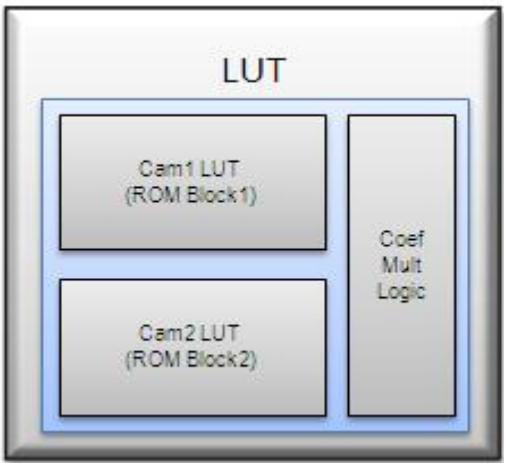


Figure 42 Look Up Table Block Diagram

The abstract circuit level of LUT is illustrated below:

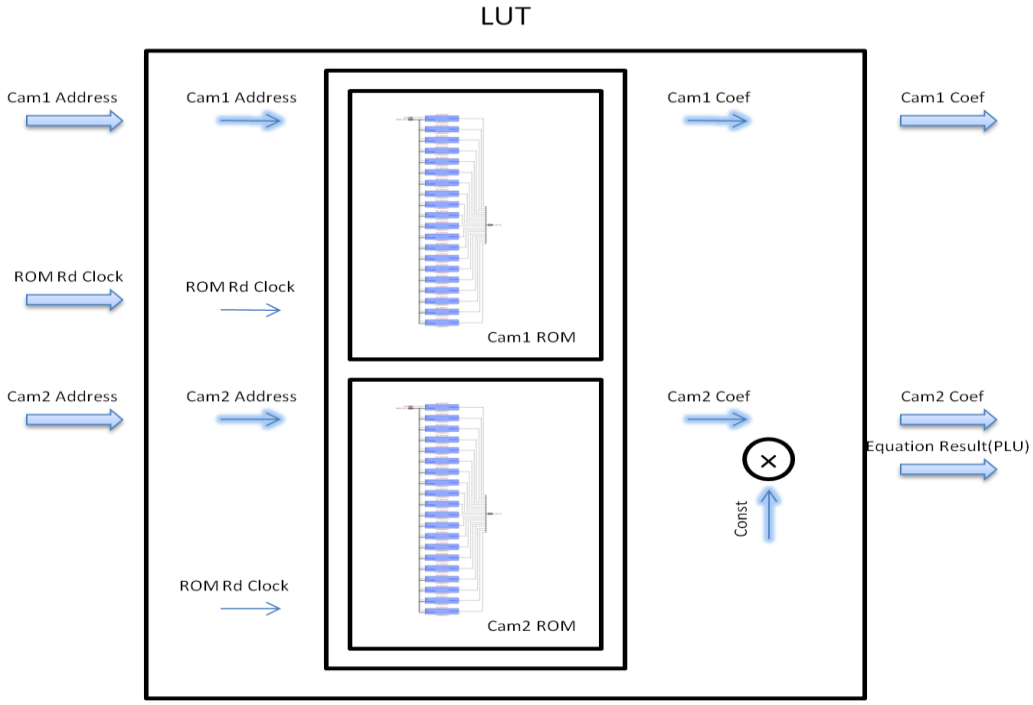


Figure 43 Look up Table Abstracted Circuit Structure

The design purpose of the look up table (LUT) in the touch screen system is to translate the camera one and camera two pixel positions (IDs) into tangent values of the real angles corresponding to the pixel ids. The content of the LUT is pre-calculated using Matlab, and configured into a memory initialization file (MIF) in hex format. The Look up table is enabled by activating the ROM Rd Clk signal from the Master Controller, with the specific table element access determined by either the cam1 or cam2 address. The outputs of the LUT are the corresponding tangent values of the input camera pixel ids. In addition, one multiplier in the LUT multiplies one tangent value by C for the following Position Localization Unit (PLU) structure.

The image sensor used in the touch screen system generally has a 90 degree range of view, and ideally the 90 degrees would be equally distributed between the start pixel id and the end pixel id. In the actual implementation, the range between the start pixel and the end pixel is typically 450-500 pixels out of the total 528 pixels per sensor. The Matlab look up table calculation is based on the ideal case since the proposed system is concept-proofing. A more accurate look up table could be generated from calibrated measurements.

X axis is the pixel id (position), Y axis is the tangent value corresponding to the pixel id:

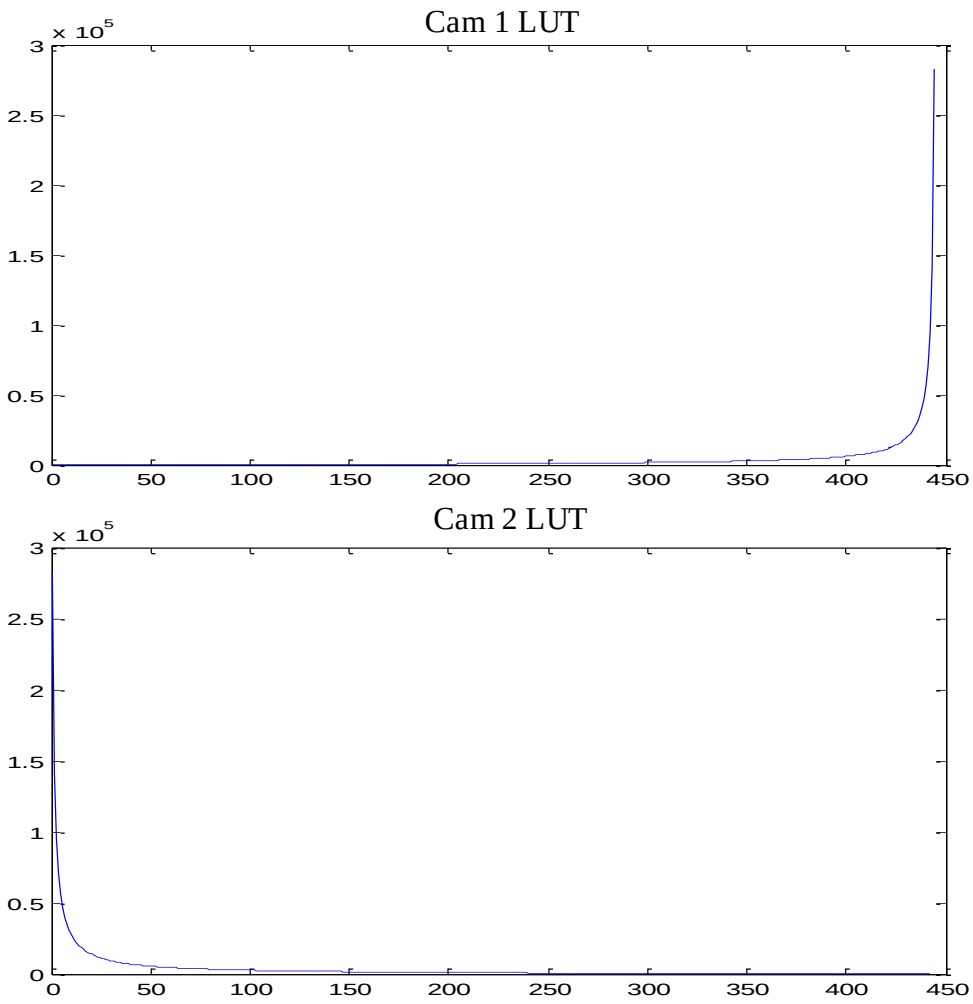


Figure 44 Look up Table Correspondence Value

We have assumed that the typical value of 450 pixels constitutes the valid range (range between the start pixel and the end pixel) in this implementation. In addition, both the start pixel and the end pixel are not used since the tangent value is either zero or infinite at these points. All tangent values are stored as integers to avoid floating point processing. The stored value is the actual tangent value multiplied by 1024, and when processing is complete the correct value is obtained by right shifting the answer by 10 bits, effectively dividing by 1024.

3.7 Normaliser Unit

The Normaliser Unit is one significant module inside the Processing Unit (PU) which is on a separate data path and pipelined with the edge and position processing. It is the unit normalizing the camera waveform into a range between zero and one. There are three reasons in the initial design phase to construct this unit as a system addition. First is noise cancellation: since the normalized waveform is obtained from the division of the current camera frame and averaged reference frame, the noise effects in the current frame will be reduced after the normalization. Second, the normalized result is within the range between zero and one; a simple threshold based comparing method could implement the edge detection efficiently which allows for potential future optimization. Last but not least, it is a necessary supplement to track and record touch history in order to indicate touch motion (touch-up or touch-down) by weighting the ratio of the normalized waveform, since the Edge Detection Unit (EDU) on the other data path is only able to detect a touch event without further in-depth monitoring during the event.

The block diagram of the Normaliser is illustrated below. The Normaliser is comprised of the Normaliser Operator, Level Detection Unit, Norm Register, Norm Comparator and Gesture Register.

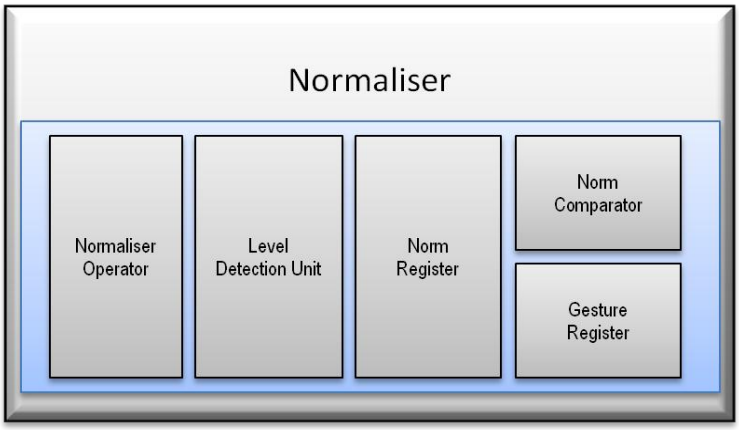


Figure 45 Normaliser Block Diagram

3.7.1 Normaliser Operator

The Normaliser Operator is used to calculate the ratio of the normalised waveform (the current camera pixel is divided by the reference camera pixel at the same position, and then subtracted from one).

3.7.2 Level Detection Logic

The Level Detection Logic is made of a group of comparators with parallel comparing value ranging from 10 to 90 (percentage) to find the match from the previous Normaliser operator.

3.7.3 Normaliser Register

Normaliser Register records ratio result for each pixel comparing at norm wr-enable clock.

3.7.4 Norm Comparator

Norm Comparator consistently compares each pixel ratio result through the whole frame and records the largest one which is the deepest touch on the screen.

3.7.5 Gesture Register

At the end of the frame, Gesture Register will shift the latest largest ratio result into the gesture buffer which keeps track of the latest five frames' touch history.

The abstracted circuit level of the Normaliser is illustrated below:

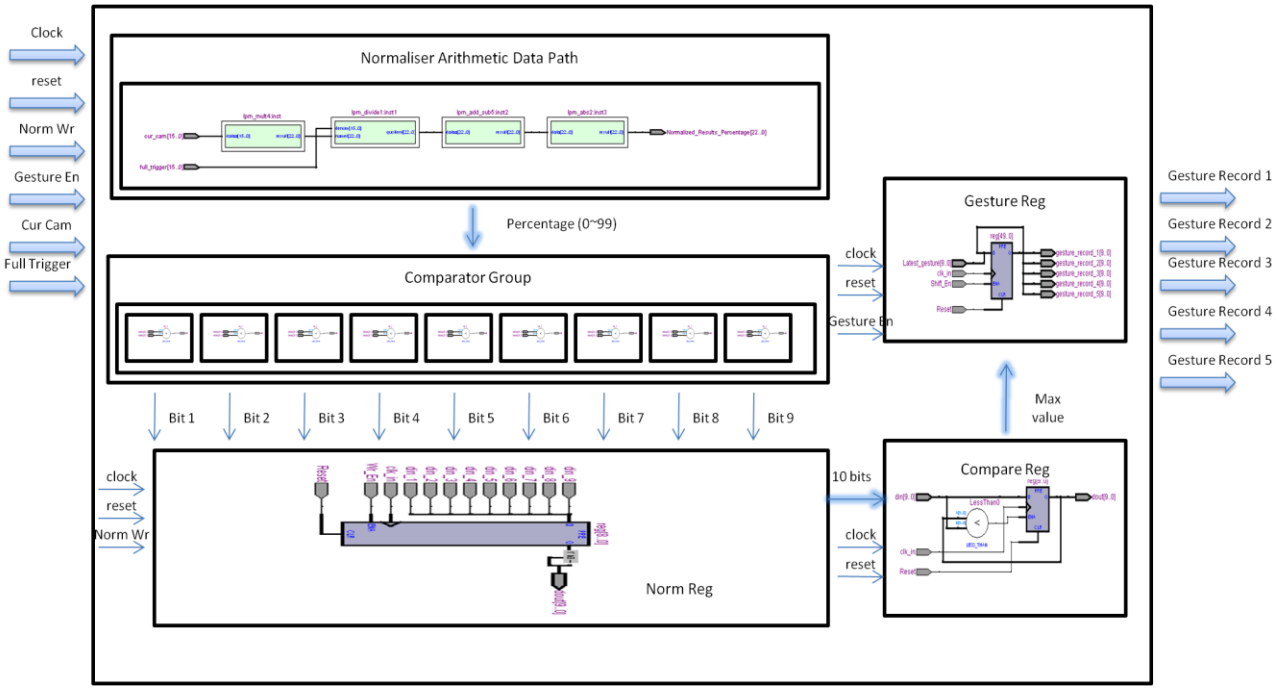


Figure 46 Normaliser Abstracted Circuit Structure

After a new camera pixel is available from the ADC module and the reference camera pixel is read from memory, the Normaliser Unit is activated with the Normaliser Operator calculating the ratio of the current pixel to the reference pixel, and the Level Detection Logic generating a 9-bit binary number indicating the percentage of the detected touch level. Then the Norm Register is enabled by the Master Controller to record the normalized result (percentage presented by the binary number) which is immediately fed into the Norm Comparator to weight and save the largest ratio value (which means deepest touch level in normaliser). After the whole frame has been analyzed, the deepest touch level binary result will be stored in the Gesture Register as the touch level status of the current frame and the oldest touch level result will be shifted out of the Gesture Register.

The Gesture Register is designed to record the most recent five touch level ratio results. The touch motion (touch-up and touch-down) can be easily evaluated by reviewing and comparing the history results with great potential to be extended to other new features. The Normaliser Unit consumes a total number of 5 processing cycles for the pipeline level detection logic. In the future, the structure can be optimized by a hardware resource design trade-off with processing time.

3.8 System Master Controller

The Master Controller is state machine based control logic which directs the system process flow and organizes all the operations of sub-modules including all the processing units' functioning and also the interfaces with the data acquisition system. The states flow in a sequential order with concurrent control signals generated to activate and deactivate other function units to perform the required operations. In the worst case, the Master Controller design consumes 22 system cycles to process each sample (pixel) so a total of 0.5 ms is consumed to finalize the touch position for the whole frame.

3.8.1 System Master Controller Control Flow

There are five major phases in the system control flow with 76 detail states: starting from the Idle phase, then the Trigger Configuration phase, Trigger Retrieving phase, to the Edge Processing phase and Position Processing phase. The control flow is illustrated below:

Simplified system control flow:

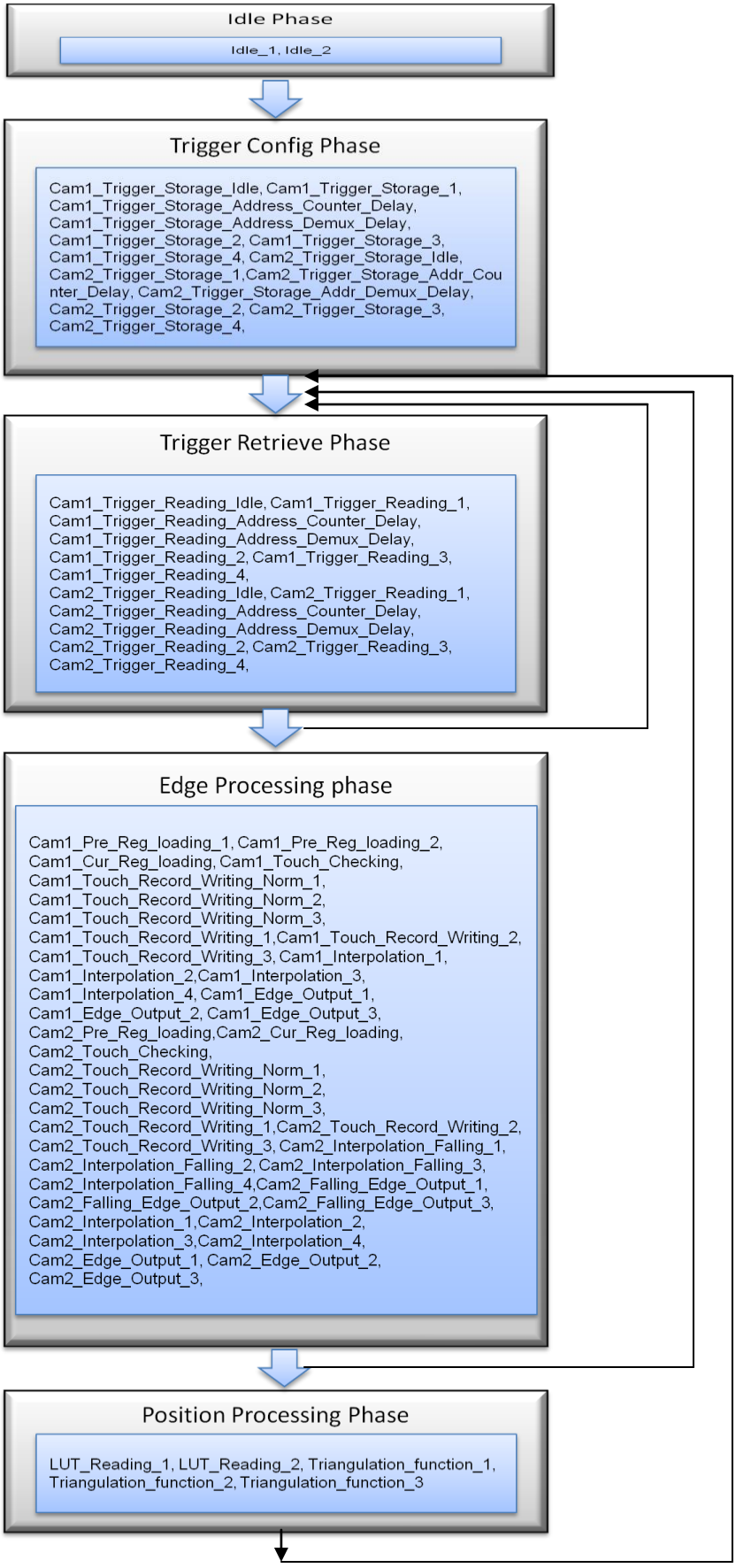


Table 3 Master Controller State Transition Table

Idle Phase:

There are two idle states in the idle phase, Idle_1 and Idle_2. An asynchronous pulse is generated from Idle_1 to Idle_2 to set certain units (Normaliser and EDU) to a particular status with all other control signals set to their default values.

Trigger Configuration Phase:

After the Idle Phase, the Master Controller will activate the MMU (Memory Management Unit) to store reference camera frames after being indicated as stabilization by the Acquisition Controller. It takes four states (from Cam1_Trigger_Storage_1 to Cam1_Trigger_Storage_4) to complete the BRAM reading process with propagation delay occurring in Address Generator and Counter module.

Trigger Retrieve Phase:

After the reference camera frames have been stored in memory in the Trigger Configuration Phase, the trigger level will be derived repeatedly from the reference camera frames through the customized trigger logic block before edge localization starts. At the end of the Trigger Retrieve Phase, all necessary preparations for edge and position processing is complete.

Edge Processing Phase:

Edge processing flow is a critical part of the whole system's operation. It guides both the ELU (Edge Localization Unit) and Normaliser Unit to indicate a touch event, calculate edge and sub-edge locations, and track touch motions (touch-down and touch-up). Once both camera frames are completely analyzed, system flow will be directed to the Position Processing Phase. Otherwise, it will flow back to the Trigger Retrieve Phase to start next sample edge detection.

Position Processing Phase:

Position processing is the final stage to calculate the x and y coordinates using the pre-calculated Look Up Table in ROM and enabling the pipelined position processing controller in the PLU (Position Processing Unit). When the final touch position is localized, the transmission engine will be activated and the system will flow back to the Trigger Retrieve Phase to initiate the next cycle detection and localization processing.

Chapter 4 -- Touch Screen System on Chip Testing Methods and Results

4.1 Overview

The testing of the hardware touch screen system has been executed in two parts with different testing focuses: the first part is the testing of the Data Acquisition System where different modules inside the acquisition system have been tested separately concentrating on acquisition speed, noise level and the proposed concurrent acquisition mechanism. The second part is the real time functionality proofing of the Processing Unit which is supported by a number of sub-module tests.

4.2 Data Acquisition System Testing Methods and Results

4.2.1 Timing Control Engine Testing Method and Results

Customized timing control logic has been designed in the Data Acquisition System which replaces the general purpose microcontroller to control both the image sensors and the illumination system to consistently obtain camera frames. The following testing photo shows the timing control logic using a 500 kHz data clock which takes about 1ms to complete the acquisition of one line-scan frame (528 pixels) from an image sensor.

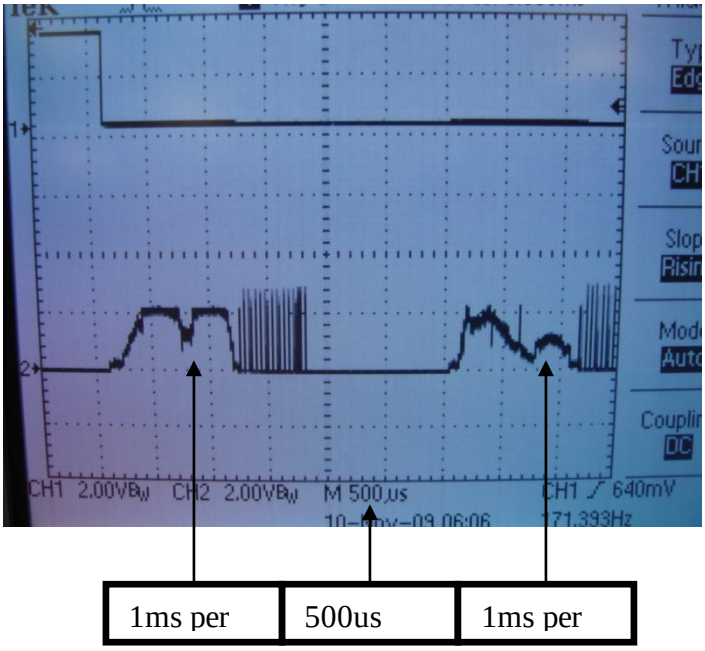


Figure 47 Acquisition Timing Control Testing Result

The following testing photos show the hardware timing control engine acquiring ambient frames, normal camera frames, camera frames when a touch occurs at one position and camera frames when a touch occurs at another position:

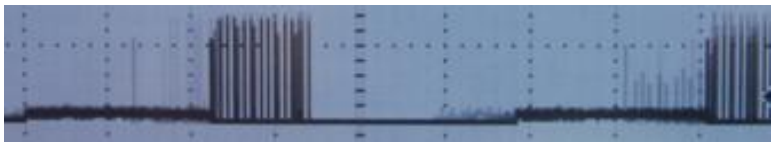


Figure 48 Hardware Timing Control Engine --- Ambient Frames

Figure 49 Hardware Timing Control Engine --- Normal Frames (No Touch)

Figure 50 Hardware Timing Control Engine --- Touch at One Position

Figure 51 Hardware Timing Control Engine --- Touch at another Position

The testing results show the customized hardware timing control engine is able to perform the same acquisition function as the general microcontroller based system. The response to the touch position (which blocks the reflection on camera frames) is represented clearly on the two camera frames.

The image sensor is designed to operate at about 1 MHz data clocking rate. The following testing photo shows the hardware timing control logic is capable of supporting faster acquisition speeds (1.8 MHz for example) which is a significant acceleration compared with the existing micro based acquisition (300 kHz).

Hardware Timing Control Engine running at 1.8MHz data clock rate:

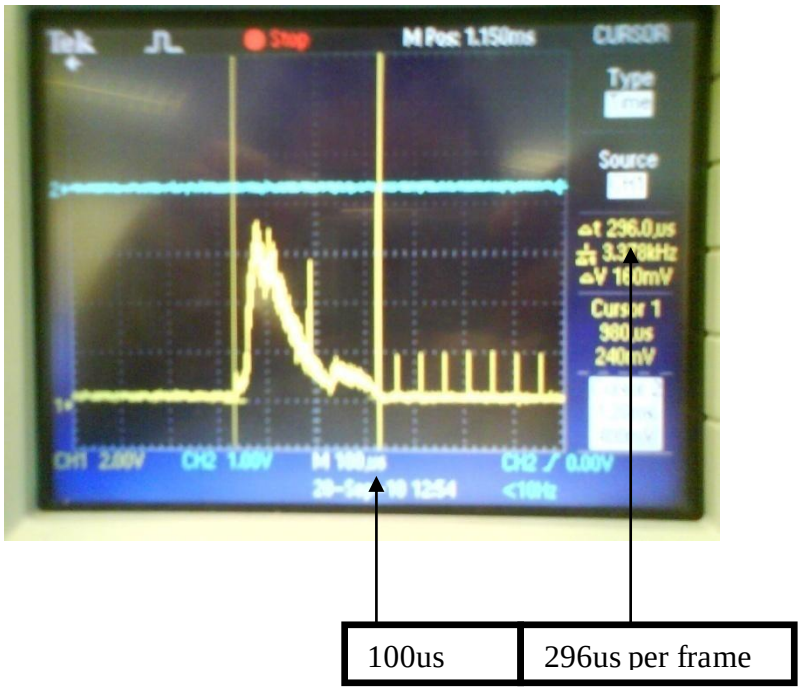


Figure 52 Figure 36 Acquisition Timing Control Testing Result 900K

The above camera scope is operating at a 1.8 MHz data clock rate which takes approximately 296 us total to transmit one complete frame (528 pixels).

4.2.2 ADC Testing Method and Results

The ADC module is tested separately before being integrated with the verified Timing Control Logic. A number of constant voltage inputs have been fed into the ADC module to test its conversion functionality and consistency. The following figure shows the digitized result based on 1.21 volts input from an adjustable power supply:

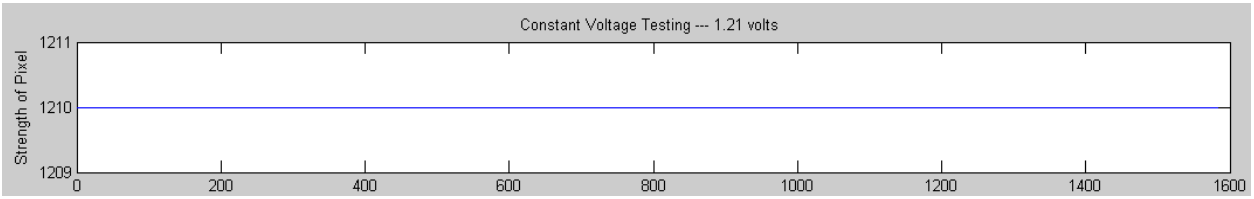


Figure 53 ADC Testing Results --- Constant Voltage Input

Then the verified ADC module was integrated with the timing controller. At this stage, the main structure of the hardware data acquisition system has been established with full capability to acquire touch information and digitize it into a digital format. The following testing photos show the conversion from analog pixel to digital values based on a range of testing situations:

Analog Camera Scope: Digitised Camera Scope:

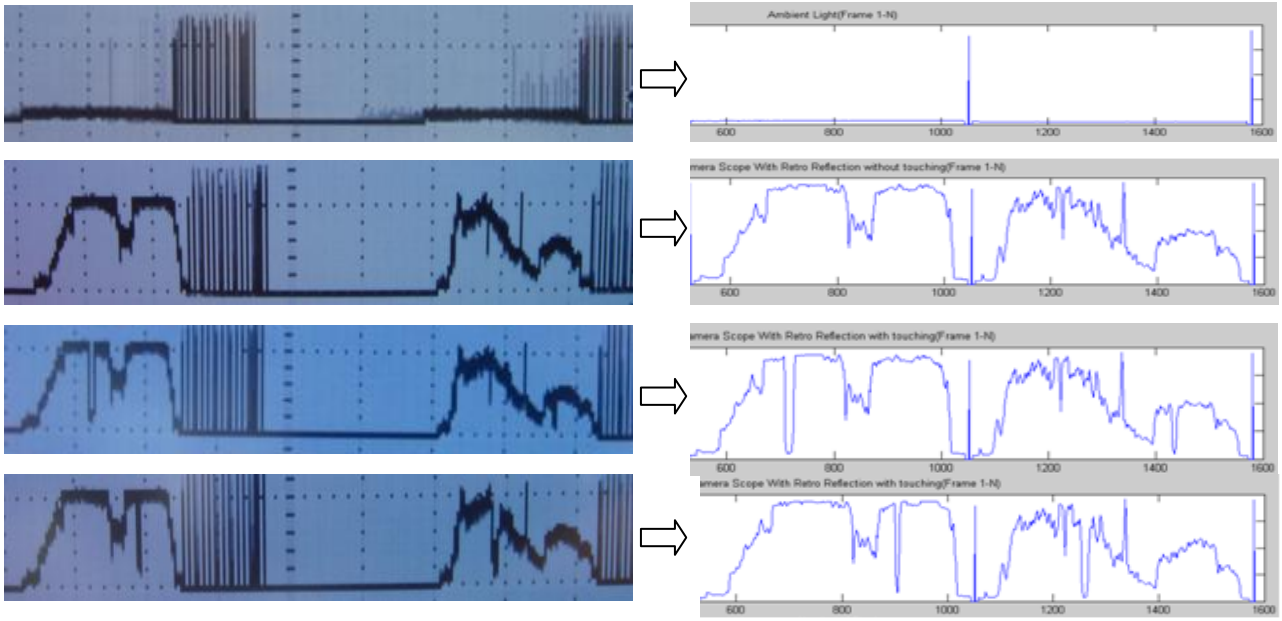


Figure 54 Data Acquisition System Digitization Result

The testing results show the hardware acquisition system (ADC module included) correctly digitizing ambient frames, normal frames and frames including a touch.

4.2.3 USB Testing Method and Results

During the the Data Acquisition System design and testing, a USB module is implemented in the system to consistently transmit digitized camera frames to the PC side monitor. By doing this, any real time touch events occurring on the hardware acquisition engine controlled touch screen are able to be displayed immediately, providing intuitive testing capability.

The USB module has been tested separately with an up counter continuously generating testing inputs to the USB module. The same incrementing numbers should be received on the PC side if the USB transmission is consistent and coherent. The corresponding testing results have demonstrated the correct functionality.

After that, the USB module was integrated into the Data Acquisition System demonstrating real time operation results. The following photo is the real time monitor displaying the hardware acquisition system operation results, on a custom interface written in C++:

Figure 55 PC Monitor Interface

The two camera frames are displayed on the real time interface through the acquisition and transmission of a complete hardware structure based solution.

4.2.4 Noise Analysis and Results

Low system noise level is one important design specification for the Data Acquisition System. The existing microcontroller based acquisition system has overall 1.5% noise variation which is mainly caused by two sources: the image sensors and the internal ADC (the ADC silicon design itself and interference from the nearby internal USB transmission). The proposed hardware Data Acquisition System has adopted a high performance external ADC chip with extra low drift and

a low noise reference voltage chip. Also, the USB module and the ADC module are separated so they cause minimum interference to each other. The system noise level has been analyzed in Matlab, running on the new hardware acquisition engine with various image sensor settings.

