

DESIGN AND DEVELOPMENT OF AN EMBEDDED  
PLATFORM FOR COMPUTER VISION APPLICATIONS

KENNY KHOO KUAN YEW

MASTER OF COMPUTER SCIENCE

FACULTY OF ENGINEERING AND SCIENCE  
UNIVERSITI TUNKU ABDUL RAHMAN  
NOVEMBER 2011

**DESIGN AND DEVELOPMENT OF AN EMBEDDED PLATFORM  
FOR COMPUTER VISION APPLICATIONS**

By

**KENNY KHOO KUAN YEW**

A dissertation submitted to the  
Department of Internet Engineering and Computer Science,  
Faculty of Engineering and Science,  
Universiti Tunku Abdul Rahman,  
in partial fulfillment of the requirements for the degree of  
Master of Computer Science  
November 2011

## **ABSTRACT**

### **DESIGN AND DEVELOPMENT OF AN EMBEDDED PLATFORM FOR COMPUTER VISION APPLICATIONS**

**Kenny Khoo Kuan Yew**

Today, silicon chips are becoming more and more powerful despite the reduction in size. Various general processors and graphic processing units (GPUs) have been embedded in countless electronic gadgets such as personal digital assistants (PDAs), mobile hand-phone, digital cameras, and etc. With the high processing power of these embedded platforms and ever increasing size of both volatile and non-volatile storage, it opens up a great opportunity to integrate computer vision (CV) applications to these relatively low-cost and standalone devices. Applying CV applications inside electronic gadgets can be the next trends in industrial or commercial market. These devices can do specific smart and intelligent job such as object or text recognition for manufacturing, surveillance and security, as well as entertainment. It is discovered that there is yet to have a standard embedded software or hardware architecture in embedded systems (ES) for CV. An open architecture is needed for CV researchers to develop CV application without worrying the underlying embedded platform or CV library dependency in ES. This project provides a preliminary study of embedded computer vision (ECV) and design of a single common ECV platform using ARM-9 embedded processor. A few CV applications are also demonstrated on this platform, which share the same

embedded system characteristics and ECV library. Analysis on the demonstration application during development has been conducted to study the design work flow.

## **ACKNOWLEDGEMENTS**

I would like to specially thank my supervisor Dr Tay Yong Haur for his continuous guidance, great passion and support. He has been sharing his ideas and knowledge that helps me a lot in my research. Next, I would like to express my utmost gratitude to my co-supervisor Mr Mok Kai Ming for his support and encouragement during this project.

**FACULTY OF ENGINEERING AND SCIENCE  
UNIVERSITI TUNKU ABDUL RAHMAN**

Date: 11<sup>th</sup> November 2011

**PERMISSION SHEET**

It is hereby certified that **KENNY KHOO KUAN YEOW** (ID No: **06UIM02006**) has completed this dissertation entitled “**DESIGN AND DEVELOPMENT OF AN EMBEDDED PLATFORM FOR COMPUTER VISION APPLICATIONS**” under supervision of Dr Tay Yong Haur (Supervisor) from the Department of Internet Engineering and Computer Science, Faculty of Engineering and Science, and Mr Mok Kai Ming from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I hereby give permission to the University to upload softcopy of my thesis in pdf format into UTAR Institutional Repository, which will be made accessible to UTAR community and public.

Yours truly,

  
\_\_\_\_\_  
(KENNY KHOO KUAN YEOW)

## APPROVAL SHEET

This dissertation entitled “**DESIGN AND DEVELOPMENT OF AN EMBEDDED PLATFORM FOR COMPUTER VISION APPLICATIONS**” was prepared by **KENNY KHOO KUAN YEW** and submitted as partial fulfillment of the requirements for the degree of Master of Computer Science at Universiti Tunku Abdul Rahman.

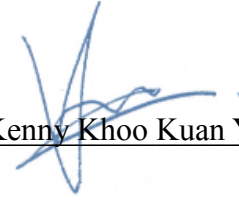
Approved by:

---

(Dr. Tay Yong Haur)  
Date: 11th November 2011  
Supervisor  
Department of Internet Engineering and Computer Science  
Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman

## DECLARATION

I hereby declare that the dissertation is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTAR or other institutions.