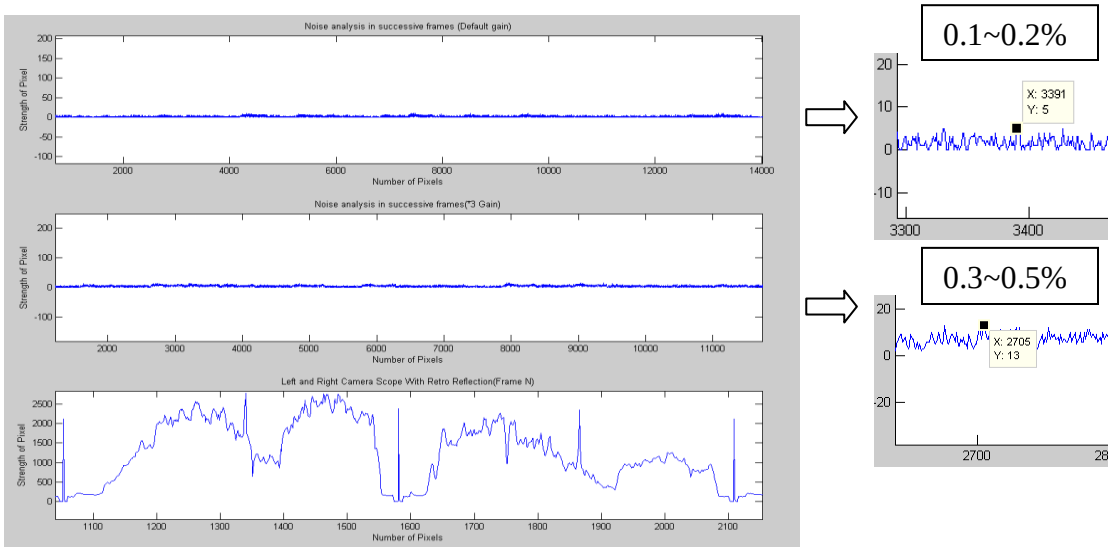


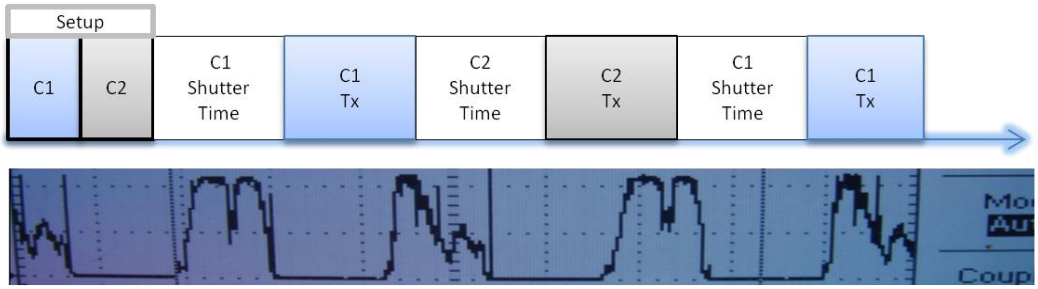
Figure 56 Data Acquisition System Noise Analysis Result

The top figure is the image sensor operating at default voltage with a typical noise level of 0.1% (5 out of 4096) and a worst case of 0.2 %. The middle figure is the image sensor running at a voltage gain of 3 with a typical noise level of 0.3% (13 out of 4096) and a worst case of 0.5%. The bottom figure shows the normal camera frames. The testing results demonstrate the new customized hardware acquisition engine has a significant improvement in reducing system noise.

4.2.5 Concurrent Acquisition Mechanism Testing Method and Results

The Concurrent Acquisition Mechanism is a more efficient acquisition control method applied in the proposed hardware acquisition engine with concurrent access to both shutter and operation control. The testing results of the new acquisition mechanism are presented below:

Sequential Mechanism:



Concurrent Acquisition Mechanism:

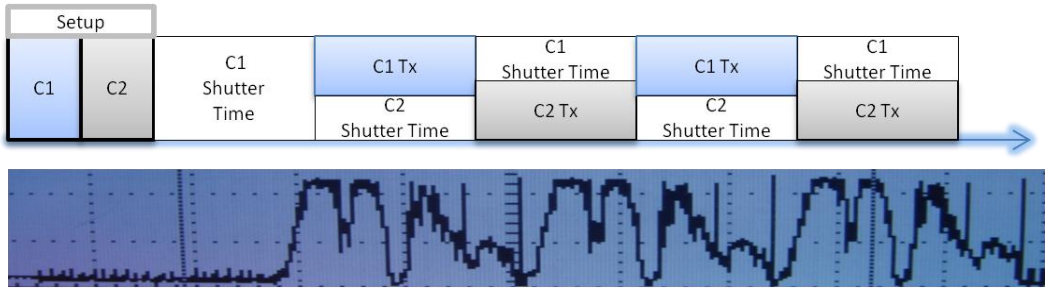


Figure 57 Concurrent Acquisition Mechanism Testing Result

The concurrent acquisition mechanism has an acquisition speed approximately twice as fast as the sequential acquisition mechanism. The new acquisition method is organized to ensure the consistency of transmission switching between the two cameras. Moreover, it has the convenience and potential to be applied to systems containing more than 2 cameras (say 4-24 cameras) to maximize the advantage in acceleration with main structure reuse.

4.3 Processing Unit Testing Methods and Results

The testing priority of the Processing Unit (PU) is to validate and demonstrate correct functionality without too much consideration of accuracy and efficiency. The Processing Unit is comprised of sub-modules to perform frame management, edge detection, position localization and touch motion tracking. All the sub-modules have been verified and tested separately.

4.3.1 Memory Management Unit Testing Method and Results

The Memory Management Unit is the storage organizer which directs digitized frame writing into the M4K memory block during the trigger configuration phase and reading from the memory block during the trigger retrieve phase. It is necessary to ensure the correct procedure and results in accessing memory. A separate testing circuit has been designed to write particular camera values at particular addresses with a LED display based on the MMU outputs to confirm the values by reading from the same memory addresses. The following structure is specifically built for MMU testing with an adjustable address and camera generator feeding testing inputs and a visible LED displaying reading results:

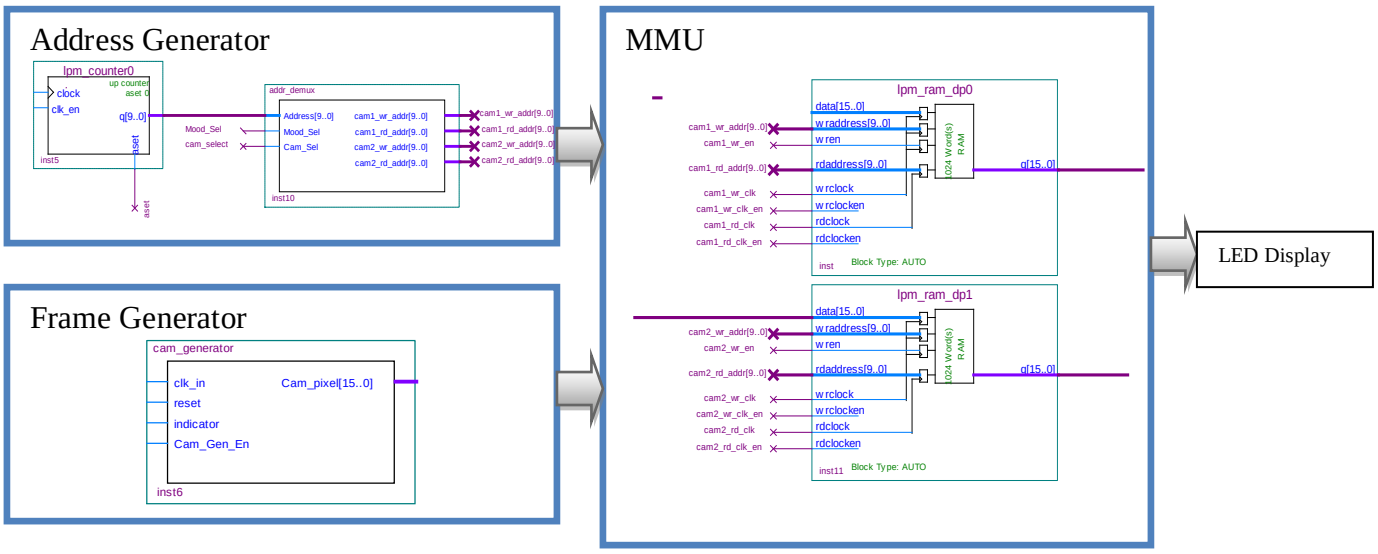


Figure 58 Memory Management Unit Testing Structure

A group of testing numbers has been applied to the MMU testing module, and the results displayed on the LED demonstrated the MMU is able to record reference input correctly, which is a fundamental verification step before running in real time.

4.3.2 System LUT Testing Method and Results

The test method for the System Look up Table is similar to the MMU module testing. Since both camera look up tables are pre-calculated and configured in ROM, we know the table value at each particular address. The LED display is also used in this part to confirm the table accessing. The LUT testing circuit is illustrated below:

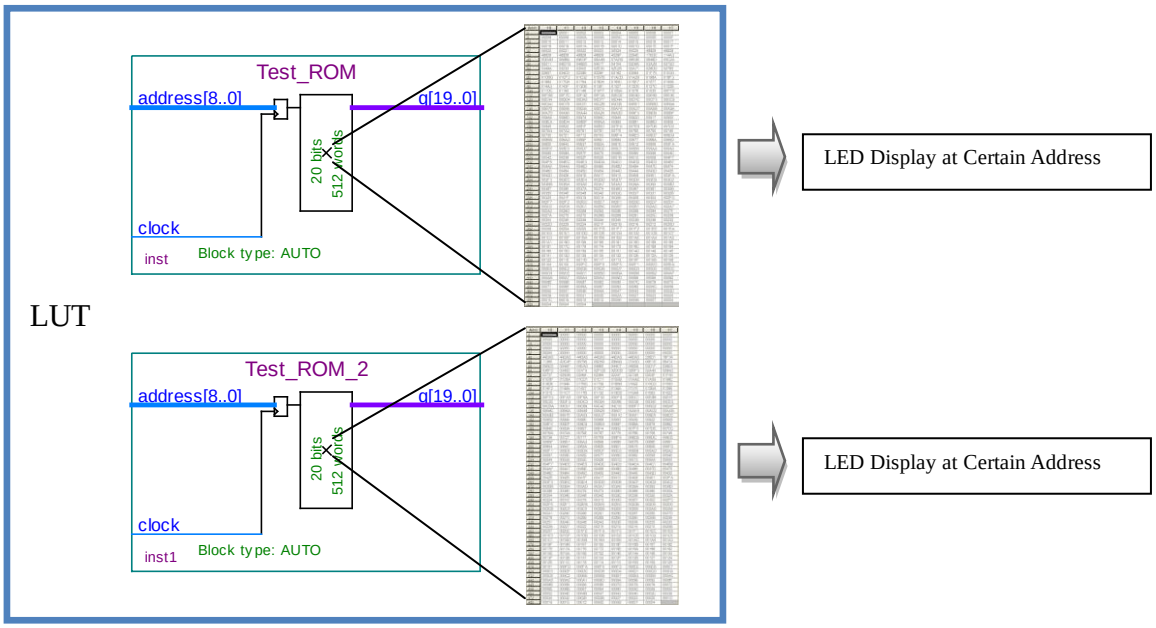


Figure 59 System Look up Table Testing Structure

A number of LUT addresses have been read from ROM and displayed correctly on the LED.

4.3.3 Edge Localization Unit Testing Method and Results

There are two types of testing for the Edge Localization Unit (ELU). One is a purpose built edge simulator generating different falling-rising edge based camera frames for proving the proposed logic is able to perform the edge detection algorithm and give correct results. The edge localization simulation structure is illustrated below with pre-defined rising and falling edges in the camera frame generator:

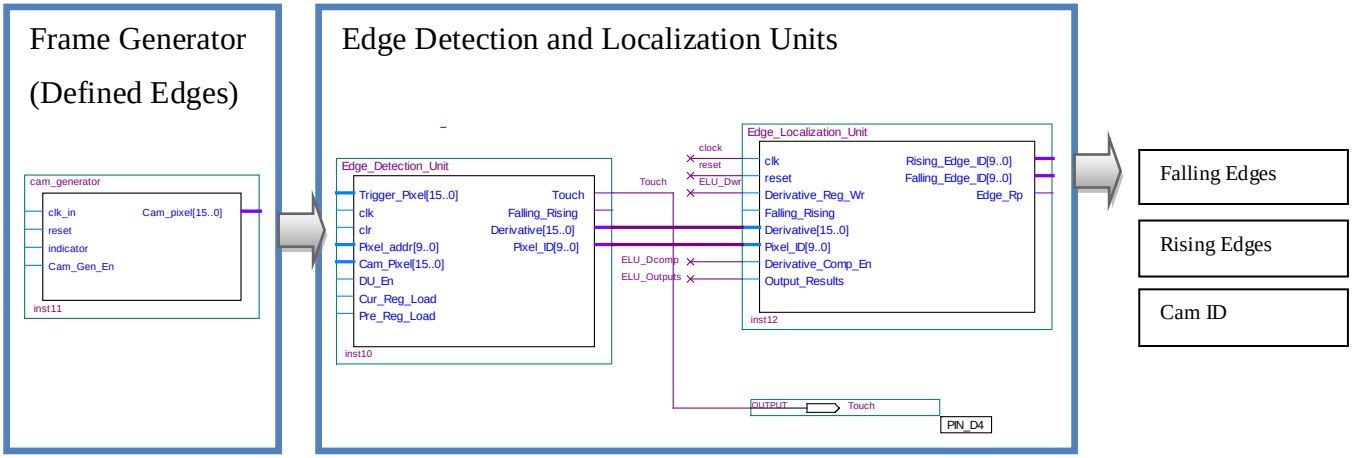
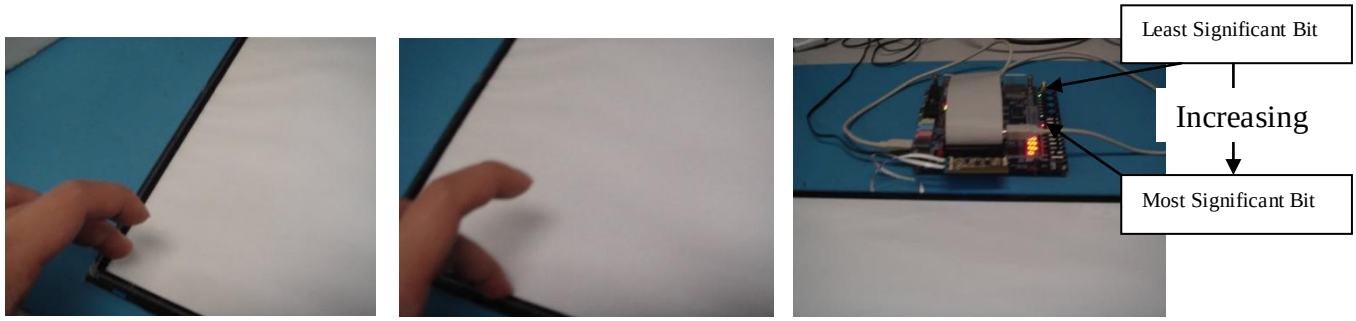


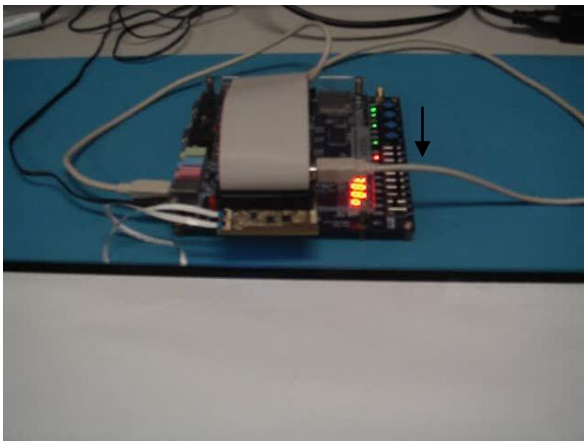
Figure 60 Edge Detection and Localization Units Testing Structure

Typical edge localization testing results has verified the required functionality.

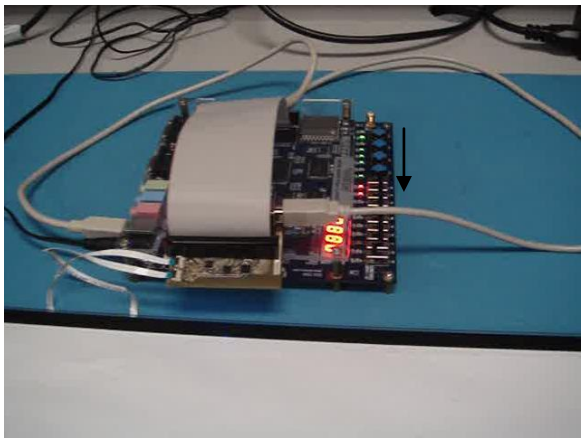
After the functionality of the ELU has been proved in simulation, it is integrated into the system to detect and localize real touch edges on the actual screen to validate the real time performance. This testing has been designed with the touch object blocking and moving from the beginning of the frame to the end of frame, as seen from one camera, and then moving back from the end to beginning. By doing this, both rising and falling edges of the detected object should be localized by the ELU as increasing (from frame beginning to end) and then decreasing (from frame end to beginning). The following are images are from video clips showing the whole real time testing process and results.

When an object (finger) moves from a low value edge position to a high value edge position, the ELU processes the edge changes as increasing:

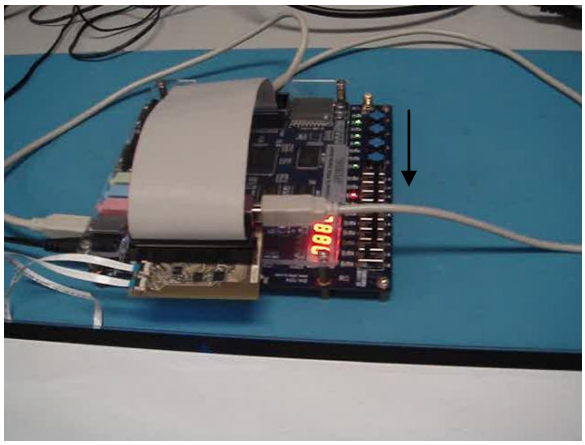




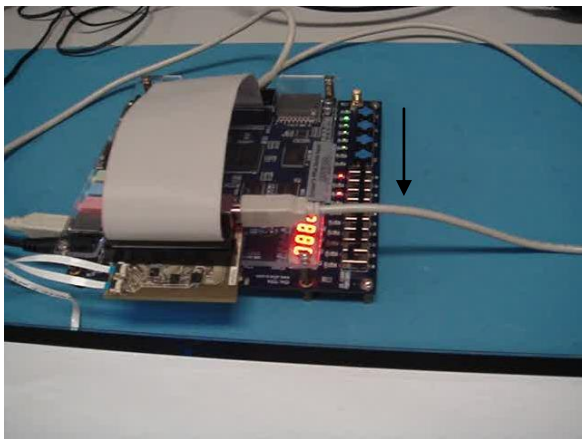
4



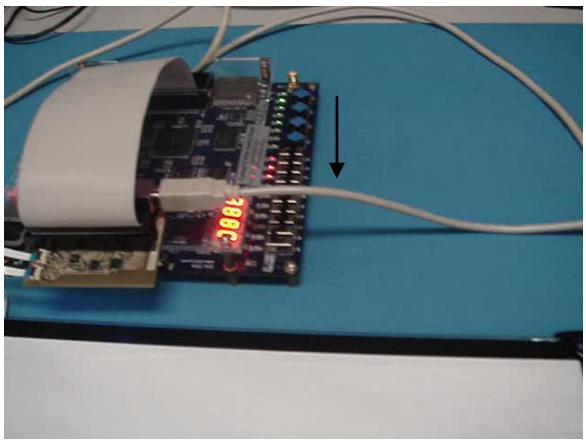
5



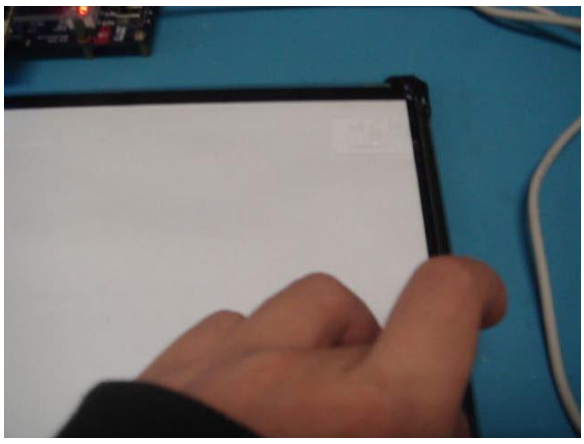
6



7



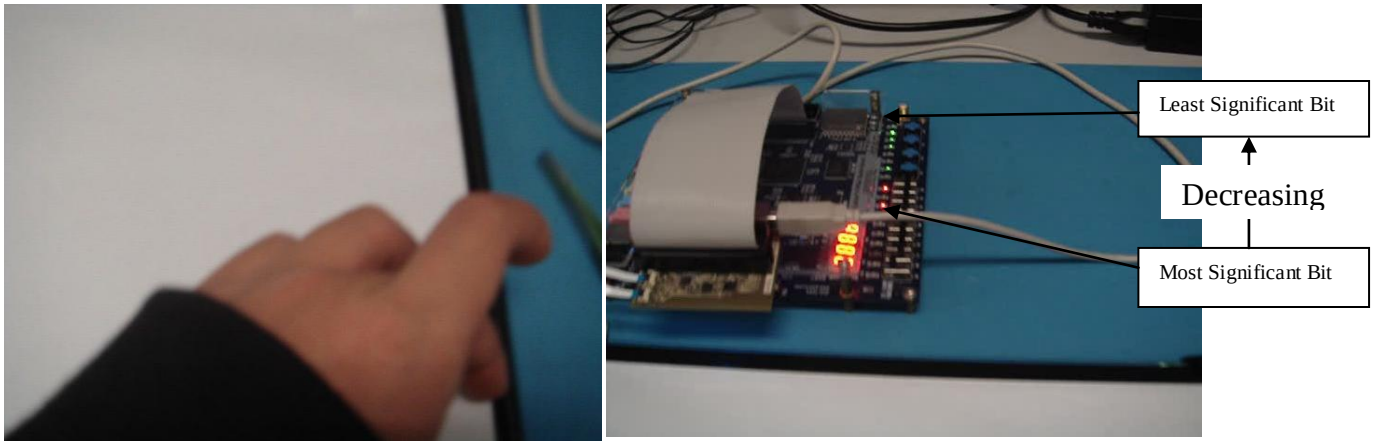
8

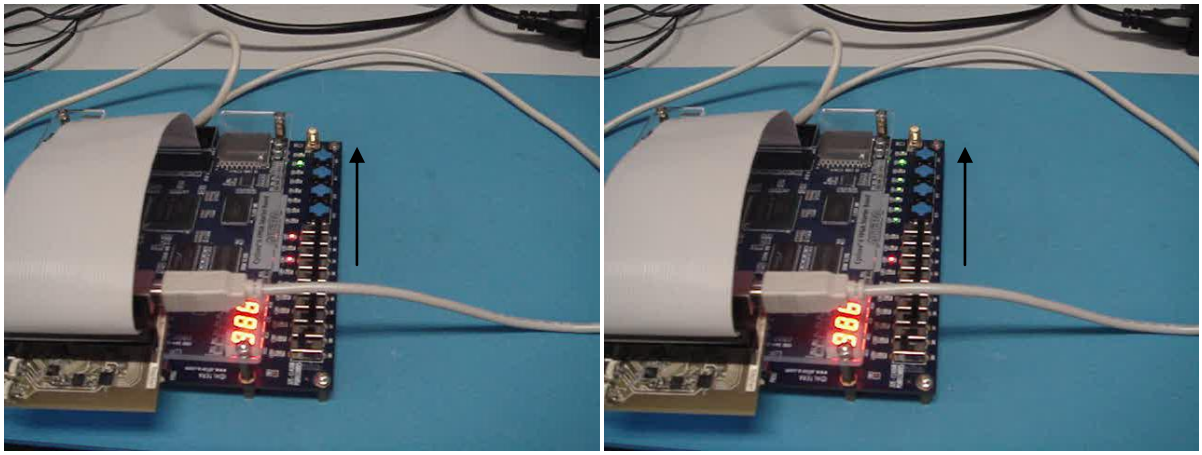


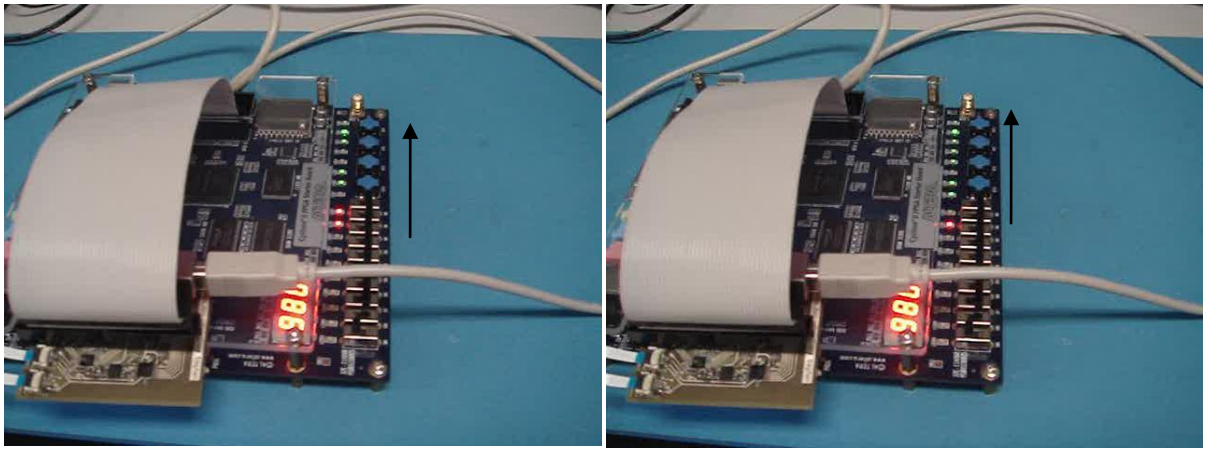
9

Figure 61 Edge Localization Unit Real Time Testing Result (Edge Up)

When an object (finger) reaches the far end (high value edge position) then the testing continues by moving the object back to the start point again, and the ELU processes the edge changes as decreasing as shown below:







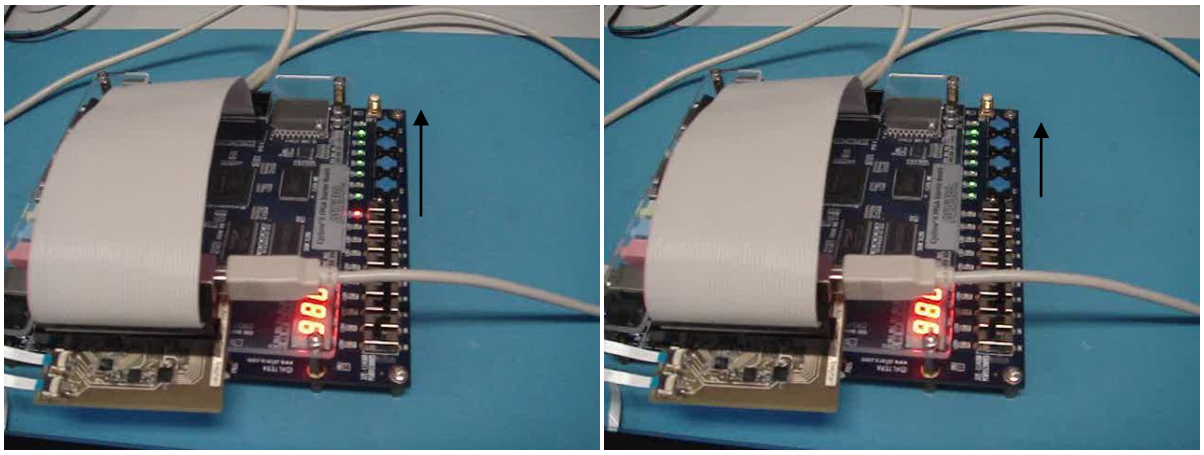


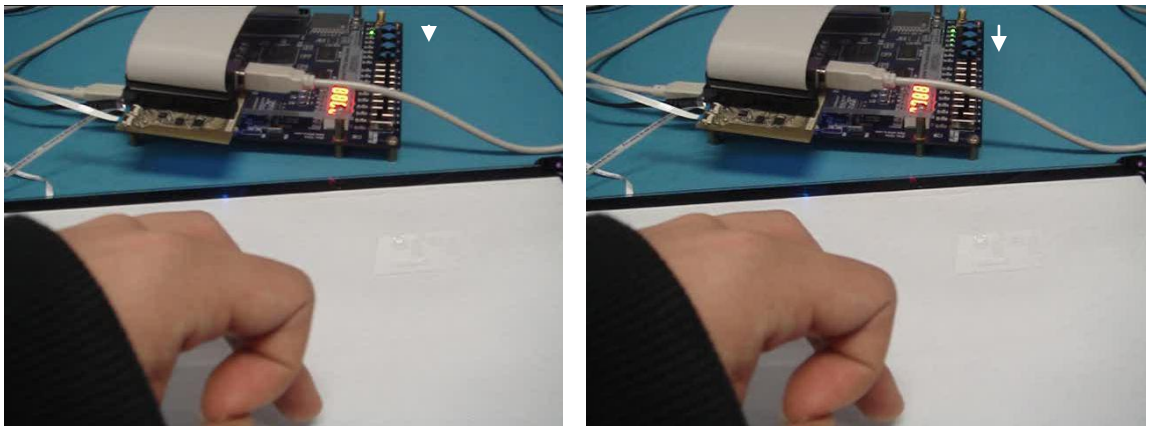
Figure 62 Figure 50 Edge Localization Unit Real Time Testing Result (Edge Down)

The real time testing results have been presented in a real time testing video (on the CD) which validated the integrated performance of the Edge localization Unit.

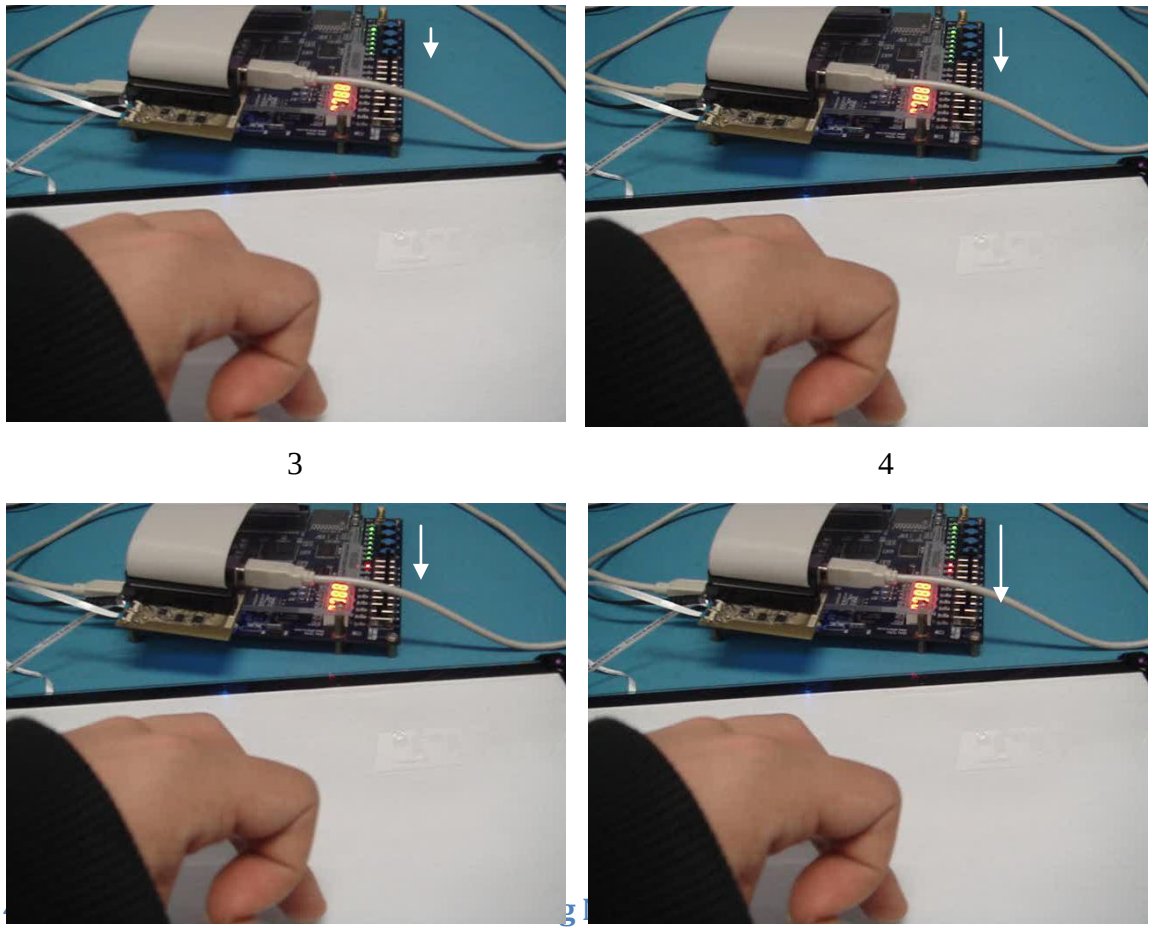
4.3.4 Normaliser Testing Method and Results

The Normaliser Unit is used to detect the touch level and then further track the touch motion (touch-up and touch-down). The LED is used to present the touch level ranging from no touch (1 led) to contacting the screen (9 leds).

Normaliser based motion detection is shown below (Touch-down tracking for this example):



1 2



3 4

5 6

Figure 63 Normaliser Testing Result (Touch Down)

The testing process of the Position Localization Unit (PLU) is performed in two stages as for the ELU testing: a Module Simulation Stage and Real Timing operation testing:

PLU Module Simulation:

An extra testing block has been designed for PLU simulation with the PLU function module and a testing controller which provides the PLU inputs (cam1 tangent, cam2 tangent and screen width) and control signals for the PLU module. The PLU processing results are displayed on the LED. The testing block structure is illustrated below:

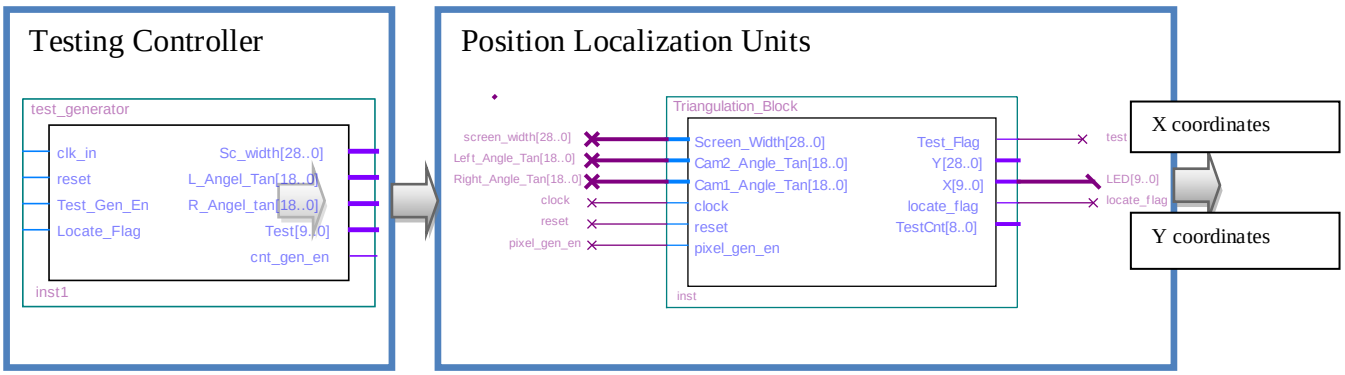


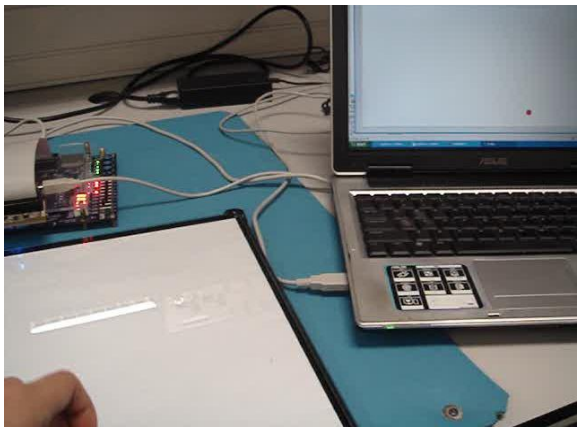
Figure 64 Position Localization Unit Testing Structure

After verification using the simulation, the Position Localization Unit was integrated into the final system, which should now consistently process x and y coordinates in real time. The final system testing is presented in section 4.4.

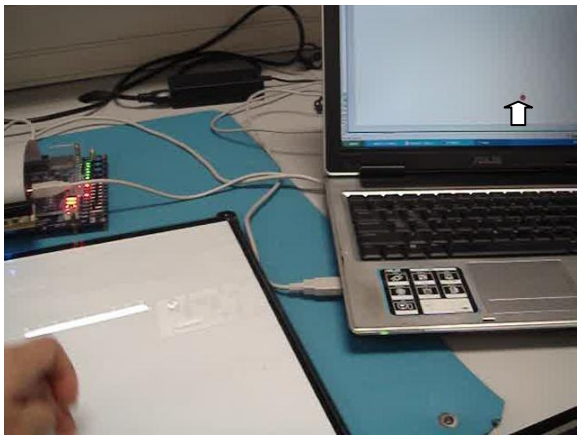
4.4 Complete Real-Time System Testing Results

A number of simple actions have been tested on the complete hardware structure based touch screen system, such as drawing a straight line, drawing a rectangle and cycles to validate and visualize the real time performance of the proposed hardware system:

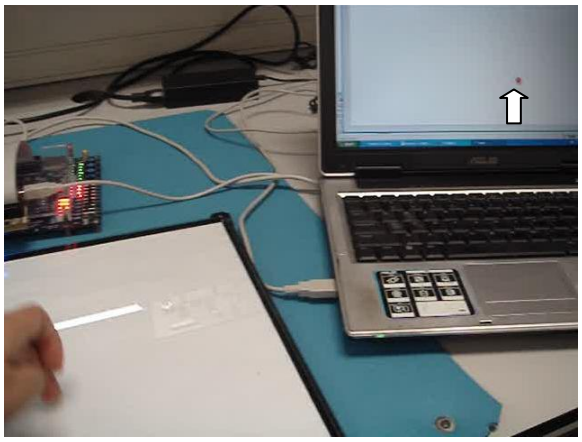
This column is drawing a straight line:



1

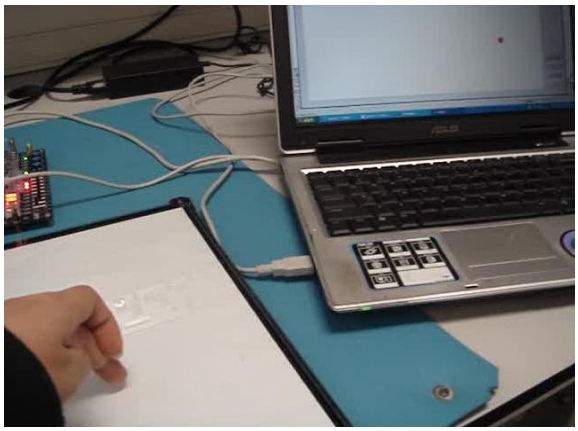


2

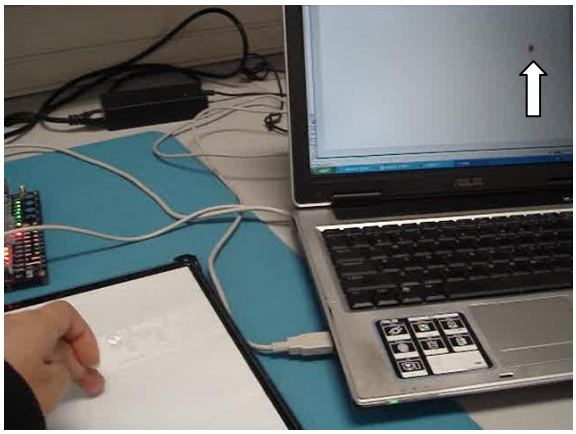


3

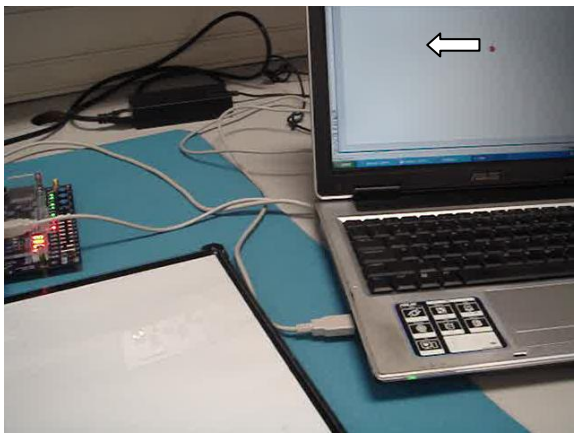
This column is drawing a rectangle:



1

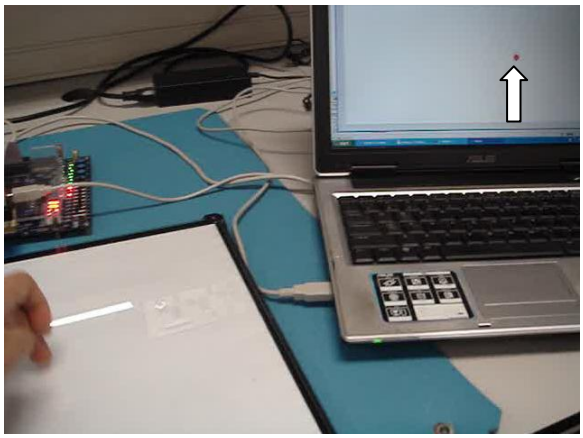


2

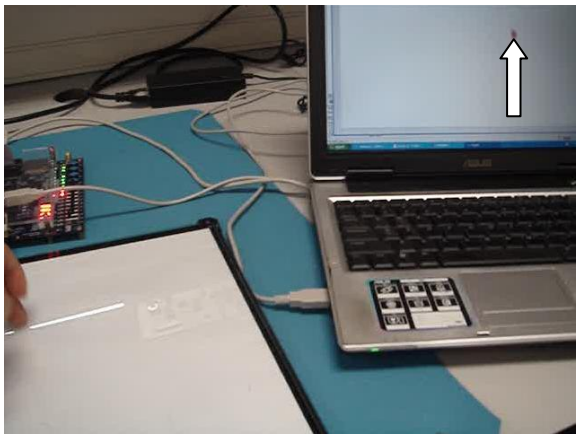


3

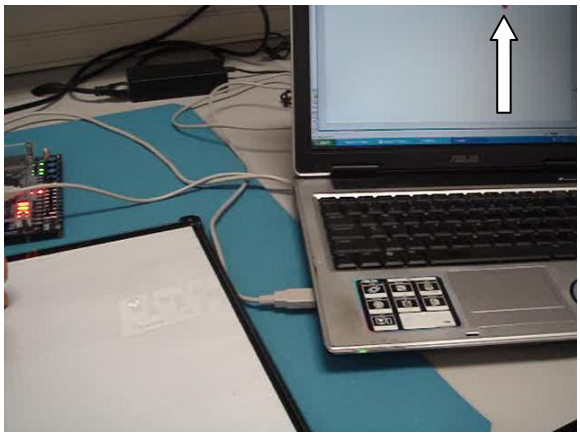
Drawing a straight line:



4

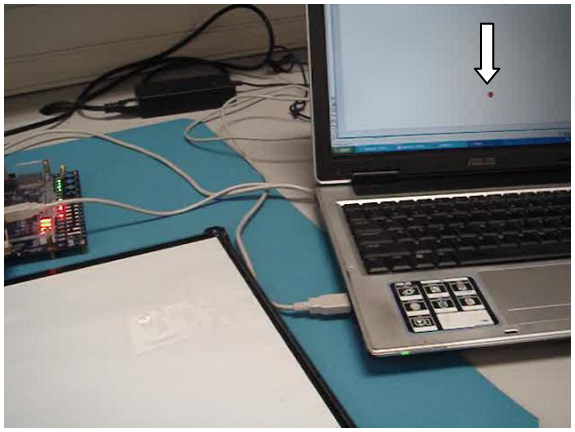


5

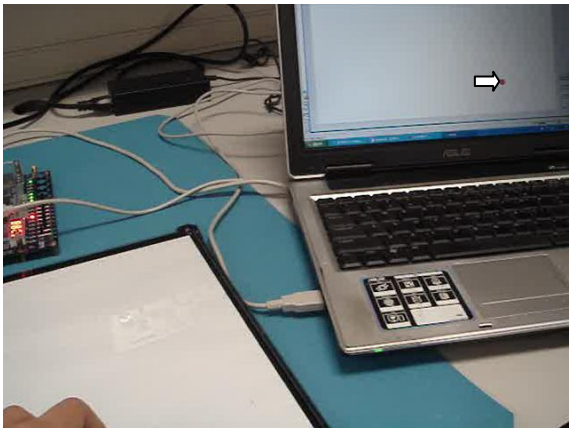


6

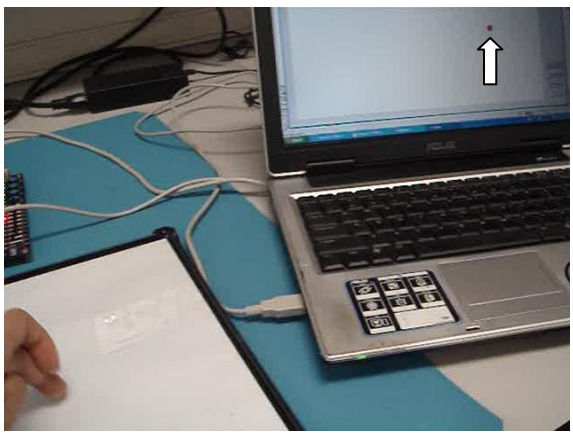
Drawing a rectangle:



4



5



6

Figure 65 Real Time Touch Screen System Testing Results (Drawing Straight Line, rectangle)

The real-time complete system testing has also been presented in a video (on the CD attached).

Chapter 5 Hardware Touch Screen System Optimization

5.1 System Optimization Overview

The original touch screen system has been developed based on a block based System-On-Chip design methodology because of the initial uncertainty of the system component level architecture. The disadvantage of block based design is the lack of consideration of system level structuring, and inefficiency on the sub-module level in terms of communication and resource sharing. After establishing a functionally-correct touch screen system, a significant number of optimization techniques have been applied to the proposed system on the architectural level, behavioral level (algorithm level), functional level (register transfer level) and physical structural level (gates and switch) from an overall system perspective.

A unique topology has been designed before implementing system optimization, which targets chip area consumption by reusing logic resources to a significant extent, and also maintains (or even improves) the overall system throughput performance while using minimum resources. In detail, a re-organization on the architectural level has been performed to accelerate the system operation speed with negligible resource addition; key algorithms have been investigated and evaluated on the behavioral level to select the solution using least resources; function sharing and other reduction techniques have been applied on the register transfer level to increase the resource utilization efficiency and an application specific chip floor planning process has been executed to improve timing and propagation capability on the physical gates level.

More details are presented in the following paragraphs regarding overall system optimization results.

5.2 Top-Level Architecture Optimization

The top level hardware system structure is mainly comprised of a data acquisition part and a processing part. In the original design, the data acquisition and processing units operate sequentially with two frames (1024 pixels) acquired, digitized and stored completely into memory with a time cost of 1ms at a 1 MHz acquisition clock rate. Then, the processing unit has to retrieve pixel information from memory again and detects, functions and analyses pixel by pixel with a total time cost of 0.5 ms (22 processing cycles per pixel, 1024 pixels at 50 MHz processing clock).

The original system flow is illustrated below with a latency of 1.5 ms:

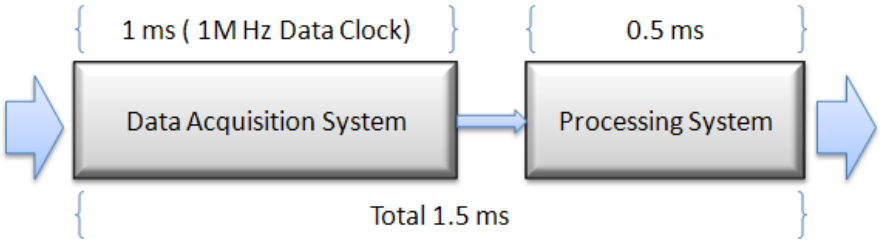


Figure 66 Original Top Level System Flow

It is apparent that, instead of storing complete frames while halting the processing unit, both units can be arranged in a pipeline with no extra function cost and the addition of minimum multiplexing logic. The processing unit starts operation as soon as receiving the first pixel from the data acquisition engine, and completes the single pixel processing before the next pixel is available, since the data acquisition cycle is longer than the processing cycle. After the reorganization, the system critical path has been successfully reduced to the data acquisition cycle, with a significant latency improvement of 33%.

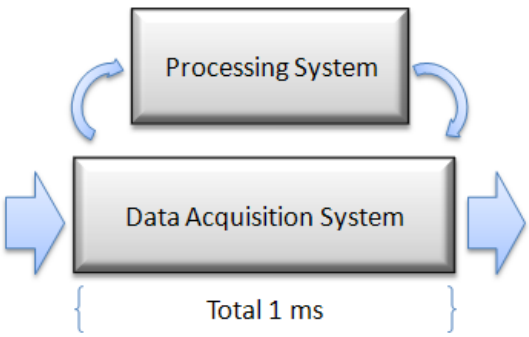


Figure 67 Optimized Top Level System Flow

5.3 Behavioral and Functional Level Optimization

5.3.1 Algorithm Evaluation (Alteration)

The highest level of abstraction of a digital design is known as the behavioral level which refers to the algorithm describing the behavior of a function using abstract constructs. The optimization and alteration on the algorithm level will significantly contribute to the efficiency of overall resource consumption. During the development of this touch screen system, edge detection has been one of the fundamental elements for object tracking, motion analysis and other requirements. Two different algorithms have been designed, constructed, implemented and tested to identify the most suitable solution under the system requirements. A number of tradeoffs and compromises have been made inside each algorithm in terms of speed and area to achieve a further optimization improvement.

5.3.1.1 Gradient Based Edge Detection Algorithm

Gradient based edge detection is the traditional image processing method which we have investigated and implemented in chapter 3.

5.3.1.2 Dynamic Linear Approximation Sub-Pixel Detection Algorithm

The proposed touch screen system is built based on line scan cameras with a one dimensional image, therefore the simplified 2D image processing oriented Gradient Edge Detection is not likely to be the most efficient solution. The simplified Gradient Edge Detection aforementioned is only able to provide one-pixel resolution which is not capable of meeting higher system requirements in a real world environment. An application specific hardware edge detection unit is required from both a resource consumption and a system requirement point of view.

The linear approximation sub-pixel detection method was originally proposed in Hussmann and Ho's [40] system, as an approach using two linear functions to locate the edge sub-pixel position. The value of the sub-pixel is determined by the intercept point of two equations, and principle is illustrated below:

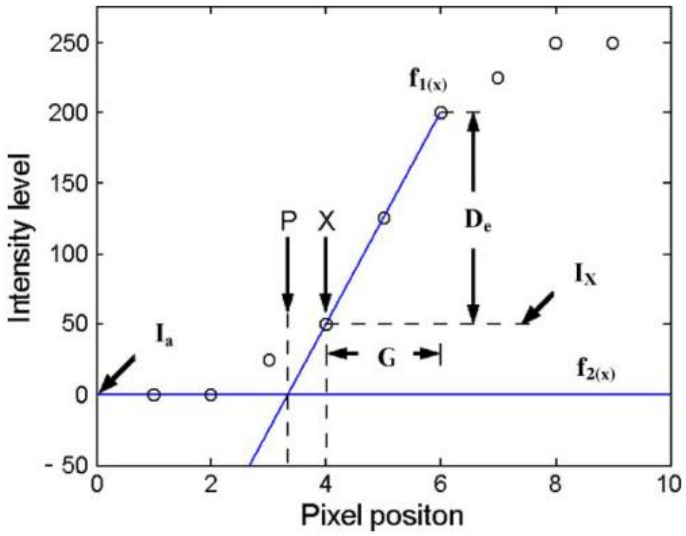


Figure 68 Linear approximation sub-pixel detection of Hussmann and Ho's system

From Hussmann and Ho's [40] description, taking the rising edge as an example, the first linear function is derived from the maximum positive gradient which is described as:

$$f_{1(x)} = D_{e_{\text{Max}}} x + (I_X - D_{e_{\text{Max}}} X), \qquad (1)$$

X is the pixel position at the maximum positive gradient where ***I***x is the corresponding intensity value. The other linear function is an arbitrary horizontal function with intensity value Ia:

$$f_{2(x)} = I_a, \qquad (2)$$

By combining equations (1) and (2), sub-pixel intersection position P can be calculated as:

$$P = \frac{I_a - I_X}{D_{e_{\text{Max}}}} + X.$$

Inside this algorithm, the arbitrary horizontal intensity Ia is preset through a look up table, which has the disadvantage of not adapting to real time ambient light intensity variations. Also, the first linear function is based on the gradient edge detection method we investigated and implemented previously, and which is not a very efficient solution in terms of hardware resources. Therefore, a more efficient edge detection method is required to minimize system resources.

The Dynamic Linear Approximation Sub-Pixel Detection Algorithm is presented in this section. This is derived from Hussmann and Ho's [40] work, with two significant modifications customized for the proposed touch screen system. The look up table based static light intensity Ia has been replaced by a dynamic updated trigger level which is calculated from the real time operation environment. Also, the resource-consuming whole frame based gradient sub-pixel detection is simplified to level triggered adjacent region based sub-pixel detection. This method is explained and illustrated below, based on a real camera scope of the touch screen system:

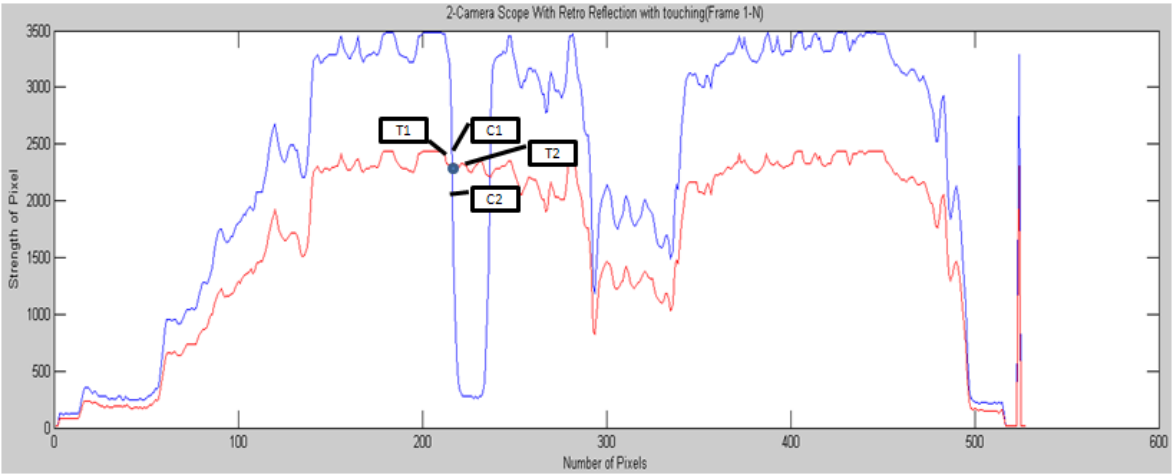


Figure 69 Dynamic Linear Approximation Sub-Pixel Detection Based on Real Camera Scope

The blue line is the full camera scope with a touch occurring near pixel 220. The red line is the trigger level which is 75% of the averaged consistent full camera scope. C1 is the camera pixel value just before the intercept point (where the blue and red lines intersect) while C2 is the pixel value just after the intercept. T1 is the trigger value just before the intercept point and T2 is the trigger value just after the trigger point.

In this method, the value of the sub-pixel at the intercept point is calculated from two modified linear equations. One is the linear function based on pixels C1 and C2 on the blue line:

$$f_{1(X)} = C_2 + (C_1 - C_2)(P_{id} - x) \qquad (1)$$

P_{id} is the id number of the pixel after the intercept point, x is sub-pixel value.

The second linear function is based on pixels T1 and T2 on the red (trigger) line:

$$f_{2(X)} = T_2 + (T_1 - T_2)(P_{id} - x) \qquad (2)$$

By combining equations (1) and (2), the sub-pixel position Pn can be calculated:

$$P_n = \frac{C_2 - T_2}{Mcam - Mtrig} + Pid$$

Mcam = C2 - C1 is the magnitude change of the two adjacent pixels in the camera image, while Mtrig = T2 -T1 is the magnitude difference of adjacent trigger pixel values. A new hardware detection unit is designed based on this optimized method which is illustrated below:

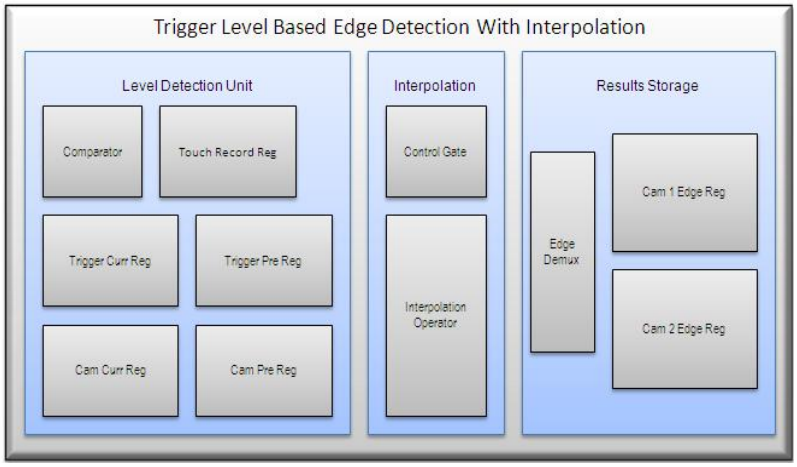


Figure 70 Trigger Level Based Edge Localization Block Diagram

The Trigger Level Based Edge Detection Unit is made up of the Level Detection Unit which has a comparator and register combination to detect and record a touch, the Interpolation Unit which calculates the sub-pixel positions and Results Storage which stores the edge pixel and sub-pixel positions for both cameras. The abstracted circuit level is illustrated below:

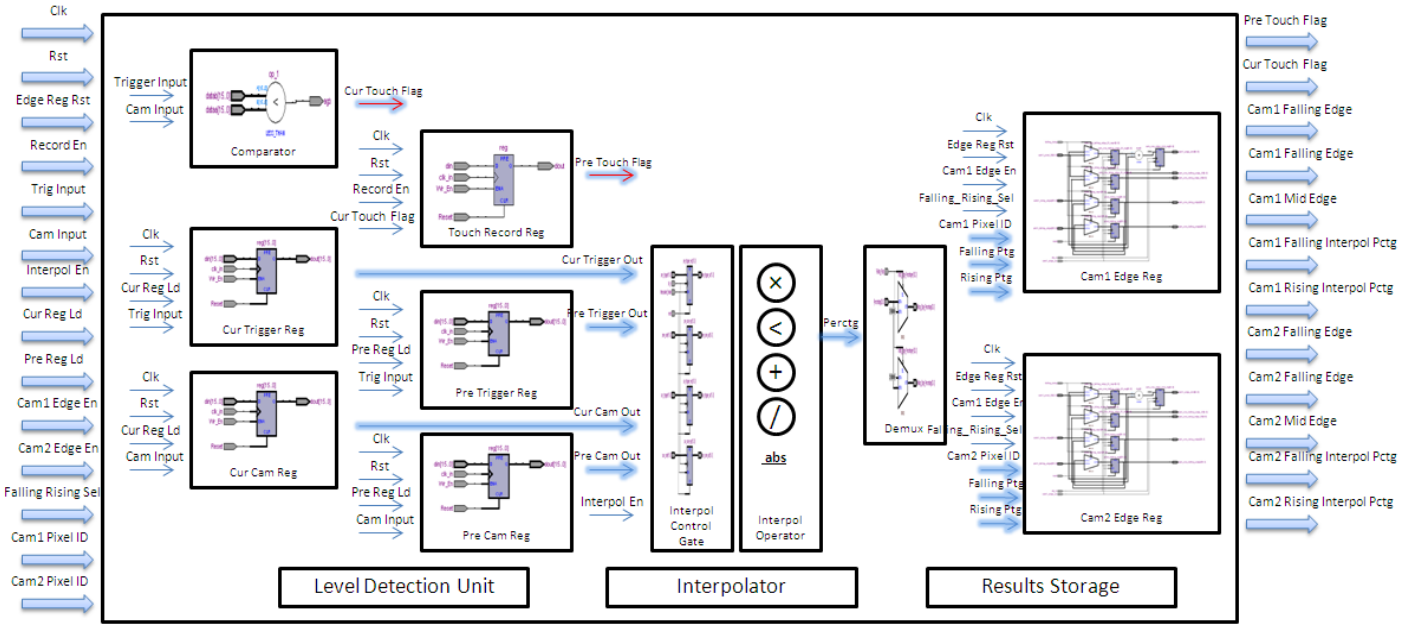


Figure 71 Trigger Level Based Edge Localization Abstracted Circuit Structure

When both the camera pixels and the trigger level values are available from the previous memory module, the previous camera pixel and trigger value will be fetched into the pre-register pair (Pre-Trigger-Reg and Pre-Cam-Reg) , and then the new camera and trigger values will be stored in the current register pair (Cur-Trigger-Reg and Cur-Cam-Reg) one cycle later. After register fetching is complete, the master controller will analyze the touch status based on the Cur-Touch-Flag from the Touch Comparator and Pre-Touch-Flag from Touch-Record-Reg (which has been initialized as no touch).

If both flags are high (active low), there is no touch in the current pixel, the current no touch status will be recorded in Touch-Record-Reg and the Edge Detection Unit will stay inactive until the next camera pixel is available. If one of Cur-Touch-Flag or Pre-Touch-Flag is low (Falling edge detected when Pre-Touch-Flag is low and Cur-Touch-Flag is high; Rising edge detected when Pre-Touch-Flag is high and Cur-Touch-Flag is low), a touch event is detected and the interpolator will be activated by the master controller. Four outputs (current camera and trigger values, and previous camera and trigger values) from the Level Detection Unit will be processed by the interpolation unit to produce more precise sub-pixel edge results. All pixel edges and sub-pixel edges are demultiplexed explicitly into rising edge, rising sub-pixel edge, falling edge, and falling sub-pixel edge and stored independently in the camera-one and camera-two registers. The last possibility, both Cur Touch Flag and Pre Touch Flag are low (active low), means the current pixel is neither falling edge nor rising edge but within the touch region. The current status will be updated in Touch-Record-Reg, with both the Interpolator and Results Storage Unit deactivated since the current pixel is not on the edge.

This Trigger Level Based Edge Detection takes a maximum of seven clock cycles for processing and storing pixel and sub-pixel results when a touch occurs, and a maximum of four clock cycles when updating the touch status when no touch is detected.