Name  Kenny Khoo Kuan Yew

Date 11<sup>th</sup> November 2011



## TABLE OF CONTENTS

	<b>Page</b>
<b>TITLE</b>	<b>i</b>
<b>ABSTRACT</b>	<b>ii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>iv</b>
<b>PERMISSION SHEET</b>	<b>v</b>
<b>APPROVAL SHEET</b>	<b>vi</b>
<b>DECLARATION</b>	<b>vii</b>
<b>TABLE OF CONTENTS</b>	<b>viii</b>
<b>LIST OF TABLES</b>	<b>xi</b>
<b>LIST OF FIGURES</b>	<b>xiii</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xvi</b>
<b>CHAPTER</b>	
<b>1.0 INTRODUCTION</b>	<b>1</b>
1.1 Background	1
1.2 Problem Statement	3
1.3 Scope of Work	5
1.4 Thesis Outline	7
1.5 Thesis Contribution	8
<b>2.0 LITERATURE REVIEW</b>	<b>11</b>
2.1 Overview	11
2.2 Study on Embedded Systems for Computer Vision Application	11
2.2.1 Imaging Input	12
2.2.2 Processor	14
2.2.3 Result Represents	17
2.2.4 Embedded Systems Platform	19
2.2.4.1 Basic Embedded Systems	19
2.2.4.2 Embedded Systems with Coprocessor	20
2.2.5 Comparison of Development Cycle	22
2.3 Study on Computer Vision Application in Embedded Systems	25
2.3.1 Typical Function of Computer Vision Applications	26
2.3.2 Typical Task of Computer Vision Application	28
2.3.3 Study of Available Computer Vision Library	30
2.3.4 Examination of CV Library Layout	34
2.4 Summary	36

<b>3.0</b>	<b>EMBEDDED COMPUTER VISION DESIGN METHODOLOGY</b>	<b>38</b>
3.1	Introduction	38
3.2	Overall Methodology	39
3.3	Specific Methodology	42
3.3.1	Generalize of CV Function into ECV Library	42
3.3.2	API Design	45
3.3.3	Code Optimization	47
3.3.4	Application Runtime Benchmark	50
3.4	Summary	52
<b>4.0</b>	<b>DEVELOPMENT OF EMBEDDED COMPUTER VISION PLATFORM</b>	<b>53</b>
4.1	Introduction	53
4.2	Examination to Identify Suitability of Hardware	53
4.2.1	Comparison of Core for Embedded Processor	54
4.2.2	Variety of ARM Embedded Processor	58
4.2.3	Coprocessor	60
4.2.4	Imaging Device	61
4.2.5	Interface Peripheral	63
4.2.6	Overall Hardware Design	64
4.3	Examination to Identify Software Layout	65
4.3.1	Operating Systems and Hardware Driver	65
4.3.2	Web Server Development	66
4.3.2.1	Server-side Scripting Engine	69
4.3.2.2	Web Server	70
4.3.2.3	Database Server	71
4.3.2.4	Streaming Media Server	73
4.3.2.5	Development and Examination	74
4.4	ECV Library Design	77
4.5	Code Optimization	78
4.5.1	Identifying the Critical Path	78
4.5.2	Compiler Optimization	80
4.5.3	Code Optimization without Coprocessor	82
<b>5.0</b>	<b>SAMPLE ECV APPLICATIONS</b>	<b>83</b>
5.1	Introduction	83
5.2	Sample ECV Applications	83
5.2.1	Basic ECV Platform Test Program	84
5.2.2	Motion Detection Test Program	85
5.2.3	Face Detection Test Program	86
5.2.4	Fingerprint Matching Test Program	87
5.2.5	Optical Character Recognition Test Program	89
5.2.6	Intelligent Character Recognition Test Program	90

<b>6.0</b>	<b>CONCLUSION AND FUTURE WORK</b>	<b>92</b>
6.1	Introduction	92
6.2	Results	93
6.3	Analysis	95
6.4	Conclusion	97
6.5	Future and Trend	98
6.6	Future Work	99
	<b>REFERENCES</b>	<b>101</b>
	<b>APPENDIX A</b>	<b>108</b>
	Manuals of ECVLIB Systems	
	<b>APPENDIX B</b>	<b>116</b>
	How to Add Your Own Function in UTAR-ECV Library	

## LIST OF TABLES

<b>Table</b>		<b>Page</b>
2.1	List of invisible spectrum and the type of sensor	13
2.2	The function block according to the type of embedded system platform	22
2.3	The development step according type of embedded systems platform	24
2.4	Computing systems resources usage according to typical computer vision function, in terms of I/O, DSP, memory and processing time	28
2.5	List of proprietary computer vision library	30
2.6	List of open source computer vision library	31
2.7	List of open source computer vision library according library classes	33
3.1	Comparison of time complexities versus number operations for an algorithms analysis notation	49
3.2	The platform combination for ECV application is benchmarking	51
4.1	Available ARM processor development board and the specification	58
4.2	The overall ECV platform hardware design specification	64
4.3	Software package that work in EP9315 development board	66
4.4	Inter-related web server and client component	68
4.5	Comparison of server-side scripting engine base on fractal benchmark	69
4.6	Compares the size of server-side scripting engine	69
4.7	List of open source web server which using C/C++ as development language	70
4.8	List of open source database server	72