The following is the pipelined data flow of the optimized Edge Detection and Localization Unit with the Normaliser Unit:

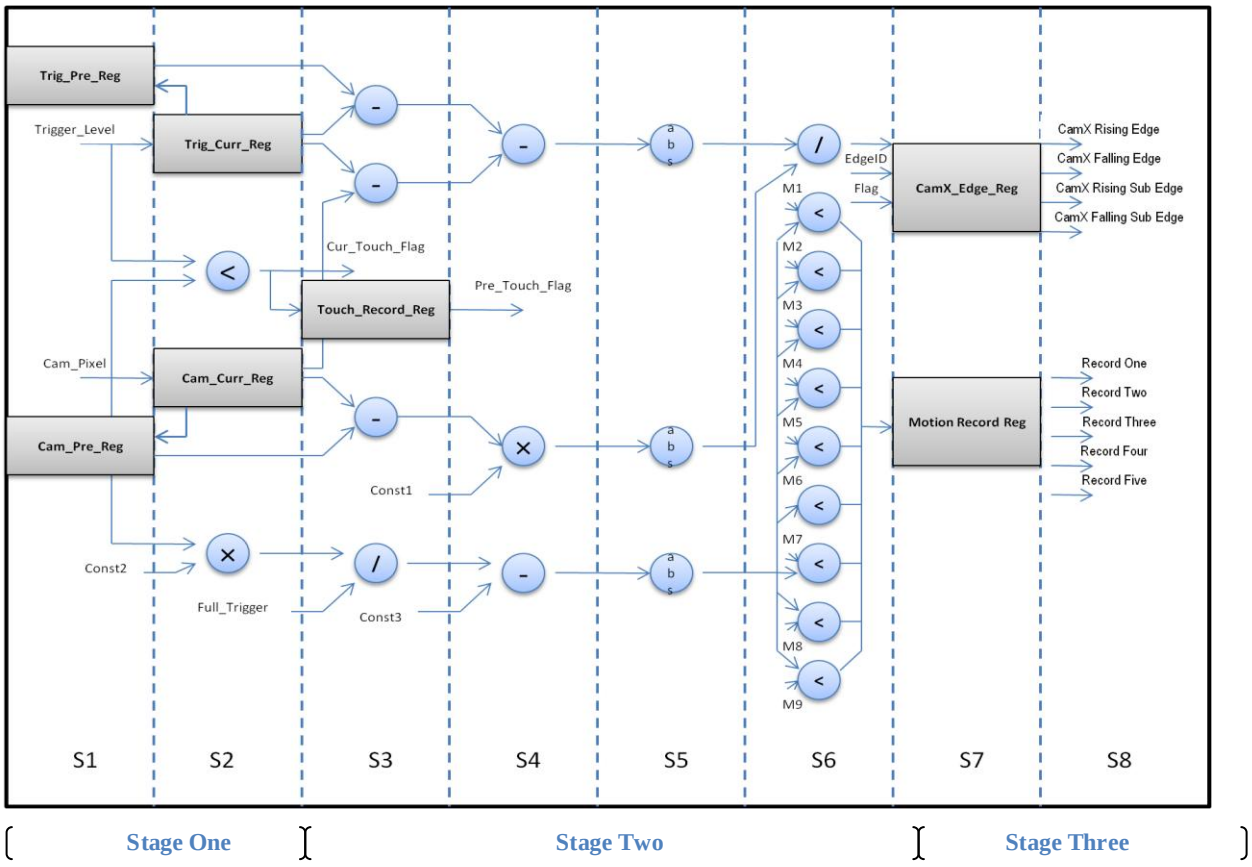


Figure 72 Pipelined Data Flow of Edge Detection and Localization Unit with Nomaliser

There are three major stages in the illustrated pipeline data path (edge detection, localization function and normalize function) with concurrent operations executed in each stage. For the edge detection and localization units, both the trigger registers and camera registers are updated at stage one as well as the touch event indication (comparison between trigger level and current camera pixel value). Once the four register values are loaded and the required flags are generated to the Master Controller, a set of concurrent arithmetic operations are performed to obtain the more accurate interpolated sub-edge results. For the normaliser unit, a group of sequential and concurrent operators have been developed to indicate touch depth ranging between 0 and 100 percent. In last stage, the edge register is used to store the processed sub-edge value with the corresponding edge value on both the rising and falling detection; and the normaliser record register is used to organize the five latest touch depth histories for further touch motion analysis.

5.3.2 Memory Resource Reduction

Memory storage is an important constituent in the touch screen system where the reference camera frames have been stored with extra logic for deriving the trigger level. Memory resource reduction will have a significant impact on chip resource efficiency, since all storage is in the on-chip M4K memory block in the proposed system design.

In the original memory unit planning, a capacity of two frames (one frame per camera) was required for basic processing purposes. Each frame consisted of a full range of 528 pixels including active pixels and dummy pixels. Two 1024 word M4K blocks (16 bits per word) were consumed to hold two full range frames with a total cost of 32768 memory bits. During the optimization, from the system point of view, it was found to be unnecessary to store the full range of pixels since only 512 active pixels are used in processing. Thus, the Data Acquisition System has been modified to acquire active pixels only. By doing this, two 1024 word blocks have been optimized to two 512 word blocks with no effect on system processing. The reduction of memory storage is illustrated below:

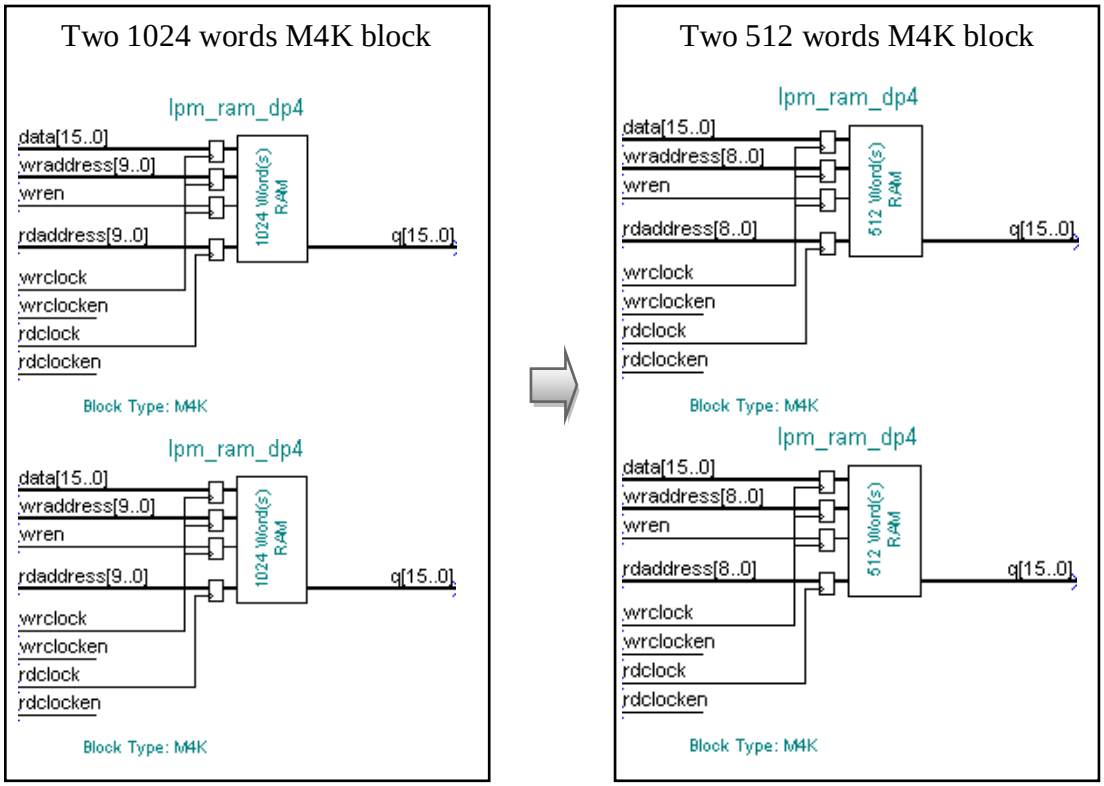


Figure 73 Memory Resource Optimization Illustration

5.3.3 Pipeline Rolling and Resource Sharing

Pipeline rolling and resource sharing are two popular techniques usually applied to digital system optimization. Pipeline rolling is defined as the opposite operation of unrolling a loop to create concurrent structures [62] . After reviewing the touch screen system design, some modules with a pipeline structure were able to be re-constructed as repeated loops also meeting the timing constraint. By doing this, replicating computational structures are reduced and the overall system performance is not affected.

Resource sharing is another common method in system resource reduction, in which a single functional block is accessed by several system operations [63, 64]. Resource sharing adds additional logic levels to multiplex the inputs to implement more than one function. Therefore it is a more complicated optimization technique which has high requirements on system control directing and switching. Resource sharing has been applied to a great content in the touch screen system optimization, with a significant modification and re-construction on both system level and sub-module level. More details are explained in the following paragraphs.

5.3.3.1 Data Acquisition Control Engine Resource Sharing

The Data Acquisition Control Engine is the key unit of the whole data acquisition system, which generates control signals to image sensors, the illumination sub-system and the ADC interface. It consumes a large amount of the hardware resources used by the data acquisition system, which in total uses 445 logic elements. The optimization of the acquisition controller is extremely difficult because the whole engine is in the system critical path and it is unlikely to trade speed into space. At the same time, the acquisition engine deals with low level signals with handshakes between external physical components in an accurately timed sequence which increases the complexity of applying any optimization techniques without interfering with its behaviour.

The following is the original acquisition controller block diagram:

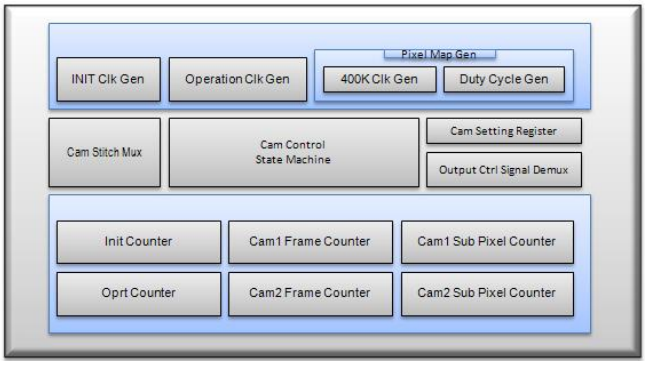


Figure 74 Data Acquisition Engine Block Diagram without Optimization

Reviewing the design of the original acquisition engine, it consists of three major parts: clock generators, the timing control state controller and the counter group. Advanced and different optimization techniques are customized for each part separately. For the clock generator region, instead of isolating it as local logic only utilized by the acquisition engine, a more efficient partitioning has been made to relocate the clock generators out of the acquisition engine into the system level so they can be shared by all other modules that need different clock inputs. For the timing control state controller, the original state transitions need to stay the same since they control fixed communication sequences with other components, but minor modifications have been made in the state controller counter for the initialization and operation modes. The obvious advantage of sub-module resource sharing is represented in the counter group optimization, where several counters used to specifically perform certain functions have been reduced to one counter which is shared by different requirements in different time slices directed by control logic. The new block diagram of the data acquisition control engine is illustrated below:

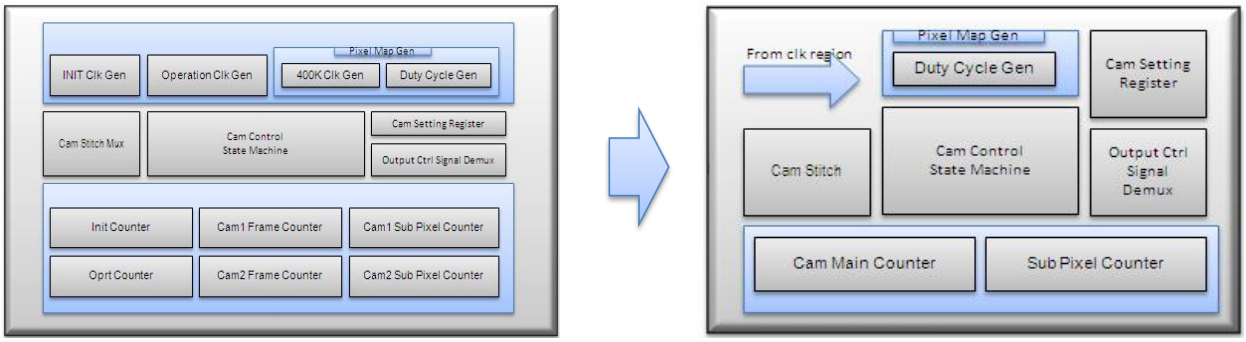


Figure 75 Optimized Data Acquisition Engine Block Diagram

The following is the abstracted circuit level of the acquisition controller:

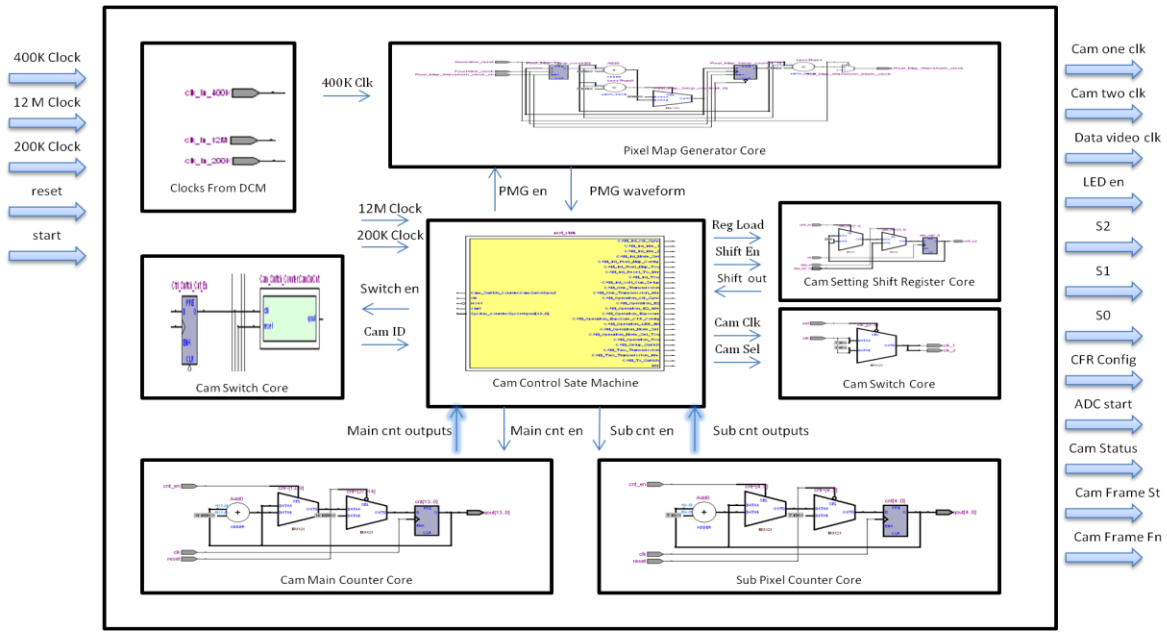


Figure 76 Optimized Acquisition Engine Abstracted Circuit Structure

All the clock generators at different frequencies are relocated to the Digital Clock Management (DCM) region, apart from the Pixel Map Generator which is unique to the Acquisition Engine and used to configure the pixel map for image sensors during initialization. Cam Switch Core is applied to direct control flow inside the state controller, for either camera, in both the initialization and operation modes, where the outputs of the controller will be re-multiplexed into either cam one or cam two depending on the control states. Cam Setting Shift Register is the component shifting reconfigurable voltage and current settings into the image sensor at the request of the state controller. Cam Control State Machine remains unchanged with three major stages: initialization cycle, exposure cycle and operation cycle which are made of a total of 25 states starting from the Cam Init Setup state back to the Idle state in a maximum of 2ms.
Only two counters are required in the engine after optimization: Cam Main Counter which is shared by both the initialization and operation cycles for timing control and frame counts (reduced from Init Counter, Operation Counter, Cam1 and Cam2 Frame Counter in the old design); Sub Pixel Counter (optimised from Cam1 and Cam2 Sub Pixel Counters) which is activated inside the frame counter to accumulate a number of subpixels for both cameras, and that cannot be combined with Main Counter because of a time collision.

The same low level advanced sub-component resource sharing idea has been applied to the ADC Interface inside the Data Acquisition System to further reduce logic consumption.

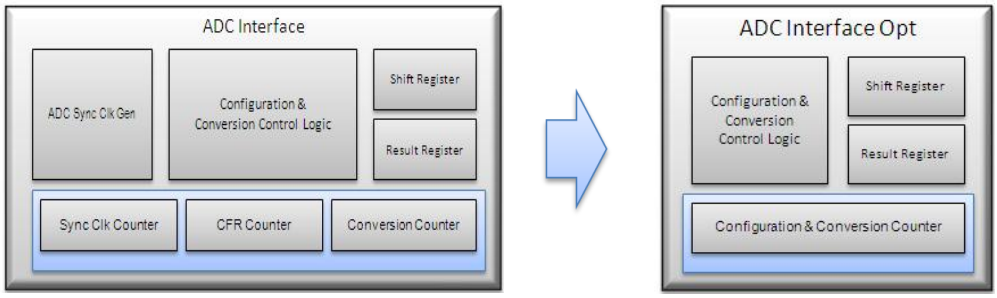


Figure 77 Optimized ADC Interface Block Diagram

Using the same principle, Sync Clock Generator has been moved to the DCM region which generates different clocks at a number of frequencies shared by all system modules. Synchronize Clock Counter is discarded after the removal of Sync Clock Generator. Configuration Counter and Conversion Counter are activated in different modes which from a timing sequence point of view are one after the other. A combined counter has been introduced which is reused by both configuration and conversion operations.

The sub components resource sharing interface structure is illustrated below:

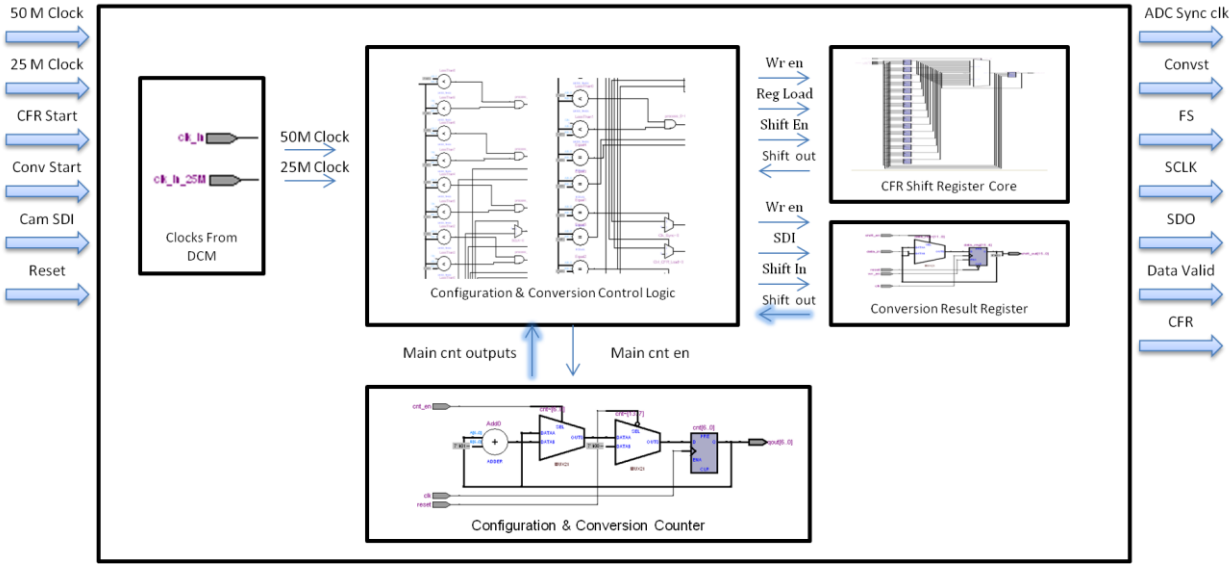


Figure 78 Optimized ADC Interface Abstracted Circuit Structure

All major sub components retain the same functionalities where Control Logic has the same timing sequence, CFR shift register configures with the same response and Result Register stores the same conversion results. One concern in this sub-component resource sharing structure is the clearing of combined counters between different operations to avoid interference.

5.3.3.2 ELU and PLU Resource Sharing (Arithmetic Resource Sharing)

The Edge Localization Unit (ELU) and Position Localization Unit (PLU) are two fundamental units for edge detection, localization and position triangulation which are also computationally expensive in terms of arithmetic operations. A total of 1211 Logic Elements are consumed by these two units which comprise more than 50% of overall system resources (990 LE in EDU and 221 LE in PLU). Therefore, it is critical and compulsory to review and optimize both units to improve the system efficiency.

Since both units have arithmetic operators made of adder, subtracter, comparator, multiplier, divider and abs (absolute value), it is considered that one common arithmetic unit is more optimal than separate arithmetic processing from a system point of view. Thus, a new general purpose ALU is constructed which covers both ELU and PLU's arithmetic operations:

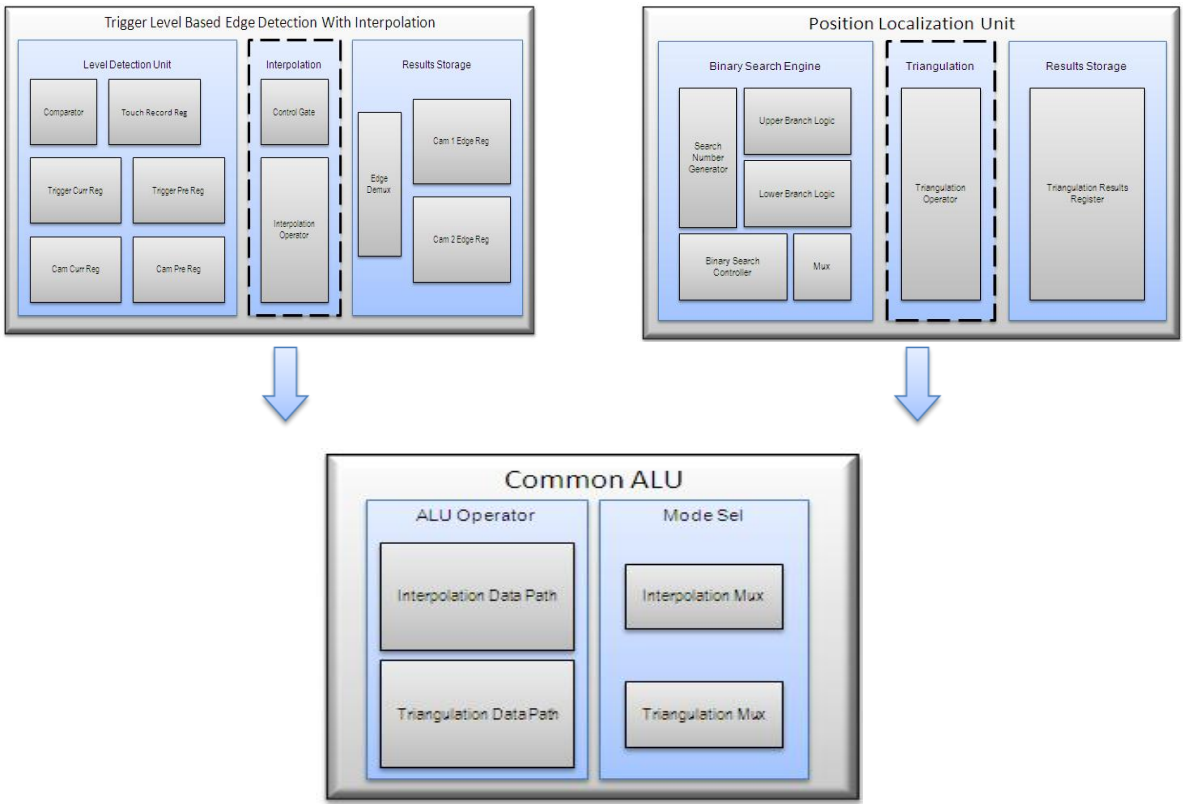


Figure 79 Arithmetic Resource Sharing Common ALU

The common ALU has been constructed with separate data paths for interpolation and triangulation which are selected by multiplexers. It is directed by the master controller for switching to the correct mode with the interpolated sub-edge or triangulated position fed back to the Edge Localization Unit or Position Localization Unit.

After the relocation of separate arithmetic operations into one common ALU, further focus has been placed on high-rank arithmetic operators such as divide, square root, and multiply which are both cycle-consuming and computationally expensive. In the ALU resource review, it was found that more than half the hardware resources are utilized by the Divide Operator (650 LE) inside the interpolation path which is not frequently used and not in the critical path.

Following is an overview of other arithmetic operator resource consumption:

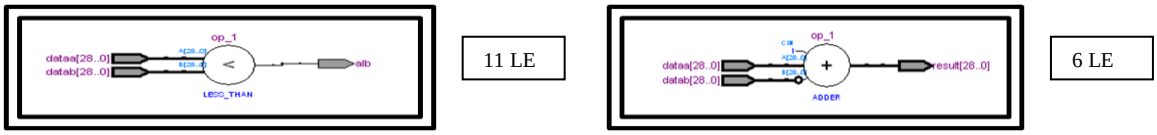


Figure 80 Comparator Resource Consumption and Adder Resource Consumption

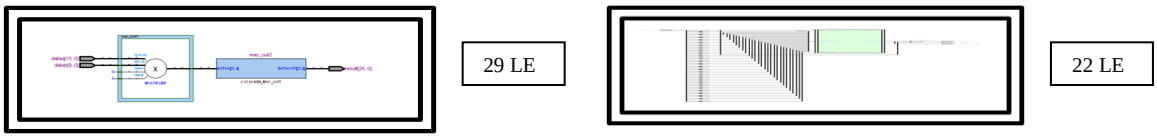


Figure 81 Multiplier Resource Consumption and Absolute value operator Consumption

5.3.4 Iterative Approach and Time-Division-Multiplexing (TDM)

From 5.3.3.2, an alternative approach is further required to replace the Divider while performing the same functionality. An iterative optimization method is inspired from Hussmann and Ho's system[40], which transformed a signed divider into an Up/Down Counter and Minimum Detector based iterative structure at the expense of latency. The result from the interpolator ranges from 0 to 1000 which represents an accuracy of one thousandth of the interpolation. In [40]'s iterative structure, in the worst case it will take exactly the maximum counter counts to complete the iterative process (in our case by applying the same structure, the worst case would be 1000 system cycles to obtain the interpolated result). The maximum latency allowed by the system for interpolation is 50 cycles which means the iterative approach mentioned above requires further modification and improvement to achieve timing closure with low resource cost.

An original Iterative Approach with a Timing-Division-Multiplexing structure is presented here to perform interpolation with a maximum 12 processing cycles that completely meets the timing closure, and a minimal additional cost of one ALU flag unit and one iterative multiplexer, by reusing the binary search engine inside the Position Location Unit. The use of the binary search engine has been explained and illustrated in detail in a previous chapter and it is mainly used by the Position Location Unit to process the touch position. In this proposed optimized structure, the binary search counter is also accessed by the Interpolation Unit during a different timing division to accelerate the iterative process. Both the interpolation path and the triangulation path inside the ALU have access to the binary search engine at different time divisions, leading to the elimination of the Divide operator.

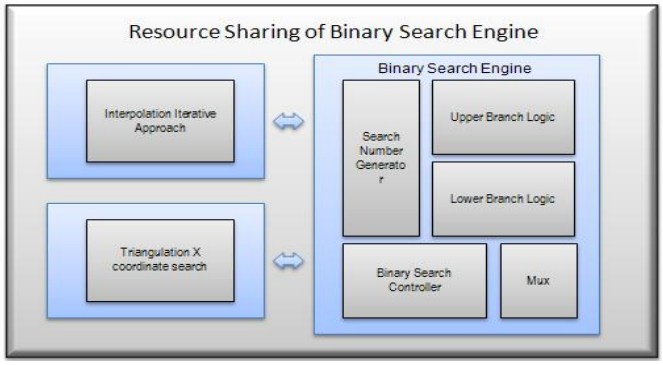


Figure 82 Resource Sharing of Binary Search Engine

The Common ALU has been optimized to a new divider-free structure:

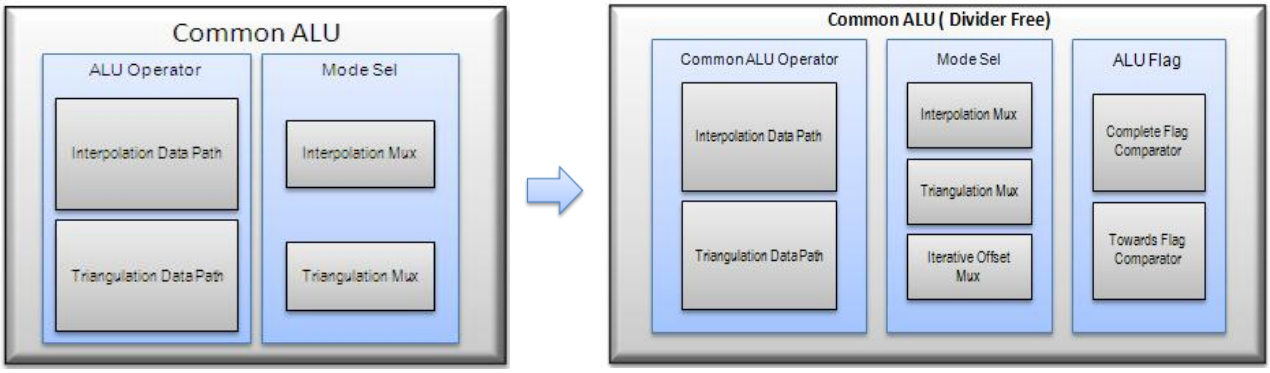


Figure 83 Divider Free Arithmetic Logic Unit

The optimised Common ALU has three major parts: the ALU operator part which is made up of the Interpolation Data Path and Triangulation Data Path; the Mode Select part which is signalled

by the master controller to switch between interpolation and triangulation and the ALU Flag part which is a necessary unit to signal outside the binary search engine for iterative implementation. The abstracted circuit level of the optimized common ALU is illustrated below:

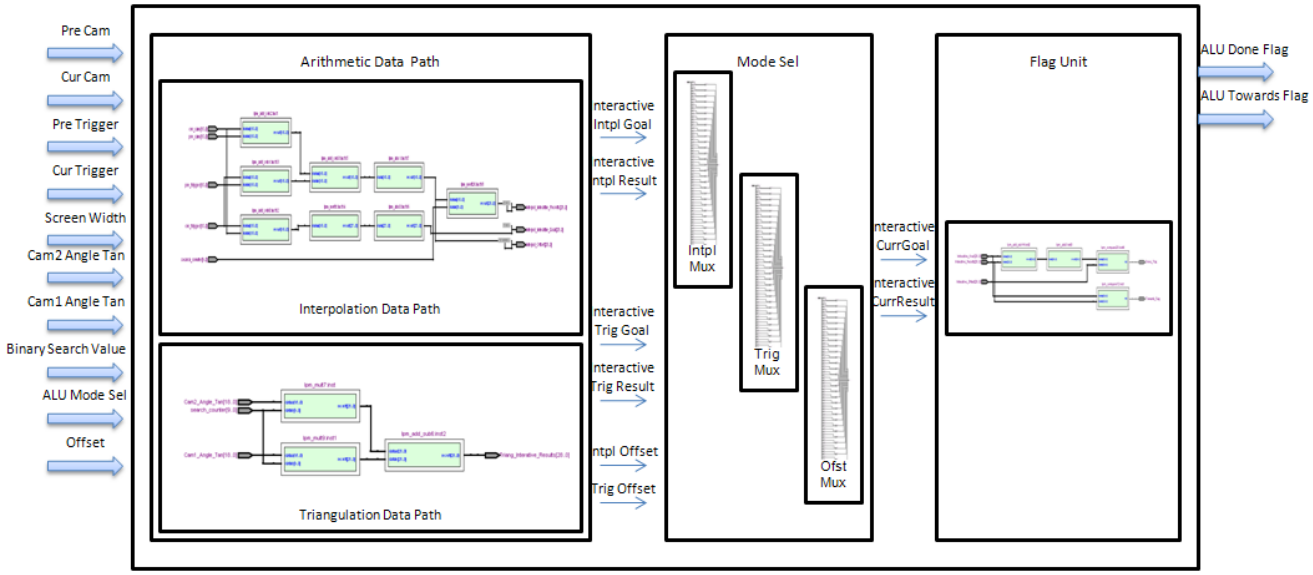


Figure 84 Common ALU Abstracted Circuit Structure

Both the Interpolation and Triangulation Data Paths are comprised of computationally simple operators. The Interpolation Data Path has four inputs (previous pixel, previous trigger, current pixel and current trigger) propagating through the data path in three cycles, and has Interpolation Iterative Goal (used as the numerator of a divide) and Interpolation Iterative Attempts (used as the denominator of a divide) as outputs. The Triangulation Data Path has three inputs (screen width, cam2 angle tan and cam1 angle tan) propagating through the data path in two cycles, and has Triangulation Iterative Goal (which is the screen width) and Interpolation Iterative Attempts (addition of multipliers of both angle tan with search value) as outputs. The Iterative Goals and Iterative Attempts are directed by the Mode Multiplexer into the binary search engine to calculate the result with the accuracy determined by the offset. After each cycle, the Flag Unit will generate Iterative Done when a sufficiently accurate answer is found or Iterative Towards Flag to guide the binary search engine to move towards required answer. Worst case latency is 12 cycles for interpolation arithmetic processing (3 cycles for propagation and 9 cycles maximum for iteration) and 11 cycles for triangulation (2 cycles for propagation and 9 cycles maximum for iteration).

5.3.5 Retiming and Register Balancing

Retiming and register balancing are common techniques applied to register level optimization which are based on the principle of balancing out negative and positive slacks throughout the design structure. The worst-case delay will be minimized between any two register stages based on this method by relocating flip flops around logic.

The concept of retiming and register balancing is illustrated below from [65] and [66]:

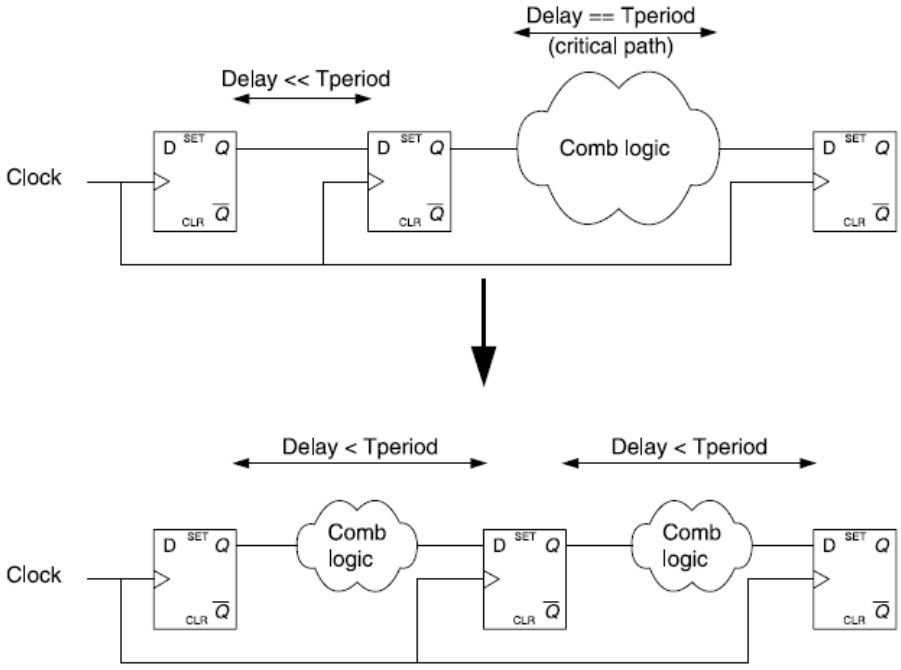


Figure 85 Retiming and Register Balancing Illustration

Some research such as [67] has proposed improving the general retiming procedure by applying a novel polynomial time algorithm with forward retiming to minimize the clock period. In our touch screen system optimization, since timing is not the high priority, instead of manually manipulating numerous registers at a very low level with the new algorithm, an optimization option provided by the design vendor in the synthesis tool is adopted, with a robust ability for redistributing logic.

Retiming and register balancing option is offered in the Quartus EDA tool [68]:

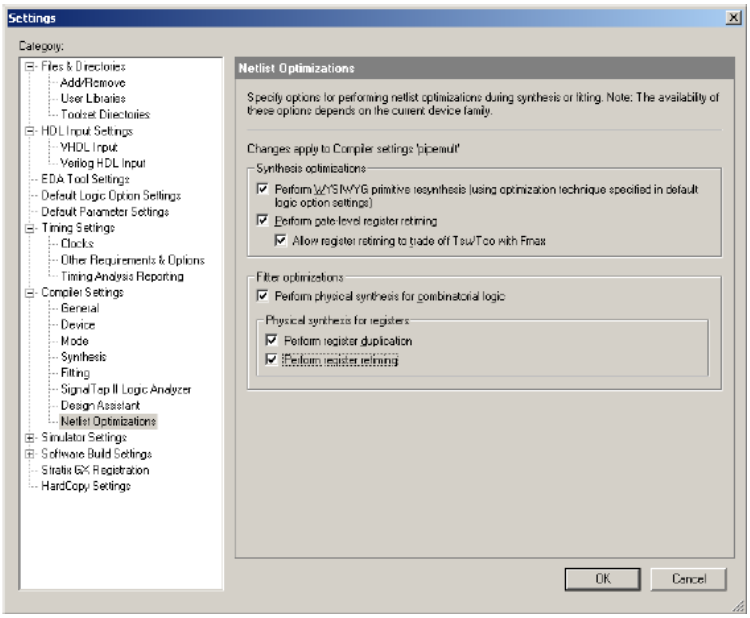


Figure 86 Retiming and Register Balancing Option

The optimization results of retiming and register balancing will be presented in a later section.

5.3.6 Multiplexer Resource Reduction

Multiplexers are fundamental digital components used in building structures for a number of applications such as a processor, function switch and many others. It is estimated that more than 25% of an FPGA design area is constituted of multiplexers, from Altera Benchmark analysis[69]. In this proposed touch screen system, a larger amount of multiplexer based resources are utilized by the Processing Unit (PU) which is heavy in both arithmetic processing and function switching. Therefore, the optimization of multiplexer based hardware resources is expected to have a significant impact on overall system efficiency.

A multiplexer restructuring algorithm has been researched by a group of engineers [70] to reduce LUT based multiplexer resource consumption by an average of 18%. The optimization across busses of multiplexers is the core part of this new method which allows area reduction to be made in every part of the bus at the cost of additional control logic.

There are two basic elements of this new algorithm: One is compression, the other is balancing.

Compression:

The compression process in Metzgen and Nancekievill's [70] algorithm converts groups of 2:1 multiplexers into the more area efficient 4:1 multiplexer: the following is the illustration from [70]:

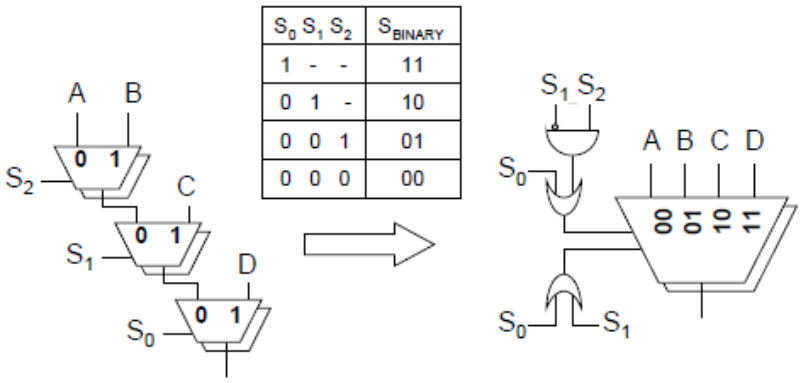


Figure 87 Multiplexer Restructuring Algorithm Compression

Balancing:

Balancing is a restructuring process used because some structures cannot be clustered after compression. A minimal amount of restructuring has been performed by balancing to achieve better performance. The concept of balancing is illustrated below from [70]:

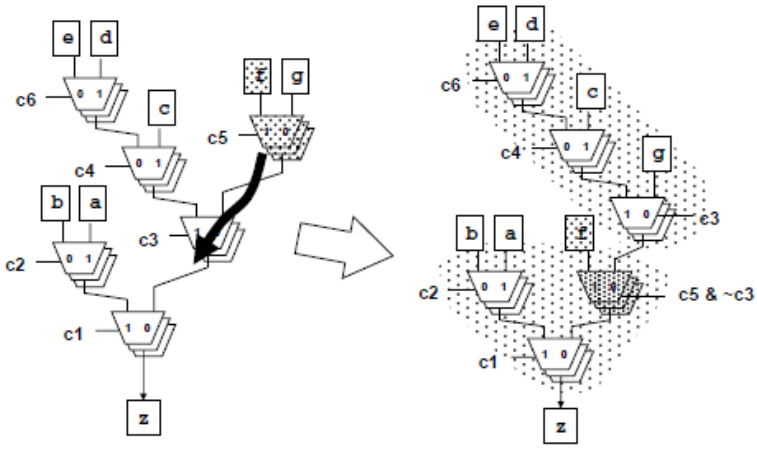
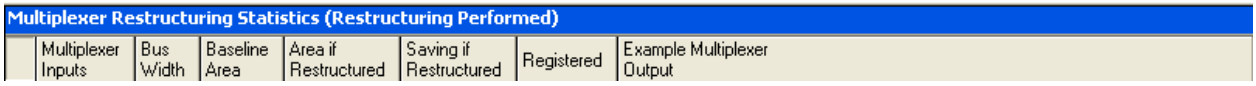


Figure 88 Multiplexer Restructuring Algorithm Balancing

This algorithm has been integrated in synthesis tools as an optimization option [68]:

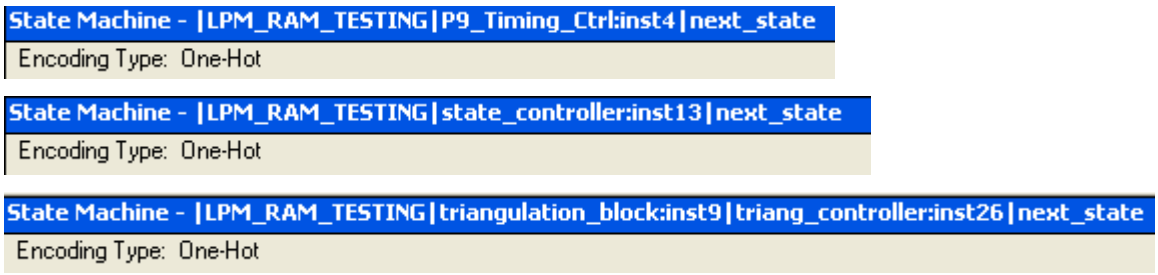


The optimized results will be presented in the Results chapter.

5.3.7 State Machine Optimization

There are three possible encoding styles that can be used in constructing a state machine: Sequential, Gray and One-shot. A sequential state machine is normally used in designs when area is the highest priority instead of timing; The Gray encoding style should be glitchless, where only one flip flop changes during a transition; A One-shot state machine provides the best performance and shortest clock-to-output delay, but consumes more resources than sequential encoding.

In the proposed touch screen system design, since most control signals to or from the state machines have significant timing requirements, all three state machines (in the Acquisition Controller, Position Localization Controller and Master Controller) have been chosen to be one-shot encoding during the optimization. Nonetheless, Liu, Sun and Zhao [71] have considered the potential unreliability of using one-shot encoding, which needs further investigation in the future.



Two optimization techniques are implemented manually on the design level since a few areas are not considered and covered by the automatic synthesis tool appropriately: one is removal of unreachable states and the other is separation of control path and data path.

Removal of unreachable states:

The purpose of this technique is to increase the reliability of the circuit design with extra “safe mode” logic added to cover all states, even if they are unreachable through normal operation.

Separation of control path and data path inside the state machine:

The data path refers to the channel carrying data from the the inputs of system to the outputs, while control path refers to state transitions in the state machine which generate control signals that further configure the data path for various operations. The original design of the state machines in the touch screen system had control and data paths mixed, which caused a discrepancy between their timing requirements, because the control path has slower timing requirements. In order to achieve a high system throughput, inspired by Synplify's design methodology [72], the data path has been manually separated from the control path with necessary connections to control signals. The following example shows this technique applied to the Acquisition Controller state machine:

Original flow:

Figure 89 Original Acquisition Controller Data and Control Path

Optimized flow:

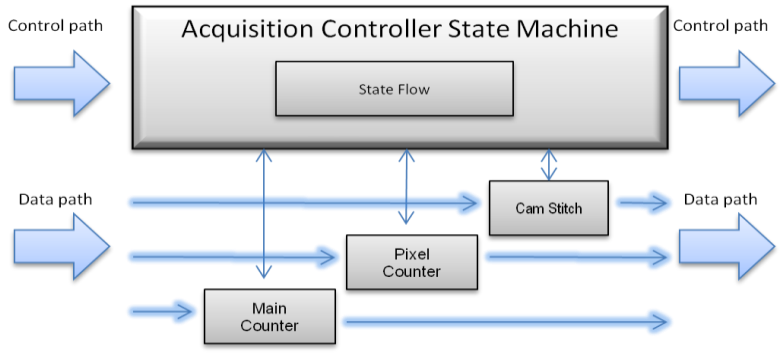


Figure 90 Optimized Acquisition Controller Data and Control Path

5.4 Floor Planning

Floor planning is an important placement technique applied on the physical logic gate level to achieve timing closure and high data throughput by reorganizing the routing delays. Since FPGA device densities are large (millions of gates), a number of placement tools have been provided by the EDA vendor to facilitate the floor planning process. In the proposed touch screen system development, an integrated Chip Editor[73] has been used to modify and optimize the post timing and netlists. The following is the original touch screen system on the physical chip view:

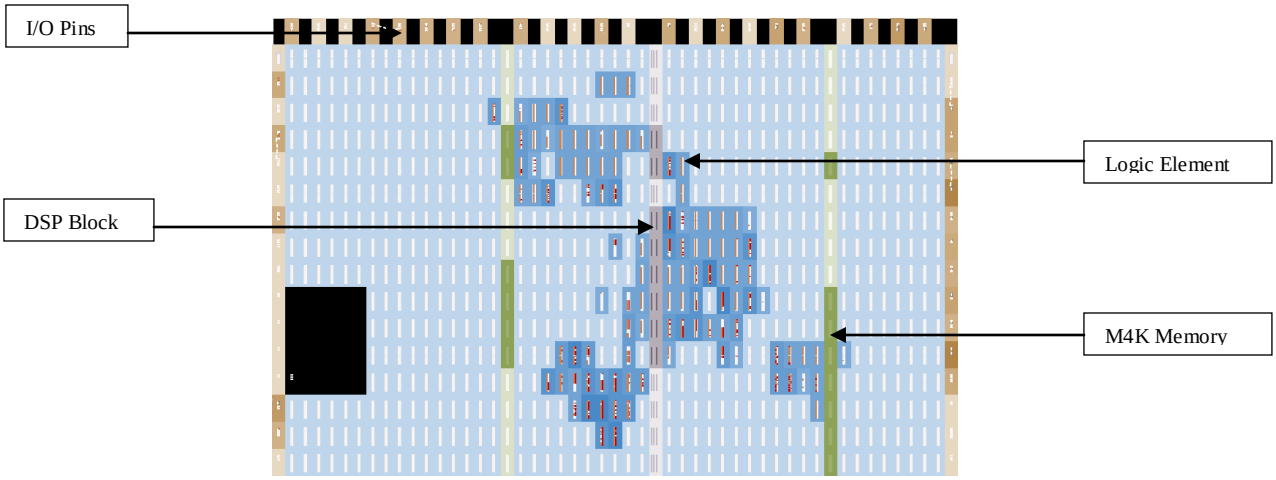


Figure 91 Physical Chip Overview

On the field view, the touch screen system is constructed based on I/O pins which control and respond to outside chip components, the DSP block which is particularly used to process arithmetic functions (such as operators in the Common ALU), M4K Memory which stores all camera frames and Logic Elements (LE) which are the smallest unit inside the FPGA chip with efficient logic utilization. Each LE contains a 4-input LUT, programmable register and carry chain and interconnects to organize a large number of LEs to perform more advanced functionalities. The detail inside each logic element is illustrated below:

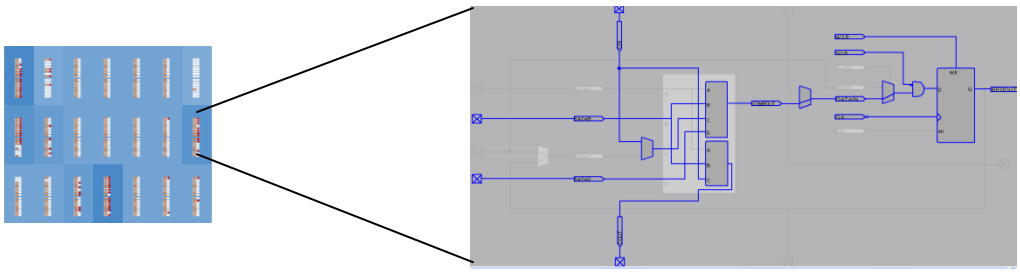


Figure 92 Logic Element Configuration

Most optimizations for floor planning will be handled by the Chip Editor automatically based on a few methodologies developed by the tool vendor. One of the most important floor planning techniques has been implemented manually in the touch screen system optimization to achieve a better result.

Floor plan Partitioning

Instead of completely relying on automatic tools, a significant number of modifications have been made manually to clearly define partitions according to functionalities and the interconnections between partitions. There are two major parts in the touch screen system: one is the data acquisition system and the other is the processing system. The original design is not partitioned appropriately with a number of interferences between processing sub-modules and the acquisition system. The original design partition is illustrated below:

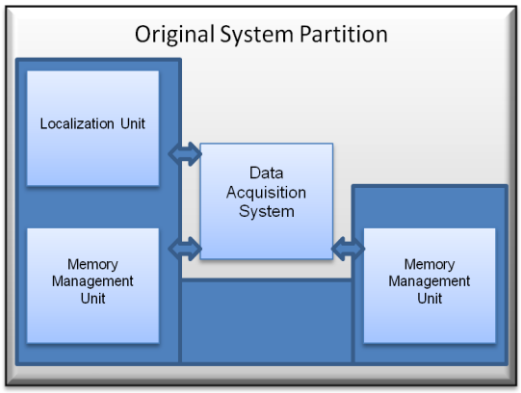


Figure 93 Original System Partition Design

In the optimized version, the interconnections between the two major partitions have been combined with all processing sub-modules placed tightly on the chip to reduce routing delay. The optimized design partition is illustrated below:

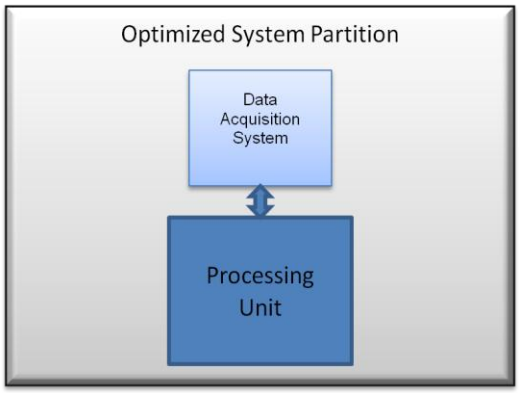


Figure 94 Optimized System Partition Design

5.5 Reducing Power Dissipation

The proposed touch screen system is constructed based on a 90-nm Cyclone II FPGA with a number of techniques for lowering power consumption in silicon such as decreased core voltage, increased transistor length, lower I/O pin capacitance and a power efficient clocking structure. The following is the typical static power consumption of the latest Cyclone III FPGA[74]:

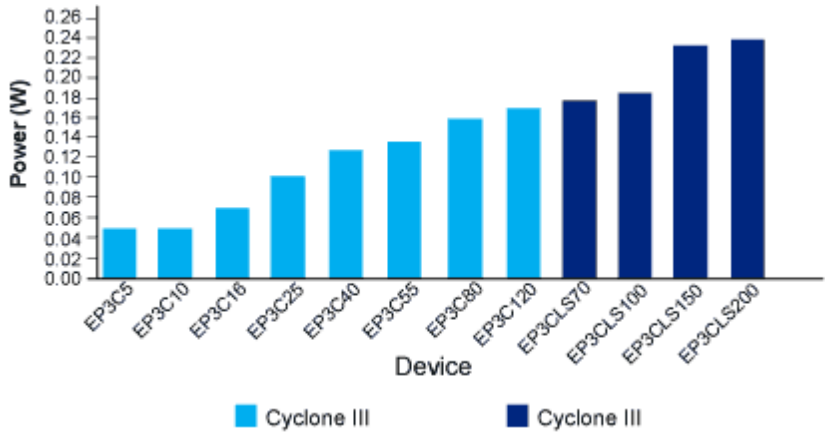


Figure 95 Programmable Chip Power Consumption

The figure shows the static power consumption of different Cyclone devices at 85 degree junction temperature. The most powerful Cyclone device consumes as little as 238 mW static power. During system power reduction, the integrated tool PowerPlay[75] is used to optimize power consumption at both synthesis and routing stages.

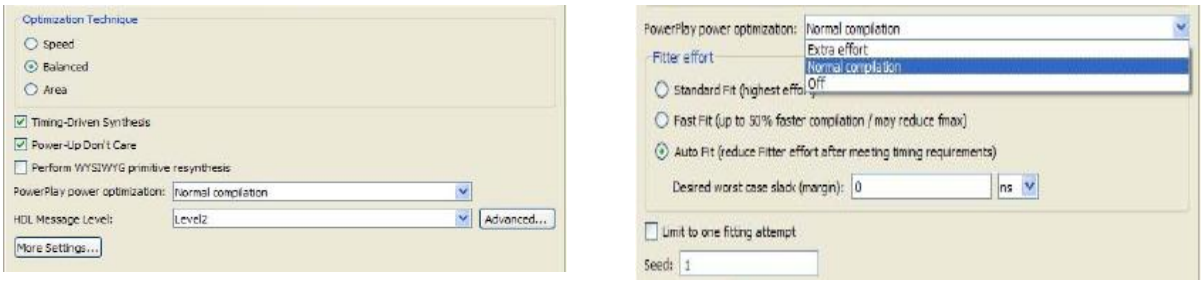


Figure 96 Power Optimization Options

Normal compilation has been selected as the optimization option where low compute effort integrated algorithms are applied to minimize power through netlist optimizations, as long as they are not expected to reduce design performance. By applying the power-driven techniques integrated in the automatic tool, power consumption could be reduced.

Chapter 6 Touch Screen System Optimization Results

The proposed touch screen system has been optimized through different versions with different orientations. The system hardware consumption is comprised of two major sources: Logic Elements which are the smallest logic units for constructing functionality and Memory bits which are the basic storage element. Two versions have been implemented, focusing on memory reduction and logic element reduction respectively with techniques introduced in the previous chapter.

6.1 Resource Optimization Results --- Original Version

The original hardware touch screen system, without any optimization techniques applied, consumes 2407 logic elements and 52224 memory bits. In the logic consumption, two dividers inside the Position Localization Unit (PLU) and Edge Localization Unit (ELU) constitute a large part of the logic resource with certain key controller structures (acquisition control, ADC control) consuming a considerable part of the other resources. Meanwhile, memory resources are consumed by storage for two camera images. The detailed resource analysis for each system module is listed below:

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
Cyclone II: EP2C20F484C7													
LPM_RAM_TESTING	2407 (3)	688 (0)	0 (0)	52224	14	16	0	8	185	0	1719 (3)	137 (0)	551 (0)
Mem_Management:inst	82 (0)	0 (0)	0 (0)	32768	8	0	0	0	0	0	67 (0)	0 (0)	15 (0)
mem_demux:inst	25 (25)	0 (0)	0 (0)	0	0	0	0	0	0	0	25 (25)	0 (0)	0 (0)
lpm_ram_dp4:inst1	0 (0)	0 (0)	0 (0)	16384	4	0	0	0	0	0	0 (0)	0 (0)	0 (0)
lpm_ram_dp5:inst2	0 (0)	0 (0)	0 (0)	16384	4	0	0	0	0	0	0 (0)	0 (0)	0 (0)
lpm_mux0:inst3	16 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	16 (0)	0 (0)	0 (0)
Trigger_Logic_75:inst4	41 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	26 (0)	0 (0)	15 (0)
Clock_Divd:inst1	2 (2)	2 (2)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	2 (2)
TIADC_Interface:inst2	88 (88)	46 (46)	0 (0)	0	0	0	0	0	0	0	42 (42)	0 (0)	46 (46)
ft245r:inst3	137 (137)	61 (61)	0 (0)	0	0	0	0	0	0	0	76 (76)	3 (3)	58 (58)
P9_Timing_Ctrl:inst4	445 (445)	155 (155)	0 (0)	0	0	0	0	0	0	0	290 (290)	0 (0)	155 (155)
lpm_counter0:inst5	10 (0)	10 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	10 (0)
demux:inst7	24 (24)	0 (0)	0 (0)	0	0	0	0	0	0	0	18 (18)	0 (0)	6 (6)
Cam_coef_LUT:inst8	11 (0)	0 (0)	0 (0)	19456	6	4	0	2	0	0	11 (0)	0 (0)	0 (0)
Test_ROM:inst	0 (0)	0 (0)	0 (0)	9728	3	0	0	0	0	0	0 (0)	0 (0)	0 (0)
Test_ROM_2:inst1	0 (0)	0 (0)	0 (0)	9728	3	0	0	0	0	0	0 (0)	0 (0)	0 (0)
lpm_mult24:inst2	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
triangulation_block:inst9	526 (0)	50 (0)	0 (0)	0	0	12	0	6	0	0	458 (0)	0 (0)	68 (0)
triangulation_reg:inst	12 (12)	12 (12)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	12 (12)
lpm_mult7:inst1	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
lpm_add_sub6:inst2	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
lpm_compare12:inst3	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
pixel_gen:inst4	20 (20)	19 (19)	0 (0)	0	0	0	0	0	0	0	1 (1)	0 (0)	19 (19)

lpm_mult22:inst5	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
upper_mid:inst6	9 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	2 (0)	0 (0)	7 (0)
lower_mid:inst7	0	0	0	0	0	0	0	0	0	0	0	0	0
lpm_add_sub14:inst8	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
lpm_mux4:inst9	0	0	0	0	0	0	0	0	0	0	0	0	0
lpm_mult9:inst10	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
lpm_divide3:inst11	291 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	274 (0)	0 (0)	17 (0)
lpm_abs3:inst19	30 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	30 (0)	0 (0)	0 (0)
lpm_compare20:inst20	10 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	9 (0)	0 (0)	1 (0)
triang_controller:inst26	38 (38)	19 (19)	0 (0)	0	0	0	0	0	0	0	19 (19)	0 (0)	19 (19)
addr_demux:inst10	39 (39)	0 (0)	0 (0)	0	0	0	0	0	0	0	21 (21)	0 (0)	18 (18)
state_controller:inst13	110 (110)	89 (89)	0 (0)	0	0	0	0	0	0	0	19 (19)	23 (23)	68 (68)
Normalizer:inst15	0	0	0	0	0	0	0	0	0	0	0	0	0
Edge_Detection_Unit:inst16	989 (0)	272 (0)	0 (0)	0	0	0	0	0	0	0	717 (0)	112 (0)	160 (0)
lpm_compare0:inst	14 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	14 (0)	0 (0)	0 (0)
touch_record_reg:inst1	1 (1)	1 (1)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	1 (1)
trigger_cur_reg:inst2	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (16)
trigger_pre_reg:inst3	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (16)
cam_cur_reg:inst4	6 (6)	6 (6)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	6 (6)
cam_pre_reg:inst5	6 (6)	6 (6)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	6 (6)
interpolation_gate:inst6	0	0	0	0	0	0	0	0	0	0	0	0	0
Interpolation_Operator:inst7	805 (0)	65 (0)	0 (0)	0	0	0	0	0	0	0	699 (0)	5 (0)	101 (0)
lpm_add_sub0:inst	6 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	6 (0)	0 (0)	0 (0)
lpm_add_sub1:inst1	6 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	6 (0)
lpm_add_sub2:inst2	16 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (0)
lpm_mult1:inst3	45 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	45 (0)	0 (0)	0 (0)
lpm_add_sub3:inst4	13 (0)	1 (0)	0 (0)	0	0	0	0	0	0	0	12 (0)	0 (0)	1 (0)
lpm_abs0:inst5	22 (0)	15 (0)	0 (0)	0	0	0	0	0	0	0	7 (0)	0 (0)	15 (0)
lpm_abs1:inst6	29 (0)	17 (0)	0 (0)	0	0	0	0	0	0	0	11 (0)	4 (0)	14 (0)
lpm_divide0:inst7	650 (0)	32 (0)	0 (0)	0	0	0	0	0	0	0	610 (0)	1 (0)	39 (0)
lpm_add_sub4:inst8	27 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	8 (0)	0 (0)	19 (0)
falling_rising_demux:inst8	0	0	0	0	0	0	0	0	0	0	0	0	0
cam1_edge_reg:inst9	83 (83)	81 (81)	0 (0)	0	0	0	0	0	0	0	2 (2)	49 (49)	32 (32)
cam2_edge_reg:inst10	83 (83)	81 (81)	0 (0)	0	0	0	0	0	0	0	2 (2)	58 (58)	23 (23)

Table 4 Original System Resource Analysis Table

6.2 Resource Optimization Results --- Memory-Oriented Optimization

Memory-oriented optimization has focused on system memory reduction, based on the memory optimization techniques applied in the previous chapter. Full range pixel (including dummy pixels) based frame storage has been reduced to active pixel based efficient frame storage. An approximate 32% improvement from 52224 memory bits to 35840 memory bits has been achieved, based on the previous optimization method. The detail memory resource analysis is listed below:

32% memory reduction

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
Cyclone II: EP2C20F484C7													
LPM_RAM_TESTING	2403 (3)	685 (0)	0 (0)	35840	10	16	0	8	185	0	1718 (3)	139 (0)	546 (0)
Mem_Management:inst	82 (0)	0 (0)	0 (0)	16384	4	0	0	0	0	0	66 (0)	0 (0)	16 (0)
mem_demux:inst	25 (25)	0 (0)	0 (0)	0	0	0	0	0	0	0	25 (25)	0 (0)	0 (0)
lpm_ram_dp4:inst1	0 (0)	0 (0)	0 (0)	8192	2	0	0	0	0	0	0 (0)	0 (0)	0 (0)
lpm_ram_dp5:inst2	0 (0)	0 (0)	0 (0)	8192	2	0	0	0	0	0	0 (0)	0 (0)	0 (0)
lpm_mux0:inst3	16 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	16 (0)	0 (0)	0 (0)
Trigger_Logic_75:inst4	41 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	25 (0)	0 (0)	16 (0)
Clock_Divd:inst1	2 (2)	2 (2)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	2 (2)
TIADC_Interface:inst2	88 (88)	46 (46)	0 (0)	0	0	0	0	0	0	0	42 (42)	0 (0)	46 (46)
ft245r:inst3	137 (137)	61 (61)	0 (0)	0	0	0	0	0	0	0	76 (76)	4 (4)	57 (57)
P9_Timing_Ctrl:inst4	445 (445)	155 (155)	0 (0)	0	0	0	0	0	0	0	290 (290)	0 (0)	155 (155)
lpm_counter0:inst5	10 (0)	10 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	10 (0)
demux:inst7	24 (24)	0 (0)	0 (0)	0	0	0	0	0	0	0	18 (18)	0 (0)	6 (6)
Cam_coef_LUT:inst8	11 (0)	0 (0)	0 (0)	19456	6	4	0	2	0	0	11 (0)	0 (0)	0 (0)
Test_ROM:inst	0 (0)	0 (0)	0 (0)	9728	3	0	0	0	0	0	0 (0)	0 (0)	0 (0)
Test_ROM_2:inst1	0 (0)	0 (0)	0 (0)	9728	3	0	0	0	0	0	0 (0)	0 (0)	0 (0)
lpm_mult24:inst2	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
triangulation_block:inst9	522 (0)	50 (0)	0 (0)	0	0	12	0	6	0	0	455 (0)	0 (0)	67 (0)
triangulation_reg:inst	12 (12)	12 (12)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	12 (12)
lpm_mult7:inst1	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
lpm_add_sub6:inst2	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
lpm_compare12:inst3	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
pixel_gen:inst4	20 (20)	19 (19)	0 (0)	0	0	0	0	0	0	0	1 (1)	0 (0)	19 (19)
lpm_mult22:inst5	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
upper_mid:inst6	9 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	2 (0)	0 (0)	7 (0)
lower_mid:inst7	0	0	0	0	0	0	0	0	0	0	0	0	0
lpm_add_sub14:inst8	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
lpm_mux4:inst9	0	0	0	0	0	0	0	0	0	0	0	0	0
lpm_mult9:inst10	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
lpm_divide3:inst11	291 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	274 (0)	0 (0)	17 (0)
lpm_abs3:inst19	30 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	30 (0)	0 (0)	0 (0)
lpm_compare20:inst20	10 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	9 (0)	0 (0)	1 (0)
triang_controller:inst26	38 (38)	19 (19)	0 (0)	0	0	0	0	0	0	0	19 (19)	0 (0)	19 (19)
addr_demux:inst10	39 (39)	0 (0)	0 (0)	0	0	0	0	0	0	0	21 (21)	0 (0)	18 (18)
state_controller:inst13	110 (110)	89 (89)	0 (0)	0	0	0	0	0	0	0	19 (19)	23 (23)	68 (68)
Normalizer:inst15	0	0	0	0	0	0	0	0	0	0	0	0	0
Edge_Detection_Unit:inst16	989 (0)	272 (0)	0 (0)	0	0	0	0	0	0	0	717 (0)	112 (0)	160 (0)
lpm_compare0:inst	14 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	14 (0)	0 (0)	0 (0)
touch_record_reg:inst1	1 (1)	1 (1)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	1 (1)
trigger_cur_reg:inst2	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (16)
trigger_pre_reg:inst3	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (16)
cam_cur_reg:inst4	6 (6)	6 (6)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	6 (6)
cam_pre_reg:inst5	6 (6)	6 (6)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	6 (6)
interpolation_gate:inst6	0	0	0	0	0	0	0	0	0	0	0	0	0
Interpolation_Operator:inst7	805 (0)	65 (0)	0 (0)	0	0	0	0	0	0	0	699 (0)	5 (0)	101 (0)
lpm_add_sub0:inst	6 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	6 (0)	0 (0)	0 (0)
lpm_add_sub1:inst1	6 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	6 (0)
lpm_add_sub2:inst2	16 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (0)
lpm_mult1:inst3	45 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	45 (0)	0 (0)	0 (0)
lpm_add_sub3:inst4	13 (0)	1 (0)	0 (0)	0	0	0	0	0	0	0	12 (0)	0 (0)	1 (0)
lpm_abs0:inst5	22 (0)	15 (0)	0 (0)	0	0	0	0	0	0	0	7 (0)	0 (0)	15 (0)
lpm_abs1:inst6	29 (0)	17 (0)	0 (0)	0	0	0	0	0	0	0	11 (0)	4 (0)	14 (0)
lpm_divide0:inst7	650 (0)	32 (0)	0 (0)	0	0	0	0	0	0	0	610 (0)	1 (0)	39 (0)
lpm_add_sub4:inst8	27 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	8 (0)	0 (0)	19 (0)
falling_rising_demux:inst8	0	0	0	0	0	0	0	0	0	0	0	0	0
cam1_edge_reg:inst9	83 (83)	81 (81)	0 (0)	0	0	0	0	0	0	0	2 (2)	49 (49)	32 (32)
cam2_edge_reg:inst10	83 (83)	81 (81)	0 (0)	0	0	0	0	0	0	0	2 (2)	58 (58)	23 (23)

Table 5 Optimized Memory Resource Analysis Table

6.3 Resource Optimization Results --- Logic Elements-Oriented Optimization

6.3.1 Algorithm alteration based logic optimization

The algorithm alteration based technique is the optimization method applied on the system level, where the gradient based edge detection algorithm has been evaluated and replaced in the previous chapter by a more efficient dynamic linear approximation detection algorithm. An approximate 15% improvement has been achieved on the pixel edge level from 117LE+110LE=227LE to 197LE. The new algorithm based detection structure is also further optimized on the sub-pixel edge level in following paragraphs. The detailed optimization list is shown below:

Gradient based edge detection algorithm resource list:

Edge_Detection_Unit:inst10	117 (0)	46 (0)	0 (0)	0	0	0	0	0	0	0	71 (0)	23 (0)	23 (0)
cam_generator:inst11	28 (28)	15 (15)	0 (0)	0	0	0	0	0	0	0	13 (13)	0 (0)	15 (15)
Edge_Localization_Unit:inst12	110 (1)	42 (0)	0 (0)	0	0	0	0	0	0	0	68 (1)	0 (0)	42 (0)
drt_demux:inst	40 (40)	0 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	40 (40)
lpm_compare1:inst3	27 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	27 (0)	0 (0)	0 (0)
lpm_compare2:inst4	27 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	27 (0)	0 (0)	0 (0)
edge_location_reg:inst5	21 (21)	10 (10)	0 (0)	0	0	0	0	0	0	0	11 (11)	0 (0)	10 (10)
rising_edge_reg:inst7	17 (17)	16 (16)	0 (0)	0	0	0	0	0	0	0	1 (1)	0 (0)	16 (16)
falling_edge_reg:inst8	17 (17)	16 (16)	0 (0)	0	0	0	0	0	0	0	1 (1)	0 (0)	16 (16)

Table 6 Gradient Based Edge Detection Algorithm Resource Analysis Table

Dynamic linear approximation detection algorithm resource list:

Edge_Detection_Unit:inst16	197 (0)	179 (0)	0 (0)	0	0	0	0	0	0	0	18 (0)	89 (0)	90 (0)
lpm_compare0:inst	14 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	14 (0)	0 (0)	0 (0)
touch_record_reg:inst1	1 (1)	1 (1)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	1 (1)
trigger_cur_reg:inst2	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (16)
trigger_pre_reg:inst3	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	3 (3)	13 (13)
cam_cur_reg:inst4	12 (12)	12 (12)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	12 (12)
cam_pre_reg:inst5	12 (12)	12 (12)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	12 (12)
interpolation_gate:inst6	28 (28)	28 (28)	0 (0)	0	0	0	0	0	0	0	0 (0)	28 (28)	0 (0)
falling_rising_demux:inst8	0	0	0	0	0	0	0	0	0	0	0	0	0
cam1_edge_reg:inst9	49 (49)	47 (47)	0 (0)	0	0	0	0	0	0	0	2 (2)	29 (29)	18 (18)
cam2_edge_reg:inst10	49 (49)	47 (47)	0 (0)	0	0	0	0	0	0	0	2 (2)	29 (29)	18 (18)

Table 7 Dynamic Linear Approximation Detection Algorithm Resource Analysis Table

6.3.2 Resource sharing, iterative approach and TDM based optimization results

In the previous chapter, a significant effort has been put into the optimization of two key blocks, the Edge Localization Unit (ELU) and the Position Localization Unit (PLU). Complicated and mixed optimization techniques have been applied to the original system to re-build an optimal computational system with a less expensive structure, based on the previously introduced

resource sharing, iterative and time-division-multiplexing methods. A common ALU module has been created to be shared by both the ELU and PLU arithmetic operations, the resource-consuming divide operator has been replaced by a counter based iterative approach and the binary search engine is accessed at different times by both the edge localization and position localization units. In the original ELU and PLU block resource analysis, a total of 1211LE was consumed in the design.

triangulation_block:inst9	221 (0)	50 (0)	0 (0)	0	0	12	0	6	0	0	171 (0)	0 (0)	50 (0)
triangulation_reg:inst	12 (12)	12 (12)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	12 (12)
lpm_mult7:inst1	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
lpm_add_sub6:inst2	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
lpm_compare12:inst3	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
pixel_gen:inst4	20 (20)	19 (19)	0 (0)	0	0	0	0	0	0	0	1 (1)	0 (0)	19 (19)
lpm_mult22:inst5	1 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	1 (0)	0 (0)	0 (0)
upper_mid:inst6	9 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	2 (0)	0 (0)	7 (0)
lower_mid:inst7	0	0	0	0	0	0	0	0	0	0	0	0	0
lpm_add_sub14:inst8	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
lpm_mux4:inst9	0	0	0	0	0	0	0	0	0	0	0	0	0
lpm_mult9:inst10	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
lpm_abs3:inst19	30 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	30 (0)	0 (0)	0 (0)
lpm_compare20:inst20	10 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	9 (0)	0 (0)	1 (0)
triang_controller:inst26	38 (38)	19 (19)	0 (0)	0	0	0	0	0	0	0	19 (19)	0 (0)	19 (19)
Edge_Detection_Unit:inst16	990 (0)	272 (0)	0 (0)	0	0	0	0	0	0	0	718 (0)	114 (0)	158 (0)
lpm_compare0:inst	14 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	13 (0)	0 (0)	1 (0)
touch_record_reg:inst1	1 (1)	1 (1)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	1 (1)
trigger_cur_reg:inst2	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (16)
trigger_pre_reg:inst3	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (16)
cam_cur_reg:inst4	6 (6)	6 (6)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	6 (6)
cam_pre_reg:inst5	6 (6)	6 (6)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	6 (6)
interpolation_gate:inst6	0	0	0	0	0	0	0	0	0	0	0	0	0
Interpolation_Operator:inst7	805 (0)	65 (0)	0 (0)	0	0	0	0	0	0	0	701 (0)	5 (0)	99 (0)
falling_rising_demux:inst8	0	0	0	0	0	0	0	0	0	0	0	0	0
cam1_edge_reg:inst9	83 (83)	81 (81)	0 (0)	0	0	0	0	0	0	0	2 (2)	54 (54)	27 (27)
cam2_edge_reg:inst10	83 (83)	81 (81)	0 (0)	0	0	0	0	0	0	0	2 (2)	55 (55)	26 (26)

Table 8 Original Edge and Position Localization Units Resource Analysis Table

The optimized edge and position localization blocks structure is made of 588 LE in total, resulting in a 52% logic reduction based on optimization techniques.

Edge_Detection_Unit:inst16	197 (0)	179 (0)	0 (0)	0	0	0	0	0	0	0	18 (0)	89 (0)	90 (0)
lpm_compare0:inst	14 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	14 (0)	0 (0)	0 (0)
touch_record_reg:inst1	1 (1)	1 (1)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	1 (1)
trigger_cur_reg:inst2	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	16 (16)
trigger_pre_reg:inst3	16 (16)	16 (16)	0 (0)	0	0	0	0	0	0	0	0 (0)	3 (3)	13 (13)
cam_cur_reg:inst4	12 (12)	12 (12)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	12 (12)
cam_pre_reg:inst5	12 (12)	12 (12)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	12 (12)
interpolation_gate:inst6	28 (28)	28 (28)	0 (0)	0	0	0	0	0	0	0	0 (0)	28 (28)	0 (0)
falling_rising_demux:inst8	0	0	0	0	0	0	0	0	0	0	0	0	0
cam1_edge_reg:inst9	49 (49)	47 (47)	0 (0)	0	0	0	0	0	0	0	2 (2)	29 (29)	18 (18)
cam2_edge_reg:inst10	49 (49)	47 (47)	0 (0)	0	0	0	0	0	0	0	2 (2)	29 (29)	18 (18)
triangulation_block:inst9	107 (0)	82 (0)	0 (0)	0	0	4	0	2	0	0	25 (0)	17 (0)	65 (0)
triangulation_reg:inst	49 (49)	44 (44)	0 (0)	0	0	0	0	0	0	0	5 (5)	10 (10)	34 (34)
pixel_gen:inst4	19 (19)	19 (19)	0 (0)	0	0	0	0	0	0	0	0 (0)	7 (7)	12 (12)
lpm_mult22:inst5	1 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	1 (0)	0 (0)	0 (0)
upper_mid:inst6	0	0	0	0	0	0	0	0	0	0	0	0	0
lower_mid:inst7	0	0	0	0	0	0	0	0	0	0	0	0	0
lpm_mux4:inst9	0	0	0	0	0	0	0	0	0	0	0	0	0
triang_controller:inst26	38 (38)	19 (19)	0 (0)	0	0	0	0	0	0	0	19 (19)	0 (0)	19 (19)
Common_ALU:inst6	284 (0)	16 (0)	0 (0)	0	0	10	0	5	0	0	231 (0)	0 (0)	53 (0)
Interpol_Operator:inst	90 (0)	16 (0)	0 (0)	0	0	2	0	1	0	0	49 (0)	0 (0)	41 (0)
Triang_Operator:inst1	79 (0)	0 (0)	0 (0)	0	0	8	0	4	0	0	79 (0)	0 (0)	0 (0)
ALU_Flag_Block:inst2	82 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	70 (0)	0 (0)	12 (0)
lpm_mux5:inst7	18 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	18 (0)	0 (0)	0 (0)
lpm_mux6:inst8	0	0	0	0	0	0	0	0	0	0	0	0	0
lpm_mux7:inst10	15 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	15 (0)	0 (0)	0 (0)

Table 9 Optimized Edge and Position Localization Units Resource Analysis Table

Based on the resource sharing technique, further optimization has been applied to key control modules (acquisition controller, and ADC interface) on a sub-module level. The original acquisition controller and ADC interface resource analysis results are listed below, with 88 LE for the ADC and 445LE for the acquisition controller:

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
Cyclone II: EP2C20F484C7													
LPM_RAM_TESTING	2407 (3)	688 (0)	0 (0)	52224	14	16	0	8	185	0	1719 (3)	137 (0)	551 (0)
Mem_Management:inst	82 (0)	0 (0)	0 (0)	32768	8	0	0	0	0	0	67 (0)	0 (0)	15 (0)
mem_demux:inst	25 (25)	0 (0)	0 (0)	0	0	0	0	0	0	0	25 (25)	0 (0)	0 (0)
lpm_ram_dp4:inst1	0 (0)	0 (0)	0 (0)	16384	4	0	0	0	0	0	0 (0)	0 (0)	0 (0)
lpm_ram_dp5:inst2	0 (0)	0 (0)	0 (0)	16384	4	0	0	0	0	0	0 (0)	0 (0)	0 (0)
lpm_mux0:inst3	16 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	16 (0)	0 (0)	0 (0)
Trigger_Logic_75:inst4	41 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	26 (0)	0 (0)	15 (0)
Clock_Divd:inst1	2 (2)	2 (2)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	2 (2)
TIADC_Interface:inst2	88 (88)	46 (46)	0 (0)	0	0	0	0	0	0	0	42 (42)	0 (0)	46 (46)
ft245r:inst3	137 (137)	61 (61)	0 (0)	0	0	0	0	0	0	0	76 (76)	3 (3)	58 (58)
P9_Timing_Ctrl:inst4	445 (445)	155 (155)	0 (0)	0	0	0	0	0	0	0	290 (290)	0 (0)	155 (155)
lpm_counter0:inst5	10 (0)	10 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	10 (0)
demux:inst7	24 (24)	0 (0)	0 (0)	0	0	0	0	0	0	0	18 (18)	0 (0)	6 (6)
Cam_coef_LUT:inst8	11 (0)	0 (0)	0 (0)	19456	6	4	0	2	0	0	11 (0)	0 (0)	0 (0)
Test_ROM:inst	0 (0)	0 (0)	0 (0)	9728	3	0	0	0	0	0	0 (0)	0 (0)	0 (0)
Test_ROM_2:inst1	0 (0)	0 (0)	0 (0)	9728	3	0	0	0	0	0	0 (0)	0 (0)	0 (0)
lpm_mult24:inst2	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
triangulation_block:inst9	526 (0)	50 (0)	0 (0)	0	0	12	0	6	0	0	458 (0)	0 (0)	68 (0)
triangulation_reg:inst	12 (12)	12 (12)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	12 (12)
lpm_mult7:inst1	11 (0)	0 (0)	0 (0)	0	0	4	0	2	0	0	11 (0)	0 (0)	0 (0)
lpm_add_sub6:inst2	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
lpm_compare12:inst3	29 (0)	0 (0)	0 (0)	0	0	0	0	0	0	0	29 (0)	0 (0)	0 (0)
pixel_gen:inst4	20 (20)	19 (19)	0 (0)	0	0	0	0	0	0	0	1 (1)	0 (0)	19 (19)

Table 10 Original Acquisition Controller and ADC Interface Resource Analysis Table

The optimized acquisition engine has a reduction of 55% resource consumption from 445 to 202LE.

Project Navigator

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
Cyclone II: EP2C20F484C7													
Timing_Engine_Opt	202 (0)	93 (0)	0 (0)	0	0	0	0	0	19	0	109 (0)	2 (0)	91 (0)
Divide_by_128:inst	7 (0)	7 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	7 (0)
Divide_by_256:inst1	8 (0)	8 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	8 (0)
Divide_by_4:inst2	2 (0)	2 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	2 (0)
P9_Timing_Ctrl:inst4	185 (147)	76 (47)	0 (0)	0	0	0	0	0	0	0	109 (100)	2 (2)	74 (45)
Cam_Switch_Counter:CamS...	1 (1)	1 (1)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	1 (1)
Cam_Clk_Demux:Demux	2 (2)	0 (0)	0 (0)	0	0	0	0	0	0	0	2 (2)	0 (0)	0 (0)
Pixel_Map_Generator:PMG	11 (11)	5 (5)	0 (0)	0	0	0	0	0	0	0	6 (6)	0 (0)	5 (5)
SHIFT8:Shifter	5 (5)	4 (4)	0 (0)	0	0	0	0	0	0	0	1 (1)	0 (0)	4 (4)
Sub_Pixel_Counter:SPCnt	5 (5)	5 (5)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	5 (5)
System_Counter:SysCnt	14 (14)	14 (14)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	14 (14)

Table 11 Optimized Acquisition Engine Resource Analysis Table

The optimized ADC interface has a reduction of 11% resource consumption from 88 to 79LE

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
Cyclone II: EP2C20F484C7													
ADC_IF	79 (0)	45 (0)	0 (0)	0	0	0	0	0	30	0	34 (0)	25 (0)	20 (0)
TIADC_Interface:inst	78 (41)	44 (9)	0 (0)	0	0	0	0	0	0	0	34 (32)	25 (0)	19 (9)
Config_Convt_Counter:CCCnt	7 (7)	7 (7)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	7 (7)
SHIFT16:CFRCnt	17 (17)	16 (16)	0 (0)	0	0	0	0	0	0	0	1 (1)	13 (13)	3 (3)
ADC_Results_Reg:RegCnt	13 (13)	12 (12)	0 (0)	0	0	0	0	0	0	0	1 (1)	12 (12)	0 (0)
Divide_by_2:inst1	1 (0)	1 (0)	0 (0)	0	0	0	0	0	0	0	0 (0)	0 (0)	1 (0)

Table 12 Optimized ADC Interface Resource Analysis Table

6.3.3 Multiplexer restructuring algorithm based logic reduction results

The wide range of optimization methods implemented in the previous chapter have a significant impact on system logic element optimization. Based on the multiplexer restructuring algorithm, the number of multiplexers has been reduced in many system modules: the acquisition controller, ADC controller, memory management unit and so on. The details are listed below:

Multiplexer Restructuring Statistics (Restructuring Performed)

	Multiplexer Inputs	Bus Width	Baseline Area	Area if Restructured	Saving if Restructured	Registered	Example Multiplexer Output
1	3:1	7 bits	14 LEs	7 LEs	7 LEs	Yes	|LPM_RAM_TESTING|TIADC_Interface:inst2|CFR_count[0]
2	3:1	5 bits	10 LEs	5 LEs	5 LEs	Yes	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|TwoHdrK_clock_count[3]
3	3:1	5 bits	10 LEs	5 LEs	5 LEs	Yes	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|Op_clock_count[4]
4	3:1	12 bits	24 LEs	12 LEs	12 LEs	Yes	|LPM_RAM_TESTING|TIADC_Interface:inst2|Temp[11]
5	3:1	4 bits	8 LEs	4 LEs	4 LEs	Yes	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|FourHdrK_clock_count[1]
6	4:1	7 bits	14 LEs	7 LEs	7 LEs	Yes	|LPM_RAM_TESTING|TIADC_Interface:inst2|Conv_count[0]
7	4:1	30 bits	60 LEs	30 LEs	30 LEs	Yes	|LPM_RAM_TESTING|TIADC_Interface:inst2|Config[2]
8	5:1	7 bits	21 LEs	7 LEs	14 LEs	Yes	|LPM_RAM_TESTING|triangulation_block:inst9|triang_controller:inst26|Cycle_Count[0]
9	5:1	20 bits	60 LEs	20 LEs	40 LEs	Yes	|LPM_RAM_TESTING|lt245r:inst3|input_reg[19]
10	6:1	7 bits	28 LEs	7 LEs	21 LEs	Yes	|LPM_RAM_TESTING|triangulation_block:inst9|triang_controller:inst26|Count[0]
11	6:1	19 bits	76 LEs	19 LEs	57 LEs	Yes	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|pixel_count2[0]
12	6:1	19 bits	76 LEs	19 LEs	57 LEs	Yes	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|pixel_count1[5]
13	6:1	2 bits	8 LEs	4 LEs	4 LEs	Yes	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|CAM_Status[1]
14	8:1	10 bits	50 LEs	10 LEs	40 LEs	Yes	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|samp_one_count[0]
15	8:1	10 bits	50 LEs	10 LEs	40 LEs	Yes	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|samp_two_count[8]
16	16:1	6 bits	60 LEs	24 LEs	36 LEs	Yes	|LPM_RAM_TESTING|lt245r:inst3|temp_reg[5]
17	16:1	2 bits	20 LEs	6 LEs	14 LEs	Yes	|LPM_RAM_TESTING|lt245r:inst3|temp_reg[0]
18	17:1	9 bits	99 LEs	18 LEs	81 LEs	Yes	|LPM_RAM_TESTING|lt245r:inst3|\protocol:wait_count[7]
19	18:1	13 bits	156 LEs	13 LEs	143 LEs	Yes	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|count[8]
20	3:1	21 bits	42 LEs	42 LEs	0 LEs	No	|LPM_RAM_TESTING|Mem_Management:inst|mem_demux:inst|Cam2_mem[11]
21	3:1	12 bits	24 LEs	24 LEs	0 LEs	No	|LPM_RAM_TESTING|Mem_Management:inst|mem_demux:inst|Cam1_mem[13]
22	12:1	16 bits	128 LEs	32 LEs	96 LEs	No	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|op_count
23	29:1	4 bits	76 LEs	68 LEs	8 LEs	No	|LPM_RAM_TESTING|P9_Timing_Ctrl:inst4|Selector123

Table 13 Multiplexer Restructuring Results Table

6.3.4 Retiming and Register Balancing based optimization results

The Retiming and Register Balancing technique aims at re-organizing negative and positive system slacks which will not reduce hardware resources but will increase system structure reliability.

The following table shows a number of registers inside different modules that have been modified, deleted and retimed to balance slacks:

Fitter Netlist Optimizations				
	Node	Action	Operation	Reason
1	Cam_coef_LUT:inst8|lpm_mult24:i...	Modified	Physical Synthesis	Timing optimization
2	Cam_coef_LUT:inst8|lpm_mult24:i...	Deleted	Physical Synthesis	Timing optimization
3	Cam_coef_LUT:inst8|lpm_mult24:i...	Modified	Physical Synthesis	Timing optimization
4	Common_ALU:inst6|ALU_Flag_Bl...	Modified	Physical Synthesis	Timing optimization
5	Common_ALU:inst6|ALU_Flag_Bl...	Modified	Physical Synthesis	Timing optimization
6	Common_ALU:inst6|ALU_Flag_Bl...	Retimed Register	Physical Synthesis	Timing optimization
7	Common_ALU:inst6|ALU_Flag_Bl...	Modified	Physical Synthesis	Timing optimization
8	Common_ALU:inst6|ALU_Flag_Bl...	Modified	Physical Synthesis	Timing optimization
9	Common_ALU:inst6|ALU_Flag_Bl...	Modified	Physical Synthesis	Timing optimization
10	Common_ALU:inst6|ALU_Flag_Bl...	Retimed Register	Physical Synthesis	Timing optimization
11	Common_ALU:inst6|Interpol_Oper...	Modified	Physical Synthesis	Timing optimization
12	Common_ALU:inst6|Interpol_Oper...	Modified	Physical Synthesis	Timing optimization
13	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
14	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
15	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
16	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
17	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
18	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
19	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
20	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
21	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
22	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
23	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
24	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
25	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
26	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
27	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
28	Common_ALU:inst6|Interpol_Oper...	Retimed Register	Physical Synthesis	Timing optimization
29	Common_ALU:inst6|Triang_Opera...	Modified	Physical Synthesis	Timing optimization

Table 14 Retiming and Register Balancing Results Table

6.4 Resource Optimization Conclusion

To conclude, an approximate 32% improvement from 52224 memory bits to 35840 memory bits has been achieved after applying the aforementioned memory optimization techniques:

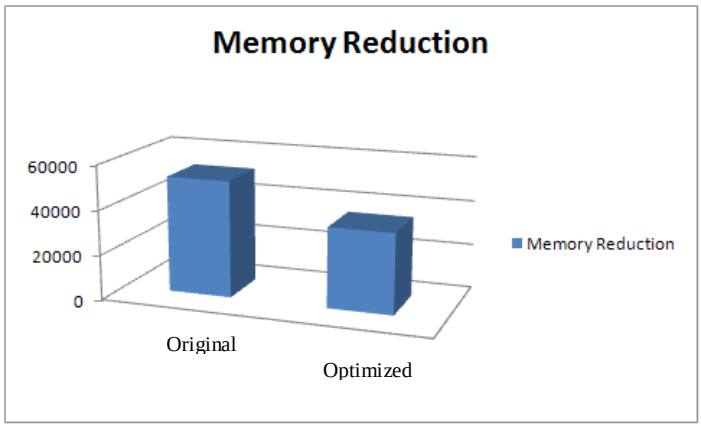


Figure 97 Memory Resource Reduction Result

For logic resources, using optimization methods on different levels such as algorithm alteration, pipeline rolling, resource sharing, iterative approach, Timing-Division-Multiplexing and register balancing, an approximate 40% improvement has been made from 2403 logic elements to 1445 logic elements without affecting overall system performance:

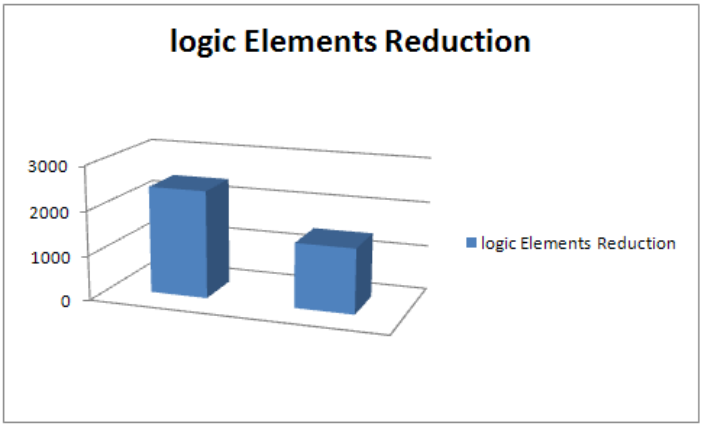


Figure 98 Logic Resource Reduction Result

6.5 Optimized Chip Floor Planning Result

After manually re-designing the system partition (connection between Data Acquisition System and Processing Unit) and adapting certain automatic floor planning tools, a more throughput-efficient floor planning result has been achieved. In the original version, key modules inside the Processing Unit such the master controller, edge localization unit and position localization unit are decentralized on the physical chip causing considerable interference with the data acquisition system partition. After optimization, all processing units are organized tightly with each other, having minimal cross-function timing delay. Meanwhile, there is only one interconnection between the acquisition partition and the processing partition to minimize possible interference. Dedicated DSP blocks and memory blocks are arranged close to processing units to achieve a low overall system propagation delay. The physical chip floor planning is illustrated below:

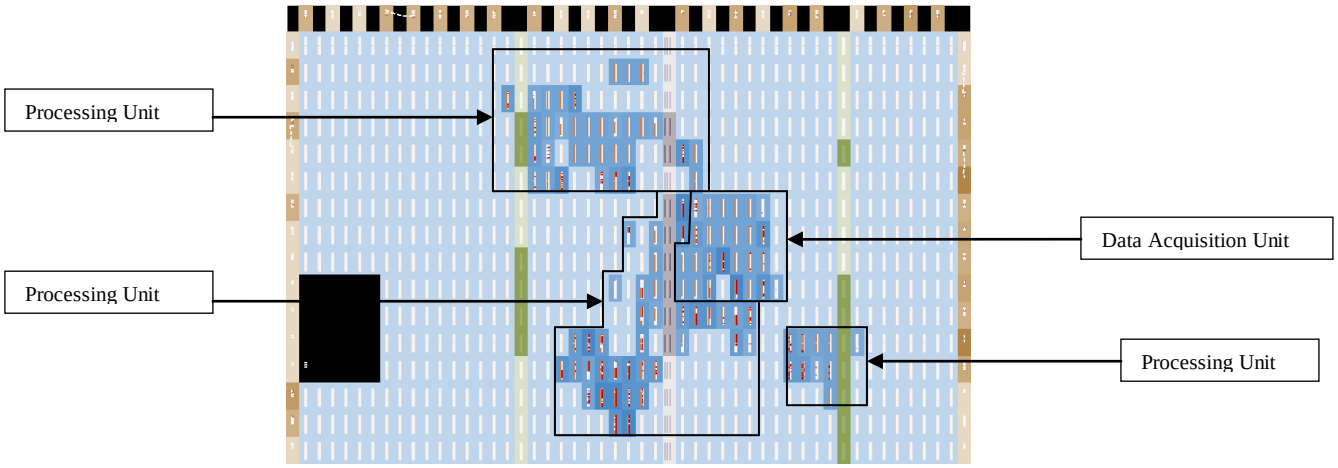


Figure 99 Original Chip Floor Planning Result

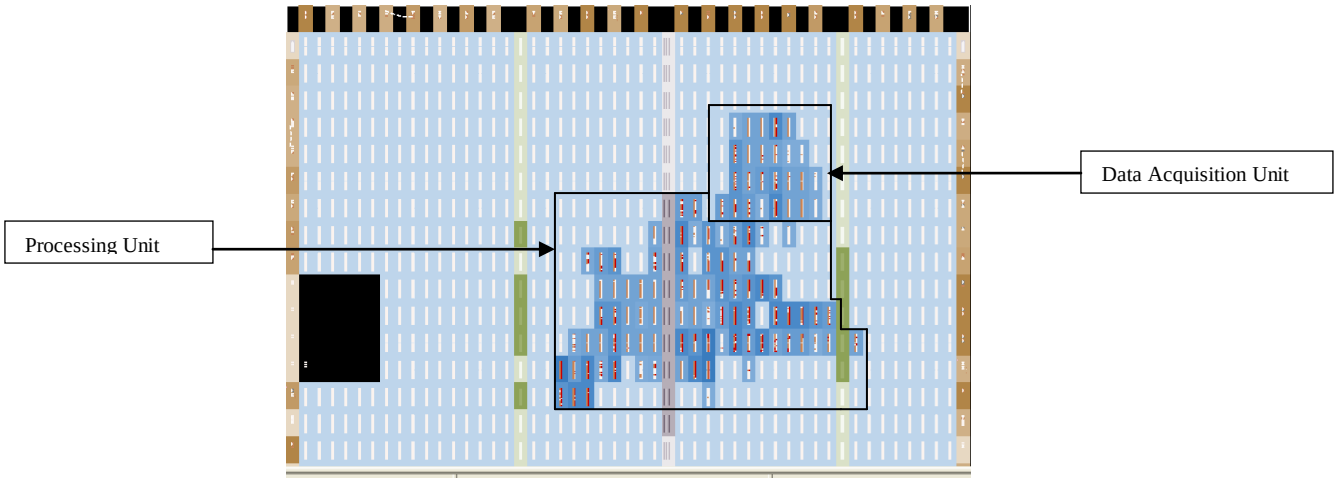


Figure 100 Optimized Chip Floor Planning Result

Chapter 7 Discussion and Conclusion

7.1 Conclusion

This research project has successfully created a complete system on programmable chip (FPGA) solution customized for a touch screen system with a real time proven performance. There are three significant outcomes achieved in this research: first, having established an on-chip hardware based touch screen acquisition engine with a more efficient control mechanism, higher speed and lower noise level compared with the existing general microcontroller based acquisition system; second, having constructed all basic touch screen functionalities into on-chip hardware blocks and realized a basic real-time operational touch screen; and third, having optimized system resource consumption to a substantial extent on different levels to lower manufacture cost, without compromising overall system performance.

Instead of a general purpose microcontroller structure, a customized acquisition engine has been built in this research with more efficient resource usage, a more flexible design mechanism and improved performance. All logic structures and components in the customized acquisition system are indispensable, compared with the general microcontroller structure where some of the structure resources are redundant in the acquisition operation. Thus, the tailored on-chip acquisition design is more efficient in resource consumption from a system point of view. By taking advantage of the on-chip pipeline structure, a new concurrent acquisition mechanism has been developed in this research, with a pipelined configuration and transmission capability for controlling a two image sensor based acquisition system. Testing has proved that the acquisition speed is approximately twice as fast on the concurrent acquisition mechanism compared with the sequential mechanism.

Furthermore, the acquisition engine architecture has been developed in a modular and self-contained manner, bearing in mind future expansion where more image sensors can be used by the same control mechanism for a multi-touch system. Without the system limitation of a fixed structure microcontroller, the programmable chip based acquisition design has the flexibility required for meeting newly emerging and updated requirements. In the customized acquisition

design, the data clock has been accelerated up to 1 MHz which is three times faster than the existing system and a high performance conversion chip with a low drift low noise reference voltage has been added to improve digitization accuracy to 12 bits and reduce the noise level by approximate 50% from the previous microcontroller based system. The design result has fully satisfied the first design objective where a general-purpose acquisition structure has been replaced with a customized concurrent acquisition engine with faster execution speed, more accurate digitization, lower signal noise and robust capability in optical imaging system expansion.

After establishing an on-chip data acquisition system which has better performance validated by test results, the touch screen processing modules have been constructed incrementally in this research, aiming to achieve a complete on-chip real time touch screen solution. The design started from the memory management unit (MMU) which stores reference frames and derived trigger levels. Then the edge localization unit (ELU) was constructed, where two algorithms were implemented and evaluated with the application specific dynamic linear approximation edge detection method being selected. This method was found to have less latency and was a more resource-efficient solution and has the capability of more accurate sub-pixel edge localization. Then, after consideration of system resources, the computationally expensive operations (tangent, for instance) have been pre-calculated in system look up tables, for converting the edge location to the tangent of the angle before position localization.

In touch screen system design, the position localization unit (PLU) is the core unit, localizing the touch x and y coordinates based on all previous processing results. In this research, an original position localization structure has been created and demonstrated with a clear and compact transformation process from the application requirement to a resource-efficiency oriented hardware block. Most important, for the first time a hardware binary search engine based localization approach has been applied to the touch screen application with a significant acceleration compared with the existing sequential localization method. Meanwhile, a normaliser based touch motion tracking system has been implemented and executed concurrently with the aforementioned localization processing on a separate data path. During tested this has shown the capability of recording subtle motion variations, which is critical in pre-touch detection. Finally,

a master controller has been designed to integrate and synchronize all processing modules to achieve the complete touch screen functionality with a transmission engine and corresponding PC interface demonstrating the real time operation performance. The resulting on-chip touch screen processing unit is the first and original all-functionalities-in-one-chip system prototype and is unique in the optical imaging based touch screen industry, with a number of extra features (such as support for pre-touch recognition) customized for real world design requirements. The real time testing has successfully demonstrated the processing design output and made the results visible.

A significant effort has been put into system optimization, after proving the correct functionality of the on-chip touch screen in real time. Numerous optimization techniques have been applied to the system on different layers with different focuses. By reducing the full range pixel frame to active range frame, 32% of memory space has been saved compared with the original version. The logic optimization process was more complicated since the system architecture design, module and sub-module interconnection design have a critical impact on overall system resource consumption. A large number of system modules and sub-modules have been manually restructured using algorithm alteration, pipeline rolling, resource sharing, iterative approach, Timing-Division-Multiplexing and register balancing techniques. An approximate 40% improvement in logic consumption has been achieved without affecting overall system performance, in consideration of resource and manufacturing cost. Physical chip floor planning has been implemented resulting in a more efficient and higher throughput.

In conclusion, this research has investigated, implemented and validated a complete functional real time touch screen system on-chip solution with a customized high performance touch imaging acquisition engine, and a fully capable basic touch localization and motion tracking system, with an economic amount of overall resource consumption.

7.2 Future Work

This investigative project has achieved significant research outcomes with an advanced multi-level solution satisfying and exceeding the specified requirements and expectations from different perspectives. Nonetheless, there are a number of developments that could be implemented in future for further improvement.

First and foremost, a greater optical imaging coverage is considered to be an important factor for supporting multiple touches, as specified by the latest industry requirements. The concurrent acquisition mechanism inside the system solution has been designed based on two image sensors, but with the capability to be adapted to accommodate more sensors without changing the main structure. In future development, such as a six-sensor touch screen as illustrated below right, more dedicated counters and registers are required in the acquisition block with modified module timing flow and increased imaging bandwidth.

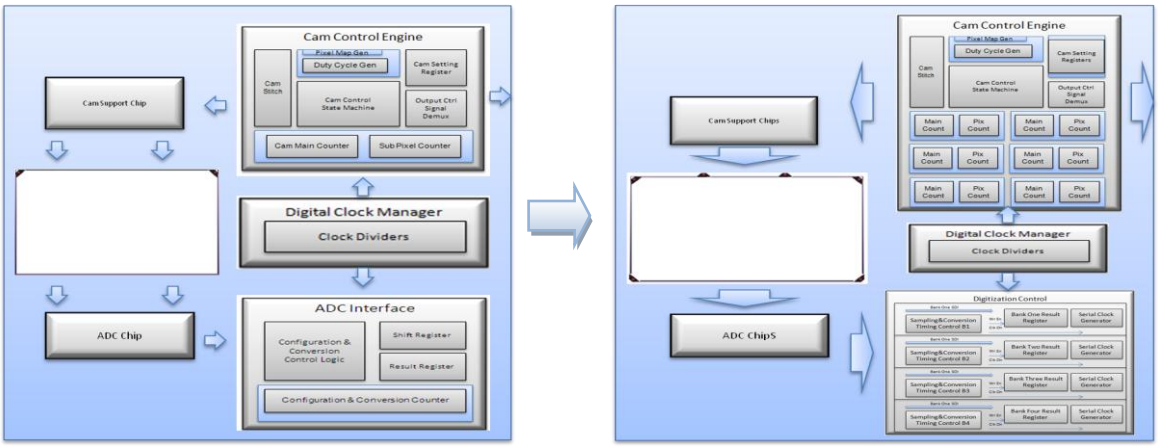


Figure 101 More optical imaging coverage

The increase in optical imaging bandwidth leads to higher requirements for subsequent touch information processing ability. And the potential demand for more user-touch recognition in the future further increases the complexity of the processing unit. As well as considering the conventional approach of sharing post-processing on the PC side, it is planned to use an embedded solution with the addition of an on-chip compact processor core that is independent of the PC operating system.

The established processing unit inside the on-chip solution has aimed at integrating all elementary and fundamental touch related functionalies on-the-fly with the acquisition process. With the increasing processing pressure, it is not efficient to further customize dedicated logic in a dynamic enviroment for potential flexible requirements in the future. Based on the existing work, a simple soft processor core can be added to handle relatively complicated and uncommon algorithms and features. Most existing modules will still play a critical role in flow acceleration and basic function processing with a foreseeable path to improve and optimize module-to-module and module-to-processor core interconnections and partitions. The proven system functionalities are able to be reorganized and reformed as new IPs for reuse for further development purposes. A potential soft processor core for the system is illustrated below:

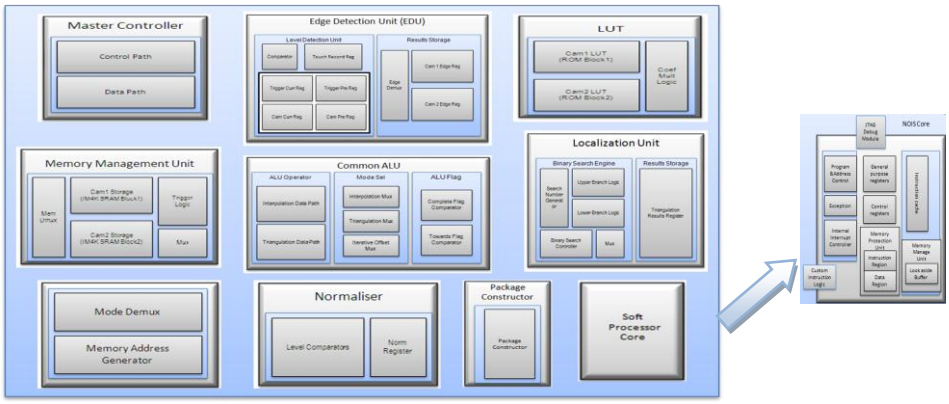


Figure 102 More robust processing handling

Finally, from a manufacturing cost perspective, the on-chip solution in this research has been implemented in a Field Programmable Gate Array with a minimum number of outside chip components and circuits. For high volume production, it is expected to achieve higher integration density on the chip level. Compared with standard-cell solutions, the Hardcopy ASIC approach is preferred in the future, with a seamless migration capability based on the existing design from this research, because of its low cost and time-to-market pressure.

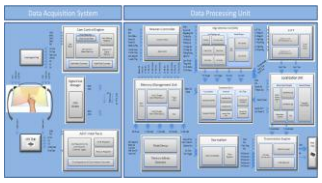


Figure 103 SoC Design

Figure 104 FPGA Chip

Figure 105 Migrated ASIC Chip

References

[1] L. Li-Yi, *et al.*, "Incremental physical design method for flat SOC design," in *VLSI Design, Automation and Test, 2009. VLSI-DAT '09. International Symposium on*, 2009, pp. 351-354.

[2] H. Yoshida, *et al.*, "Demonstration of hardware-accelerated formal verification," in *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, 2009, pp. 380-383.

[3] M. D. Edwards and J. Forrest, "Software acceleration using programmable hardware devices," *Computers and Digital Techniques, IEE Proceedings* -, vol. 143, pp. 55-63, 1996.

[4] D. Andrews, *et al.*, "Programming models for hybrid CPU/FPGA chips," *Computer*, vol. 37, pp. 118-120, 2004.

[5] F. Garufi, *et al.*, "A hybrid modular control and acquisition system," in *Real-Time Conference, 2007 15th IEEE-NPSS*, 2007, pp. 1-5.

[6] Y. Pengfei, *et al.*, "ARM and FPGA based electrical signal source for swing scanning infrared earth sensor," in *Image and Signal Processing (CISP), 2010 3rd International Congress on*, 2010, pp. 2742-2746.

[7] N. Alt, *et al.*, "Hardware/software architecture of an algorithm for vision-based real-time vehicle detection in dark environments," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008, pp. 176-181.

[8] X. Dang, *et al.*, "Hardware acceleration for motion tracking system used in image-guided surgery," in *Biomedical Engineering and Informatics (BMEI), 2010 3rd International Conference on*, pp. 1498-1502.

[9] G. W. Morris, *et al.*, "FPGA Accelerated Low-Latency Market Data Feed Processing," in *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*, 2009, pp. 83-89.

[10] B. Madhusudan and J. W. Lockwood, "A hardware-accelerated system for real-time worm detection," *Micro, IEEE*, vol. 25, pp. 60-69, 2005.

[11] O. Dandekar and R. Shekhar, "FPGA-Accelerated Deformable Image Registration for Improved Target-Delineation During CT-Guided Interventions," *Biomedical Circuits and Systems, IEEE Transactions on*, vol. 1, pp. 116-127, 2007.

[12] Y. Kaeriyama, *et al.*, "Ray Tracing Hardware System Using Plane-Sphere Intersections," in *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, 2006, pp. 1-6.

[13] C. R. Castro-Pareja, *et al.*, "FAIR: a hardware architecture for real-time 3-D image registration," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 7, pp. 426-434, 2003.

[14] J. Majeed, "Hardware based Security for Wireless Devices," presented at the 3rd International Conference: Sciences of Electronic,Technologies of Information and Telecommunications, 2005.

[15] V. Pejovic, *et al.*, "Adding Value to TCP/IP Based Information exchange Security by Specialized Hardware," in *Emerging Security Information, Systems, and Technologies, 2007. SecureWare 2007. The International Conference on*, 2007, pp. 145-150.

[16] W. Wolf, "A decade of hardware/software codesign," *Computer*, vol. 36, pp. 38-43, 2003.
[17] N. S. Voros, *et al.*, "Hardware/Software Co-Design of Complex Embedded Systems: An Approach Using Efficient Process Models, Multiple Formalism Specification and Validation via Co-Simulation," *Design Automation for Embedded Systems*, vol. 8, pp. 5-49, 2003.
[18] N. Nedjah and L. D. M. Mourelle, *Co-design for System Acceleration A Quantitative Approach*: Springer Netherlands, 2007.
[19] H. K. H. So, *et al.*, "A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH," in *Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference*, 2006, pp. 259-264.
[20] T. B. Ismail and A. A. Jerraya, "Design models and steps for codesign," in *Verification of Hardware Software Codesign, IEE Colloquium on*, 1995, pp. 1/1-1/2.
[21] M. K. Purvis and D. W. Franke, "An overview of hardware/software codesign," in *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, 1992, pp. 2665-2668 vol.6.
[22] N. Sudha and A. R. Mohan, "Hardware-Efficient Image-Based Robotic Path Planning in a Dynamic Environment and Its FPGA Implementation," *Industrial Electronics, IEEE Transactions on*, vol. 58, pp. 1907-1920.
[23] A. Laureshyn, *et al.*, "Application of automated video analysis for behavioural studies: concept and experience," *Intelligent Transport Systems, IET*, vol. 3, pp. 345-357, 2009.
[24] N. Sawasaki, *et al.*, "Embedded vision system for mobile robot navigation," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 2006, pp. 2693-2698.
[25] M. Gokhale, *et al.*, "Image Processing," in *Reconfigurable Computing*, ed: Springer US, 2005, pp. 119-139.
[26] M. S. Hromalik, *et al.*, "Data acquisition and control for the LCLS pixel array detector," in *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE*, 2007, pp. 1744-1750.
[27] X. Hui, *et al.*, "Acquisition board design of high-speed image data based on ARM and FPGA," in *Computer Design and Applications (ICCDA), 2010 International Conference on*, pp. V1-374-V1-376.
[28] J. Yun Ho, *et al.*, "Design of real-time image enhancement preprocessor for CMOS image sensor," *Consumer Electronics, IEEE Transactions on*, vol. 46, pp. 68-75, 2000.
[29] D. Nguyen, *et al.*, "Real-time face detection and lip feature extraction using field-programmable gate arrays," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 36, pp. 902-912, 2006.
[30] A. R. Akoushideh and A. Shahbahrami, "Accelerating Texture Features Extraction Algorithms Using FPGA Architecture," in *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, pp. 232-237.
[31] B. Kisačanin, *et al.*, "Benchmarks of Low-Level Vision Algorithms for DSP, FPGA, and Mobile PC Processors," in *Embedded Computer Vision*, ed: Springer London, 2009, pp. 101-120.

[32] P. Dillinger, *et al.*, "FPGA-Based Real-Time Image Segmentation for Medical Systems and Data Processing," *Nuclear Science, IEEE Transactions on*, vol. 53, pp. 2097-2101, 2006.
[33] N. Aaraj, *et al.*, "Hybrid Architectures for Efficient and Secure Face Authentication in Embedded Systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 15, pp. 296-308, 2007.
[34] H. Pei-Yung, *et al.*, "A Portable Vision-Based Real-Time Lane Departure Warning System: Day and Night," *Vehicular Technology, IEEE Transactions on*, vol. 58, pp. 2089-2094, 2009.
[35] V. Bonato, *et al.*, "A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 18, pp. 1703-1712, 2008.
[36] A. G. Vicente, *et al.*, "Embedded Vision Modules for Tracking and Counting People," *Instrumentation and Measurement, IEEE Transactions on*, vol. 58, pp. 3004-3011, 2009.
[37] I. Ishii, *et al.*, "Development of high-speed and real-time vision platform, H³ vision," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, pp. 3671-3678.
[38] H. Yean Choon and S. Yu, "Developing a smart camera for gesture recognition in HCI applications," in *Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on*, 2009, pp. 994-998.
[39] K. Ohtani, *et al.*, "An intelligent position sensor for light spots using an analog scan circuit and fpga," in *Instrumentation and Measurement Technology Conference, 2003. IMTC '03. Proceedings of the 20th IEEE*, 2003, pp. 711-715.
[40] S. Hussmann and T. H. Ho, "A high-speed subpixel edge detector implementation inside a FPGA," *Real-Time Imaging*, vol. 9, pp. 361-368, 2003.
[41] A. Braeken, *et al.*, "Secure FPGA technologies and techniques," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009, pp. 560-563.
[42] Altera. (2007). *Cyclone II Architecture*. Available: http://www.altera.com/literature/hb/cyc2/cyc2_cii51002.pdf
[43] Actel. (2008). *Designing for Performance on Flash-Based FPGAs*. Available: http://www.actel.com/documents/Design_Performance_AN.pdf
[44] H. Selvaraj, *et al.*, "Partitioning of large HDL ASIC designs into multiple FPGA devices for prototyping and verification," in *Computational Intelligence and Multimedia Applications, 2001. ICCIMA 2001. Proceedings. Fourth International Conference on*, 2001, pp. 411-415.
[45] W. Hinrichs, *et al.*, "A 1.3-GOPS parallel DSP for high-performance image-processing applications," *Solid-State Circuits, IEEE Journal of*, vol. 35, pp. 946-952, 2000.
[46] S. Asano, *et al.*, "Performance comparison of FPGA, GPU and CPU in image processing," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, 2009, pp. 126-131.
[47] S. Anvar, *et al.*, "FPGA-based system-on-chip designs for real-time applications in particle physics," in *Real Time Conference, 2005. 14th IEEE-NPSS*, 2005, p. 5 pp.
[48] X. Zhihui, *et al.*, "A platform-based SoC hardware/software co-design environment," in *Computer Supported Cooperative Work in Design, 2004. Proceedings. The 8th International Conference on*, 2004, pp. 443-448 Vol.2.

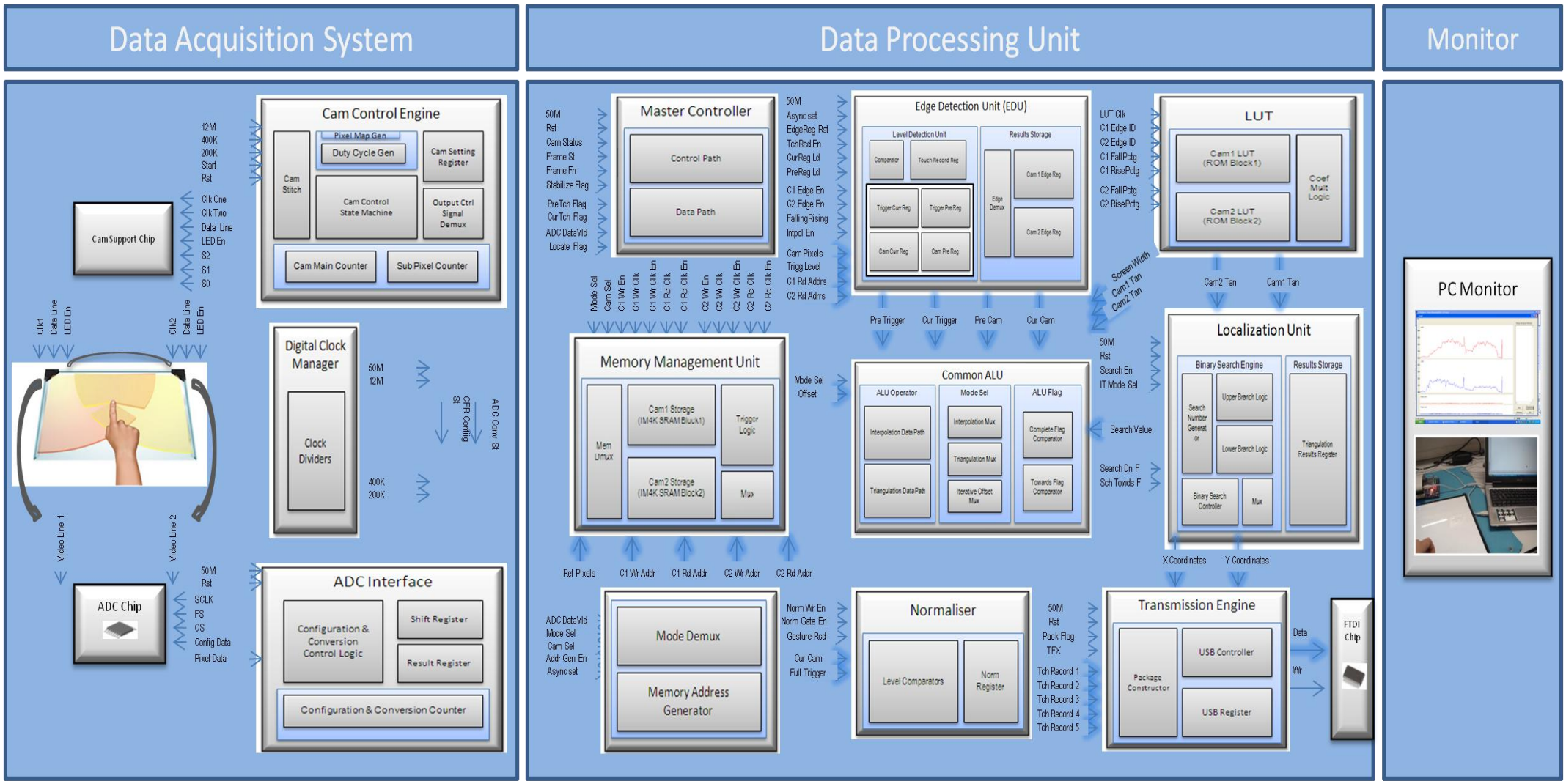
[49] L. Cheng-Hong and L. P. Carloni, "Leveraging Local Intracore Information to Increase Global Performance in Block-Based Design of Systems-on-Chip," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, pp. 165-178, 2009.
[50] C. Soo Ho and K. Soo Dong, "Reuse-based Methodology in Developing System-on-Chip (SoC)," in *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on*, 2006, pp. 125-131.
[51] B. Younghoon, *et al.*, "The Development of SoC Platform for Embedded System Applications," in *Convergence Information Technology, 2007. International Conference on*, 2007, pp. 2286-2291.
[52] L. Idkhajine, *et al.*, "Fully Integrated FPGA-Based Controller for Synchronous Motor Drive," *Industrial Electronics, IEEE Transactions on*, vol. 56, pp. 4006-4017, 2009.
[53] K. Jeonghun, *et al.*, "Surveillance camera SOC architecture using one-bit motion detection for portable applications," in *SOC Conference, 2007 IEEE International*, 2007, pp. 71-74.
[54] O. Eun Tack, *et al.*, "An Embedded SoC System to Trace Moving Object and Detect Distance," in *Natural Computation, 2008. ICNC '08. Fourth International Conference on*, 2008, pp. 379-383.
[55] H. Chun, *et al.*, "A novel SoC architecture on FPGA for ultra fast face detection," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, 2009, pp. 412-418.
[56] TexasInstruments. (2009). *Low-Power, 12-Bit, 1-MHz, Single/Dual Unipolar Input, ADCs with Serial Interface*. Available: http://focus.ti.com/lit/ds/symlink/ads7229.pdf
[57] TexasInstruments. (2010). *Low Noise, Very Low Drift, Precision Voltage Reference*. Available: http://focus.ti.com/lit/ds/symlink/ref5025.pdf
[58] FTDIChip. (2005). *FT245R USB FIFO I.C.* Available: http://www.asix.cz/download/ftdi/ft2xxr/ds_ft245r_v102.pdf
[59] Altera. (2010). *SCFIFO and DCFIFO Megafunctions*. Available: http://www.altera.com/literature/ug/ug_fifo.pdf
[60] Altera. (2008). *Cyclone II Memory Blocks*. Available: http://www.altera.com/literature/hb/cyc2/cyc2_cii51008.pdf
[61] I. Yasri, *et al.*, "An FPGA Implementation of Gradient Based Edge Detection Algorithm Design," in *Computer Technology and Development, 2009. ICCTD '09. International Conference on*, 2009, pp. 165-169.
[62] S. KILTS, "Architecting Area," in *Advanced FPGA Design: Architecture, Implementation, and Optimization*, ed: JOHN WILEY&SONS,INC,, 2007, p. 18.
[63] S. M. Qasim, *et al.*, "A review of FPGA-based design methodology and optimization techniques for efficient hardware realization of computation intensive algorithms," in *Multimedia, Signal Processing and Communication Technologies, 2009. IMPACT '09. International*, 2009, pp. 313-316.
[64] S. Mondal and S. O. Memik, "Resource sharing in pipelined CDFG synthesis," in *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific*, 2005, pp. 795-798 Vol. 2.
[65] S. KILTS, "Synthesis Optimization," in *Advanced FPGA Design: Architecture, Implementation, and Optimization*, ed: JOHN WILEY&SONS,INC,, 2007, p. 208.

[66] C. Maxfield, "Simulation, Synthesis, Verification, etc. Design Tools," in *The Design Warrior's Guide to FPGAs*, ed: Elsevier, 2004, p. 312.
[67] J. Cong and W. Chang, "Optimal FPGA mapping and retiming with efficient initial state computation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 18, pp. 1595-1607, 1999.
[68] Altera. (2007). *Netlist optimization and Physical Synthesis*. Available: http://www.altera.com/literature/hb/qts/qts_qii52007.pdf
[69] Altera. (2007). *FPGA Performance Benchmarking Methodology*. Available: http://www.altera.com/literature/wp/wpfpgapbm.pdf
[70] P. Metzgen and D. Nancekievill, "Multiplexer restructuring for FPGA implementation cost reduction," in *Design Automation Conference, 2005. Proceedings. 42nd*, 2005, pp. 421-426.
[71] Y. Liu, *et al.*, "Research on the problems of satellite borne FPGA based finite state machine," in *Systems and Control in Aerospace and Astronautics, 2008. ISSCAA 2008. 2nd International Symposium on*, 2008, pp. 1-4.
[72] Synopsys. (2003). *Synplify & Quartus II Design Methodology*. Available: http://www.altera.ru/Disks/Altera%20Documentation%20Library/literature/an/an226.pdf
[73] Altera. (2003). *Using the Quartus II Chip Editor*. Available: http://www.altera.co.jp/literature/an/an310.pdf
[74] Altera. (2010). *CycloneIII FPGA Power Consumption*. Available: http://www.altera.com/devices/fpga/cyclone3/overview/power/cy3-power.html#cyclone_power
[75] Altera. (2010). *PowerPlay Power Analysis*. Available: http://www.altera.com/literature/hb/qts/qts_qii53013.pdf?GSA_pos=2&WT.oss_r=1&WT.oss=powerplay

Appendices

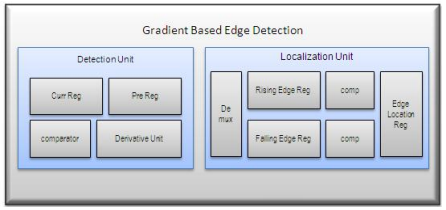
Because of confidential issue, a restricted amount of design files and source codes are selectively attached below:

Touch Screen on Chip (TOC) Architectural Level Block Diagram:



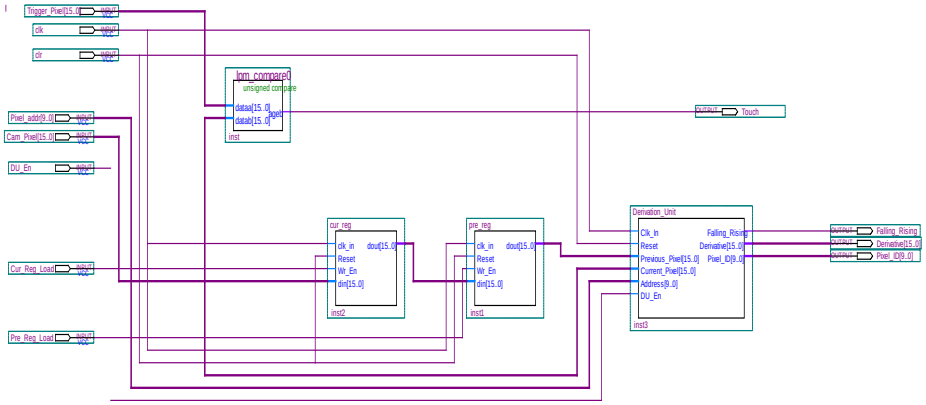
The following is the top layer schematic of Gradient Based Edge Detection Unit (EDU Method 1):

Block diagram: ⇨

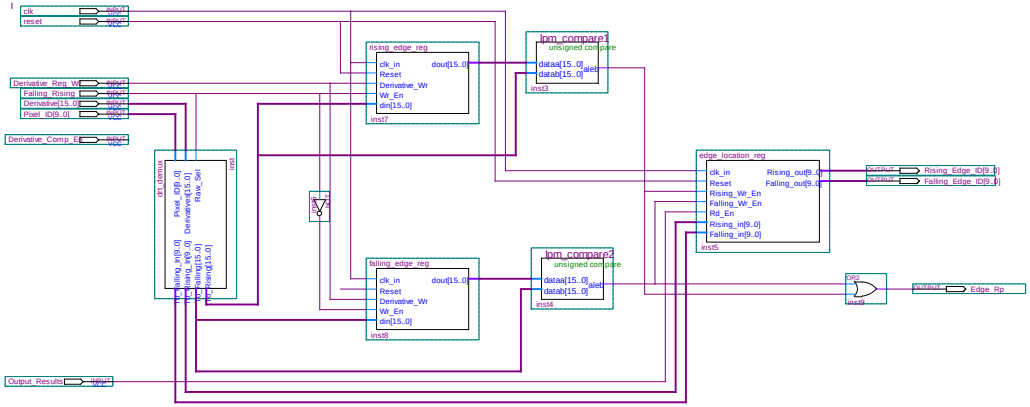


Actual implementation of EDU Method One on Register-Transfer-Level (RTL):

Detection Unit:

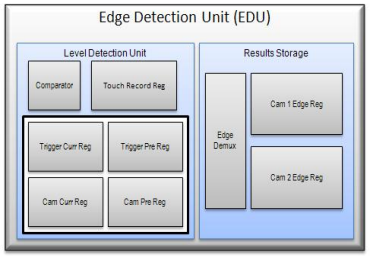


Localization Unit:

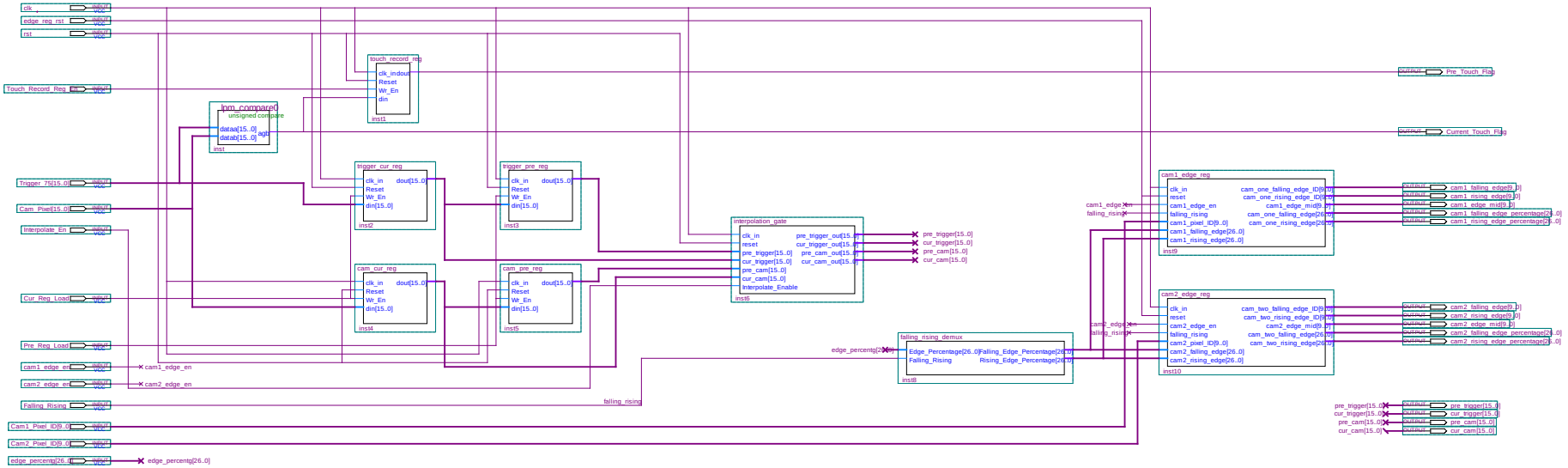


The following is the top layer schematic of Dynamic Linear Approximation Based Edge Detection Unit (EDU Method 2):

Block diagram: ⇨

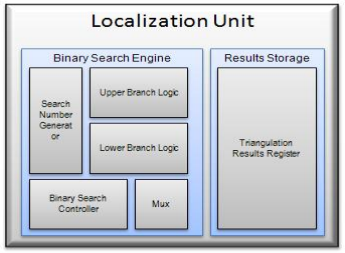


Actual implementation of EDU Method Two on Register-Transfer-Level (RTL):

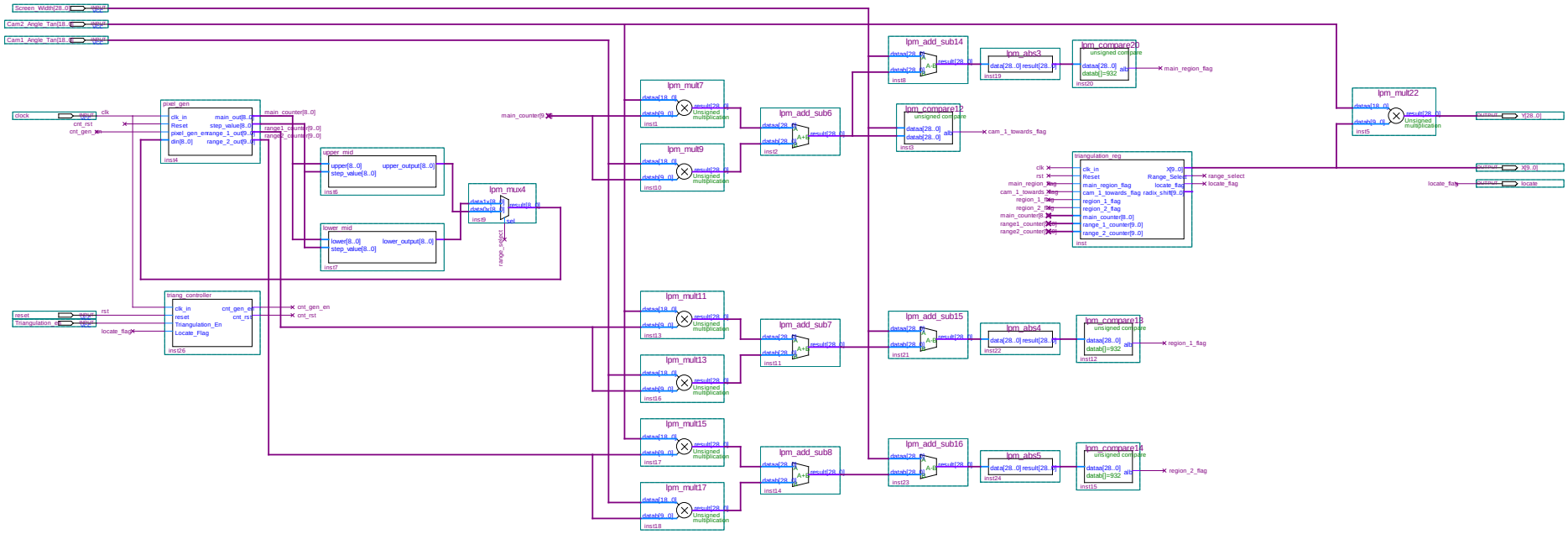


The following is the Majority of top layer schematic of Position Localization Unit (PLU):

Block diagram: ⇨



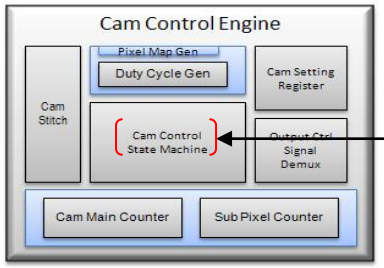
Actual implementation of PLU on Register-Transfer-Level (RTL):



All the other system module RTL schematics and the corresponding VHDL codes are not presented as they are confidential.

There are three major design program sources (code): one is VHDL code for all major components and sub-components (with corresponding test bench simulation code as well); another is Matlab code for offline system performance analysis (noise analysis) and look-up-table (LUT) pre-calculation; the last is Borland C++ based user interface code for real-time demonstration (which can be seen in the demonstration videos).

Most source code will not be attached however selected VHDL code is shown as an example: this is the timing control code inside the Camera Control Engine which is a sub-module of the Data Acquisition Unit.



bel The hardware description code for the camera timing control sub-component is attached

```
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.numeric_std.all;
5  use work.pkg_constants.all;
6  --1/12 us is assumed as minimum interval received from outside clock
7  entity P9_Timing_Ctrl is
8                 port(
9                       clk_in_12M                      : in std_logic;
10                      clk_in_400K                     : in std_logic;
11                      clk_in_200K                     : in std_logic;
12                      reset                           : in std_logic;
13                      start                           : in std_logic;
14                      Thk_clk_sync                    : out std_logic;
15                      Fhk_clk_sync                    : out std_logic;
16                      sync_clock_one                  : out std_logic;
17                      sync_clock_two                  : out std_logic;
18                      data_video_clock                : out std_logic;
19                      LED_En                          : out std_logic;
20                      S2                              : out std_logic;
21                      S1                              : out std_logic;
22                      S0                              : out std_logic;
23                      CFR_Config                      : out std_logic;
24                      ADC_Start                       : out std_logic;
25                      CAM_Status                      : out std_logic_vector(1 downto 0);
26                      CAM1_Frame_St                   : out std_logic;
27                      CAM1_Frame_Fn                   : out std_logic;
28                      CAM2_Frame_St                   : out std_logic;
29                      CAM2_Frame_Fn                   : out std_logic;
30                      Stablization_Indicator          : out std_logic
31                     );
32 end P9_Timing_Ctrl;
33
34 architecture behavioral of P9_Timing_Ctrl is
35                type state_list is (Idle, CAM_Int_Tsu, CAM_Int_Clk_Sync, CAM_Int_Mode_Sel, CAM_Int_Pixel_Map_Tsu,
36                                    CAM_Int_Pixel_Map_Config, CAM_Int_Volt_Curr_Setup, CAM_Int_Reset_To_Idle,
37                                    CAM_Int_Idle_1, CAM_Int_Idle_2,CAM_Operation_Tsu, CAM_Operation_Clk_Sync,
38                                    CAM_Operation_Mode_Sel_Tsu, CAM_Operation_Mode_Sel, CAM_Operation_LED_En,
39                                    CAM_Operation_Exposure, CAM_Operation_Exposure_CFR_Config, CAM_Operation_E2,
40                                    CAM_Operation_E2_Idle, CAM_One_Transmission, CAM_One_Transmission_Idle,
41                                    CAM_Two_Transmission, CAM_Two_Transmission_Idle, CAM_Setup_Switch, CAM_Tx_Switch );
```

```
43              signal current_state, next_state : state_list;
44
45              --signal Pixel_Map_Waveform_clock_on : std_logic;
46              --signal TwoHdrK_clock_count : integer range 0 to 30;
47              --signal FourHdrK_clock_count : integer range 0 to 15;
48              --signal FiveHdrK_clock_count : integer range 0 to 25;
49              --signal Op_clock_count : integer range 0 to 30;
50              --signal Pixel_Map_Setup_count : integer range 0 to 19;
51
52              -- 200K clock sync signal
53
54
55              -- Pixel Map generator signals
56              signal Ctrl_Generator_Reset             : std_logic;
57              signal Ctrl_Pixel_Map_Waveform_clock_on : std_logic;
58              signal Ctrl_Pixel_Map_Waveform_clock    : std_logic;
59
60              -- System counter signals
61              signal Ctrl_Sys_Cnt_Rst                 : std_logic;
62              signal Ctrl_Sys_Cnt_En                  : std_logic;
63              signal Ctrl_Sys_Cnt_Out                 : std_logic_vector(13 downto 0);
64
65              -- Sub Pixel counter signals
66              signal Ctrl_Sub_Pixel_Cnt_Rst           : std_logic;
67              signal Ctrl_Sub_Pixel_Cnt_En            : std_logic;
68              signal Ctrl_Sub_Pixel_Cnt_Out           : std_logic_vector(4 downto 0);
69
70              -- Cam switch counter signals
71              signal Ctrl_Switch_Cnt_Rst              : std_logic;
72              signal Ctrl_Switch_Cnt_En               : std_logic;
73              signal Ctrl_Switch_Cnt_Flag             : std_logic;
74
75              -- Voltage and current shift register signals
76              signal Ctrl_Int_Shift_Reg_Ld            : std_logic;
77              signal Ctrl_Int_Shift_Reg_Default       : std_logic_vector(7 downto 0);
78              signal Ctrl_Int_Shift_Reg_En            : std_logic;
79              signal Ctrl_Int_Shift_Reg_Out           : std_logic;
80
81              -- Cam clock demux signals
82              signal Ctrl_Sync_Clk                   : std_logic;
83              signal Ctrl_Sync_Clk_Sel               : std_logic;
84
85              -- LED control signal
86              -- signal LED_En                          : std_logic;
87
88
89
90  -- ********************** COMPONENT DECLARATIONS **********************************
91
92  -- Component: Pixel Map Generator
93  component Pixel_Map_Generator
94  port(
95
96          FourHdrK_clock                  : in STD_LOGIC;
97          Generator_reset                 : in STD_LOGIC;
98          Pixel_Map_Waveform_clock_on     : in STD_LOGIC;
99          Pixel_Map_Waveform_clock        : out STD_LOGIC
100
101     );
102 end component;
103
104 -- Component: System Counter (Data and Video Cycle)
105 component System_Counter
106 port(
107
108         clk                             : in STD_LOGIC;
109         reset                           : in STD_LOGIC;
110         cnt_en                          : in STD_LOGIC;
111         qout                            : out std_logic_vector(13 downto 0)
112
113     );
114 end component;
```

```
116   -- Component: Cam Switch Counter
117   component Cam_Switch_Counter
118   port(
119
120           clk                                     : in STD_LOGIC;
121           reset                                   : in STD_LOGIC;
122           qout                                    : out STD_LOGIC
123
124       );
125   end component;
126
127   -- Component: Sub Pixel Counter
128   component Sub_Pixel_Counter
129   port(
130
131           clk                                     : in STD_LOGIC;
132           reset                                   : in STD_LOGIC;
133           cnt_en                                  : in STD_LOGIC;
134           qout                                    : out std_logic_vector(4 downto 0)
135
136       );
137   end component;
138
139   -- Component: Shift register for voltage and current setting (Data Cycle)
140   component SHIFT8
141   port(
142
143           clk                                     : in STD_LOGIC;
144           data_ld                                 : in STD_LOGIC;
145           data_in                                 : in STD_LOGIC_VECTOR (7 downto 0);
146           shift_en                                : in STD_LOGIC;
147           shift_out                               : out STD_LOGIC
148
149       );
150   end component;
151
152   -- Component: Sync clock demux
153   component Cam_Clk_Demux
154   port(
155
156           clk                                 : in STD_LOGIC;
157           sel                                 : in STD_LOGIC;
158           clk_1                               : out STD_LOGIC;
159           clk_2                               : out STD_LOGIC
160
161       );
162   end component;
163
164   begin
165
166   -- ************************** COMPONENT DEFINITIONS ****************************
167
168   -- Component: Pixel Map Generator (PMG)
169   -- Pixel Map Setup Waveform
170   -- 20% duty cycle, fed by 200k signal and start generating from position 8 out of 20
171   -- Giving exact waveform as is supposed to be
172   PMG: Pixel_Map_Generator
173       port map (
174                   FourHdrK_clock                => clk_in_400K,
175                   Generator_reset               => Ctrl_Generator_Reset,
176                   Pixel_Map_Waveform_clock_on   => Ctrl_Pixel_Map_Waveform_clock_on,
177                   Pixel_Map_Waveform_clock      => Ctrl_Pixel_Map_Waveform_clock
178                 );
179
180   -- Component: System Counter (SysCnt)
181   -- Either Initialization or Operation counting
182   SysCnt: System_Counter
183       port map (
184                   clk      => clk_in_12M,
185                   reset    => Ctrl_Sys_Cnt_Rst,
186                   cnt_en   => Ctrl_Sys_Cnt_En,
187                   qout     => Ctrl_Sys_Cnt_Out
188                 );
189
```

```
190   -- Component: Cam Switch Counter (CamSwCnt)
191   CamSwCnt: Cam_Switch_Counter
192       port map (
193                   clk      => Ctrl_Switch_Cnt_En,
194                   reset    => Ctrl_Switch_Cnt_Rst,
195                   qout     => Ctrl_Switch_Cnt_Flag
196                 );
197
198   -- Component: Sub Pixel Counter (SPCnt)
199   SPCnt: Sub_Pixel_Counter
200       port map (
201                   clk      => clk_in_12M,
202                   reset    => Ctrl_Sub_Pixel_Cnt_Rst,
203                   cnt_en   => Ctrl_Sub_Pixel_Cnt_En,
204                   qout     => Ctrl_Sub_Pixel_Cnt_Out
205                 );
206
207   -- Component: shift register (SHIFT8)
208   -- Voltage and current setting
209   Shifter: SHIFT8
210       port map (
211                   clk       => clk_in_200K,
212                   data_ld   => Ctrl_Int_Shift_Reg_Ld,
213                   data_in   => Ctrl_Int_Shift_Reg_Default,
214                   shift_en  => Ctrl_Int_Shift_Reg_En,
215                   shift_out => Ctrl_Int_Shift_Reg_Out
216                 );
217
218   -- Component: Cam clock demux (Demux)
219   Demux: Cam_Clk_Demux
220       port map (
221                   clk      => Ctrl_Sync_Clk,
222                   sel      => Ctrl_Sync_Clk_Sel,
223                   clk_1    => sync_clock_one,
224                   clk_2    => sync_clock_two
225                 );
231   -------------------- State Machine --------------------
232   process(clk_in_12M, reset, start)
233     |
234   --variable CAM_setup_count : integer RANGE 0 TO 10;
235   --variable CAM_Tx_count : integer RANGE 0 TO 10;
236   --variable samp_one_count : integer RANGE 0 TO 1000;
237   --variable samp_two_count : integer RANGE 0 TO 1000;
238   BEGIN
239                 IF reset = '0' THEN
240
241                    Thk_clk_sync <= '0';
242                    Fhk_clk_sync <= '0';
243                    --
244                    Ctrl_Generator_Reset <= '0';
245                    Ctrl_Pixel_Map_Waveform_clock_on <= '0';
246                    --
247                    Ctrl_Sys_Cnt_Rst <= '0';
248                    Ctrl_Sys_Cnt_En  <= '0';
249                    --
250                    Ctrl_Sub_Pixel_Cnt_Rst <= '0';
251                    Ctrl_Sub_Pixel_Cnt_En  <= '0';
252                    --
253                    Ctrl_Switch_Cnt_Rst <= '0';
254                    Ctrl_Switch_Cnt_En  <= '0';
255                    --
256                    Ctrl_Int_Shift_Reg_Ld <= '1';
257                    Ctrl_Int_Shift_Reg_En <= '0';
```

```
258                     Ctrl_Int_Shift_Reg_Default <= "001100"&"00";
259                     --
260                     Ctrl_Sync_Clk <= '0';
261                     Ctrl_Sync_Clk_Sel <= '0';
262                     --
263                     data_video_clock <= '0';
264                     LED_En <= '0';
265                     S2 <= '0';
266                     S1 <= '0';
267                     S0 <= '0';
268                     CAM_Status <= "00";
269                     CFR_Config <= '0';
270                     ADC_Start <= '0';
271                     CAM1_Frame_St <= '0';
272                     CAM1_Frame_Fn <= '0';
273                     CAM2_Frame_St <= '0';
274                     CAM2_Frame_Fn <= '0';
275                     Stablization_Indicator <= '0';
276                     next_state <= Idle;
277
278                 ELSIF clk_in_12M'EVENT AND clk_in_12M='1' THEN
279
280                 CASE current_state IS
281
282                      WHEN Idle  =>
283                                   Thk_clk_sync <= '0';
284                                   Fhk_clk_sync <= '0';
285                                   --
286                                   Ctrl_Generator_Reset <='1';
287                                   Ctrl_Pixel_Map_Waveform_clock_on <='0';
288                                   --
289                                   Ctrl_Sys_Cnt_Rst <= '1';
290                                   Ctrl_Sys_Cnt_En  <= '0';
291                                   --
292                                   Ctrl_Sub_Pixel_Cnt_Rst <= '1';
293                                   Ctrl_Sub_Pixel_Cnt_En  <= '0';
294                                   --
295                                   Ctrl_Switch_Cnt_Rst <= '1';
296                                   Ctrl_Switch_Cnt_En  <= '0';
298                                   Ctrl_Int_Shift_Reg_Ld <= '0';
299                                   Ctrl_Int_Shift_Reg_En <= '0';
300                                   --
301                                   Ctrl_Sync_Clk <= '0';
302                                   Ctrl_Sync_Clk_Sel <= '0';
303                                   --
304                                   LED_En <= '0';
305                                   S2 <= '0';
306                                   S1 <= '0';
307                                   S0 <= '1';
308                                   CAM_Status <= "00";
309                                   data_video_clock <= '0';
310                                   CFR_Config <= '0';
311                                   ADC_Start <= '0';
312                                   CAM1_Frame_St <= '0';
313                                   CAM1_Frame_Fn <= '0';
314                                   CAM2_Frame_St <= '0';
315                                   CAM2_Frame_Fn <= '0';
316                                   Stablization_Indicator <= '0';
317
318                                   IF start = '1' THEN
319                                      next_state <= CAM_Setup_Switch;
320                                   ELSE
321                                      next_state <= Idle;
322                                   END IF;
323
324                     WHEN CAM_Int_Tsu =>
325
326  ------------------- 2.5us set up time before start signal (2.5us) ----------------
327
```

```
328                                     Ctrl_Sync_Clk <= CLOCK_HIGH;
329                                     data_video_clock <= CLOCK_HIGH;
330                                     Ctrl_Switch_Cnt_En <= CLOCK_LOW;
331                                     IF (Ctrl_Sys_Cnt_Out = INT_OPRN_TSU) THEN
332                                         next_state <= CAM_Int_Clk_Sync;
333                                     END IF;
334
335                         WHEN CAM_Int_Clk_Sync =>
336
337                                     -- sync 200k clock
338                                     Thk_clk_sync <= CLOCK_HIGH;
339                                     next_state <= CAM_Int_Mode_Sel;
340
341                         WHEN CAM_Int_Mode_Sel =>
342
343          ---------------------start signal and mode select (10us) ---------------------
344                                     Thk_clk_sync <= CLOCK_LOW;
345                                     Ctrl_Sync_Clk <= clk_in_200K;
346                                     data_video_clock <= CLOCK_HIGH;
347                                     IF Ctrl_Sys_Cnt_Out = SETUP_INT_MODE_SEL THEN
348                                        next_state <= CAM_Int_Pixel_Map_Tsu;
349                                     END IF;
350
351                         WHEN CAM_Int_Pixel_Map_Tsu =>
352
353          --------------- One instruction cycle to turn on Pixel map generator (0.083us) -------
354                                     Fhk_clk_sync <= CLOCK_HIGH;
355                                     Ctrl_Pixel_Map_Waveform_clock_on <= CLOCK_HIGH;
356                                     next_state <= CAM_Int_Pixel_Map_Config;
357
358                         WHEN CAM_Int_Pixel_Map_Config =>
359
360          ---------------------- PM 1 to PM 100 Setup (500us) -----------------------------
361
362                                     Fhk_clk_sync <= CLOCK_LOW;
363                                     Ctrl_Sync_Clk <= clk_in_200K;
364                                     data_video_clock <= Ctrl_Pixel_Map_Waveform_clock;
365                                     IF Ctrl_Sys_Cnt_Out > SETUP_INT_PMC THEN
366                                       Thk_clk_sync <= CLOCK_HIGH;
367                                       data_video_clock <= CLOCK_LOW;
368                                       Ctrl_Pixel_Map_Waveform_clock_on <= CLOCK_LOW;
369                                       next_state <= CAM_Int_Volt_Curr_Setup;
370                                    END IF;
371
372          --------------------  Voltage and Current Setup (40us) ---------------------------------
373                         WHEN CAM_Int_Volt_Curr_Setup =>
374
375                                     Thk_clk_sync <= CLOCK_LOW;
376                                     Ctrl_Sync_Clk <= clk_in_200K;
377                                     Ctrl_Int_Shift_Reg_En <= CLOCK_HIGH;
378                                     data_video_clock <= Ctrl_Int_Shift_Reg_Out;
379                                     IF Ctrl_Sys_Cnt_Out = SETUP_INT_VOLT_CURR THEN
380                                        next_state <= CAM_Int_Reset_To_Idle;
381                                     END IF;
382
383
384          --------------------  Reset to Idle (10us) ---------------------------------
385                         WHEN CAM_Int_Reset_To_Idle =>
386
387                                     Ctrl_Sync_Clk <= clk_in_200K;
388                                     data_video_clock <= CLOCK_LOW;
389                                     IF Ctrl_Sys_Cnt_Out > SETUP_INT_RST_TO_IDLE THEN
390                                        next_state <= CAM_Int_Idle_1;
391                                     END IF;
392
393          ---------------------    Idle  ---------------------------------------------
394                         WHEN CAM_Int_Idle_1 =>
395
396                                     Ctrl_Sync_Clk <= CLOCK_LOW;
397                                     data_video_clock <= CLOCK_LOW;
398                                     -- reset system counter to switch from Int to operation
```

```
399                                 Ctrl_Sys_Cnt_Rst <= '0';
400                                 Ctrl_Sys_Cnt_En  <= '0';
401                                 next_state <= CAM_Int_Idle_2;
402
403                     WHEN CAM_Int_Idle_2 =>
404
405                                 Ctrl_Sync_Clk <= CLOCK_LOW;
406                                 data_video_clock <= CLOCK_LOW;
407                                 Ctrl_Sys_Cnt_Rst <= '1';
408                                 Ctrl_Sys_Cnt_En  <= '0';
409                                 -- re-sync 200k clock before entry into Video cloc
410                                 Thk_clk_sync <= CLOCK_LOW;
411                                 next_state <= CAM_Operation_Tsu;
412
413   ---------------------- set up time before operation start-----------------
414                     WHEN CAM_Operation_Tsu =>
415
416                                 Ctrl_Sys_Cnt_En  <= CLOCK_HIGH;
417                                 Ctrl_Sync_Clk <= CLOCK_LOW;
418                                 data_video_clock <= CLOCK_HIGH;
419                                 IF Ctrl_Sys_Cnt_Out = INT_OPRN_TSU THEN
420                                    next_state <= CAM_Operation_Clk_Sync;
421                                 END IF;
422
423                     WHEN CAM_Operation_Clk_Sync =>
424
425                                 -- re-sync 200k clock
426                                 Thk_clk_sync <= CLOCK_HIGH;
427                                 next_state <= CAM_Operation_Mode_Sel;
428
429   ---------------------- set up time before mode select----------------------
430                     WHEN CAM_Operation_Mode_Sel_Tsu =>
431
432                                 Thk_clk_sync <= CLOCK_LOW;
433                                 Ctrl_Sync_Clk <= clk_in_200K;
434                                 data_video_clock <= CLOCK_HIGH;
435                                 IF Ctrl_Sys_Cnt_Out = OPRN_MODE_SEL_TSU THEN
436                                    next_state <= CAM_Operation_Mode_Sel;
437                                 END IF;
439   --------------------- mode select stage ---------------------------------
440                   WHEN CAM_Operation_Mode_Sel =>
441
442                                 Ctrl_Sync_Clk <= clk_in_200K;
443                                 data_video_clock <= CLOCK_LOW;
444                                 IF Ctrl_Sys_Cnt_Out = OPRN_MODE_SEL THEN
445                                    next_state <= CAM_Operation_LED_En;
446                                 END IF;
447
448   ---------------------LED enabled when shutter signal starts (7.5us)-----------
449                   WHEN CAM_Operation_LED_En =>
450
451                                 Ctrl_Sync_Clk <= clk_in_200K;
452                                 data_video_clock <= CLOCK_LOW;
453                                 LED_En <= CLOCK_HIGH;
454                                 IF Ctrl_Sys_Cnt_Out = OPRN_LED THEN
455                                    next_state <= CAM_Operation_Exposure;
456                                 END IF;
457
458   ---------------------Exposure Time (1000us)(Testing purpose 100us)----------------
459                   WHEN CAM_Operation_Exposure =>
460
461                                 Ctrl_Sync_Clk <= CLOCK_LOW;
462                                 IF Ctrl_Sys_Cnt_Out = OPRN_EXPOSURE THEN
463                                    next_state <= CAM_Operation_Exposure_CFR_Config;
464                                 END IF;
465
466   ---------------Start CFR Configuration while still exposuring(16 cycles at 25M = 9 cycles
467                   WHEN CAM_Operation_Exposure_CFR_Config =>
468
469                                 Ctrl_Sync_Clk <= CLOCK_LOW;
470                                 CFR_Config <= CLOCK_HIGH;
471                                 IF Ctrl_Sys_Cnt_Out = OPRN_EXPOSURE_CFR THEN
472                                    next_state <= CAM_Operation_E2;
473                                 END IF;
```

```
475   ---------------- stop CFR configuration and Turn off LED at the end of E2 -----------
476                     WHEN CAM_Operation_E2 =>
477
478                                   CFR_Config <= CLOCK_LOW;
479                                   LED_En <= CLOCK_LOW;
480   =                               IF Ctrl_Sys_Cnt_Out = OPRN_EXPOSURE_E2 THEN
481                                      next_state <= CAM_Operation_E2_Idle;
482                                   END IF;
483
484                     WHEN CAM_Operation_E2_Idle =>
485
486                                   CFR_Config <= CLOCK_LOW;
487                                   LED_En <= CLOCK_LOW;
488                                   -- reset system counter to switch from operation to transmission
489                                   Ctrl_Sys_Cnt_Rst <= '0';
490                                   Ctrl_Sys_Cnt_En  <= '0';
491                                   Ctrl_Sub_Pixel_Cnt_Rst <= '0';
492                                   Ctrl_Sub_Pixel_Cnt_En  <= '0';
493                                   next_state <= CAM_Tx_Switch;
494
495                      WHEN CAM_One_Transmission =>
496    ---------------------CAM One Transmission(528 cycles) -------------------------------------
497
498   =                               IF Ctrl_Sys_Cnt_Out >= TX_ST AND Ctrl_Sys_Cnt_Out <= TX_FN THEN
499
500                                            CAM_Status <= "01";
501   =                                        IF Ctrl_Sys_Cnt_Out >= TX_ST AND Ctrl_Sys_Cnt_Out <= TX_ST_FLAG THEN
502                                               CAM1_Frame_St <= '1';
503                                               CAM1_Frame_Fn <= '0';
504   =                                        ELSIF Ctrl_Sys_Cnt_Out >= TX_FN_FLAG_1 AND Ctrl_Sys_Cnt_Out <= TX_FN_FLAG_2 THEN
505                                               CAM1_Frame_Fn <= '1';
506                                               CAM1_Frame_St <= '0';
507   =                                        ELSE
508                                               CAM1_Frame_St <= '0';
509                                               CAM1_Frame_Fn <= '0';
510                                            END IF;
511
512   =                                        IF Ctrl_Sub_Pixel_Cnt_Out <= TX_SUB_PX_ST THEN
513                                               Ctrl_Sync_Clk <= '1';
514                                               ADC_Start <= '0';
515                                               next_state <= CAM_One_Transmission;
516
517   =                                        ELSIF Ctrl_Sub_Pixel_Cnt_Out > TX_SUB_PX_ST AND Ctrl_Sub_Pixel_Cnt_Out <= TX_SUB_PX_AD
518                                               Ctrl_Sync_Clk <= '0';
519                                               ADC_Start <= '1';
520                                               next_state <= CAM_One_Transmission;
521
522   =                                        ELSE
523                                               Ctrl_Sync_Clk <= '0';
524                                               ADC_Start <= '0';
525                                               next_state <= CAM_One_Transmission;
526                                            END IF;
527   =                                ELSE
528                                            Ctrl_Sys_Cnt_Rst <= '0';
529                                            Ctrl_Sys_Cnt_En  <= '0';
530                                            Ctrl_Sub_Pixel_Cnt_Rst <= '0';
531                                            Ctrl_Sub_Pixel_Cnt_En  <= '0';
532                                            next_state <= CAM_One_Transmission_Idle;
533                                   END IF;
534
535                      WHEN CAM_One_Transmission_Idle =>
536
537                                               Ctrl_Sys_Cnt_Rst <= '1';
538                                               Ctrl_Sys_Cnt_En  <= '0';
539                                               Ctrl_Sub_Pixel_Cnt_Rst <= '1';
540                                               Ctrl_Sub_Pixel_Cnt_En  <= '0';
541                                               next_state <= CAM_Setup_Switch;
542
543                      WHEN CAM_Two_Transmission =>
544
545    ---------------------CAM Two Transmission(528 cycles) -------------------------------------
546
547   =                                IF Ctrl_Sys_Cnt_Out >= TX_ST AND Ctrl_Sys_Cnt_Out <= TX_FN THEN
548
549                                         CAM_Status <= "10";
550   =                                     IF Ctrl_Sys_Cnt_Out >= TX_ST AND Ctrl_Sys_Cnt_Out <= TX_ST_FLAG THEN
551                                                CAM2_Frame_St <= '1';
552                                                CAM2_Frame_Fn <= '0';
553   =                                     ELSIF Ctrl_Sys_Cnt_Out >= TX_FN_FLAG_1 AND Ctrl_Sys_Cnt_Out <= TX_FN_FLAG_2 THEN
554                                                CAM2_Frame_Fn <= '1';
555                                                CAM2_Frame_St <= '0';
556   =                                     ELSE
557                                                CAM2_Frame_Fn <= '0';
558                                                CAM2_Frame_St <= '0';
559                                         END IF;
560
561   =                                     IF Ctrl_Sub_Pixel_Cnt_Out <= TX_SUB_PX_ST THEN
562                                            Ctrl_Sync_Clk <= '1';
563                                            ADC_Start <= '0';
564                                            next_state <= CAM_Two_Transmission;
565
566   =                                     ELSIF Ctrl_Sub_Pixel_Cnt_Out > TX_SUB_PX_ST AND Ctrl_Sub_Pixel_Cnt_Out <= TX_SUB_PX_ADC T
567                                            Ctrl_Sync_Clk <= '0';
568                                            ADC_Start <= '1';
569                                            next_state <= CAM_Two_Transmission;
570
571   =                                     ELSE
572                                            Ctrl_Sync_Clk <= '0';
573                                            ADC_Start <= '0';
574                                            next_state <= CAM_Two_Transmission;
575                                         END IF;
576   =                                ELSE
577                                            Ctrl_Sys_Cnt_Rst <= '0';
578                                            Ctrl_Sys_Cnt_En  <= '0';
579                                            Ctrl_Sub_Pixel_Cnt_Rst <= '0';
580                                            Ctrl_Sub_Pixel_Cnt_En  <= '0';
581                                            next_state <= CAM_Two_Transmission_Idle;
582                                   END IF;
583
584                      WHEN CAM_Two_Transmission_Idle =>
```

```
586                                         Ctrl_Sys_Cnt_Rst <= '1';
587                                         Ctrl_Sys_Cnt_En  <= '0';
588                                         Ctrl_Sub_Pixel_Cnt_Rst <= '1';
589                                         Ctrl_Sub_Pixel_Cnt_En  <= '0';
590                                         next_state <= Idle;
591
592                     WHEN CAM_Setup_Switch =>
593
594                                        -- Selecting signal for Cam 1
595                                        IF Ctrl_Switch_Cnt_Flag = CLOCK_LOW THEN
596                                           Ctrl_Sync_Clk_Sel <= '0';
597                                           S2 <= '0';
598                                           S1 <= '1';
599                                           S0 <= '1';
600                                           Ctrl_Switch_Cnt_En <= CLOCK_HIGH;
601                                           Ctrl_Sys_Cnt_En  <= CLOCK_HIGH;
602                                           next_state <= CAM_Int_Tsu;
603
604                                        -- Selecting signal for Cam 2
605                                        ELSE
606                                           Ctrl_Sync_Clk_Sel <= '1';
607                                           S2 <= '0';
608                                           S1 <= '0';
609                                           S0 <= '1';
610                                           Ctrl_Switch_Cnt_En <= CLOCK_HIGH;
611                                           Ctrl_Sys_Cnt_En  <= CLOCK_HIGH;
612                                           next_state <= CAM_Int_Tsu;
613
614                                        END IF;
615
616                     WHEN CAM_Tx_Switch =>
617                                        IF Ctrl_Switch_Cnt_Flag = CLOCK_HIGH THEN
618                                           Ctrl_Sys_Cnt_Rst <= '1';
619                                           Ctrl_Sys_Cnt_En  <= '1';
620                                           Ctrl_Sub_Pixel_Cnt_Rst <= '1';
621                                           Ctrl_Sub_Pixel_Cnt_En  <= '1';
622                                           next_state <= CAM_One_Transmission;
623
624                                         ELSE
625                                            Ctrl_Sys_Cnt_Rst <= '1';
626                                            Ctrl_Sys_Cnt_En  <= '1';
627                                            Ctrl_Sub_Pixel_Cnt_Rst <= '1';
628                                            Ctrl_Sub_Pixel_Cnt_En  <= '1';
629                                            next_state <= CAM_Two_Transmission;
630
631                                         END IF;
632
633
634                      WHEN OTHERS =>
635                                    next_state <= Idle;
636                 END CASE;
637                 END IF;
638                 current_state <= next_state;
639    END process;
640
641
642  END behavioral;
643
644
645
```