4.9	List of open source database server using C/C++ as development language	72
4.10	Open source streaming media server	73
4.11	List of evaluate of combination of open source internet server	75
4.12	Miniature internet server software package that work in ARM920T embedded system	77
4.13	GCC optimizations and the compilations level	81
6.1	Motion detection test result	94
6.2	Face detection test result	94
6.3	Fingerprint matching test result	94
6.4	Optical character recognition test result	95
6.5	Intelligent character recognition test result	95
6.6	Summary of all test application CPU run time in all type of compilation setting	96

## LIST OF FIGURES

<b>Figures</b>		<b>Page</b>
1.1	Rovers on mars and computer-aided surgery systems	1
1.2	Research protocol and study life cycle	5
2.1	Basic structure of embedded computer vision	11
2.2	Example for calling colour.brightness function to reduce 10% brightness	15
2.3	Early state of ES evolution	15
2.4	Middle state of ES evolution	16
2.5	Current state of ES evolution	16
2.6	Web based control service building block	19
2.7	Simple CV processing block for basic embedded systems	20
2.8	Basic CV processing block	21
2.9	Typical function of computer vision application	26
2.10	ECV library layout and class diagrams	35
3.1	Interconnect between project main tasks	40
3.2	ECV library project's working flow	40
3.3	Proposed matrix map for ECV library layout.	43
3.4	Process to exanimate of open source CV library layout	44
3.5	Proposed 3D matrix map for ECV library layout	45
3.6	The ECV library API design flow chart	46
3.7	Code optimization flow	48
4.1	Suggestion of embedded computer vision platform for this project	54

4.2	Price and power comparison	56
4.3	Cirrus Logic EP9315 development board input/output	59
4.4	Cirrus Logic EP9315 systems on chip processor	60
4.5	Imaging device (From left fingerprint, 2 of colour imaging sensor)	62
4.6	Schematic of add on input/output peripheral interface controller	63
4.7	Overall ECV platform hardware design	64
4.8	Basic software block diagram	65
4.9	Inter-related web server and client software component	67
4.10	Detail of software block diagram for embedded computer vision platform	77
4.11	ECV library file directory and the library layout	78
4.12	Listing of flat profile output	79
4.13	Listing of call graph output	80
5.1	Web interface of the ECV platform	85
5.2	Motion detection application from the ECV's web interface	86
5.3	Face detection application from the ECV's web interface	87
5.4	Fingerprint matching illustration	88
5.5	Fingerprint matching application from the ECV's web interface	89
5.6	Optical character recognition application from the ECV's web interface	90
5.7	Intelligent character recognition application from the ECV's web interface	91
6.1	Illustrated of speed improvement between 3 platforms	97





## LIST OF ABBREVIATIONS

<b>Abbreviation</b>	<b>Meaning</b>
2D	Two Dimensional
3D	Three Dimensional
ALU	Arithmetic Logic Unit
ANN	Artificial Neural Network
API	Application Programming Interface
ARM	Advanced RISC Machine
ARM-9	Advanced RISC Machine, Version 9
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
CCD	Charge Coupled Device
CMOS	Complimentary Metal-Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CT	Computed Tomography
CV	Computer Vision
DMA	Direct Memory Bus
DSP	Digital Signal Processor
ECV	Embedded Computer Vision
EDA	Electronic Design Automation
ES	Embedded Systems
FAU	Fix-point Arithmetic Unit
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit

GCC	GNU Compiler Collection
GNU	GNU's Not Unix!
GPU	Graphic Processing Unit
HDL	Hardware Description Language
I/O	Input/Output
IC	Integrated Circuit
ICR	Intelligent Character Recognition
IDE	Integrated Drive Electronics Interface
IEEE	Institute of Electrical and Electronics Engineers
LAN	Local Area Network
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LIDAR	Light Detection and Ranging
MIPS	Microprocessor without Interlocked Pipeline Stages
MMU	Memory Management Unit
OCR	Optical Character Recognition
OS	Operating System
PC	Personal Computer
PDA's	Personal Digital Assistants
RADAR	Radio Detection and Ranging
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SIMD	Single Instruction and Multiple Data
SOC	System On Chip
SONAR	Sound Navigation and Ranging

SPECT	Single Photon Emission Computed Tomography
SQL	Structured Query Language
SVM	Support Vector Machine
USB	Universal Serial Bus
VGA	Video Graphics Array
VOIP	Voice Over Internet Protocol

# CHAPTER 1

## INTRODUCTION

### 1.1 Background

In recent years, we have witnessed a dramatic increment in the usage of computer vision (CV) in embedded systems (ES). CV was successfully used in various mission-critical systems right from the landing of the exploration rovers on Mars to computer-aided surgery (Larry Matthies, 2005).

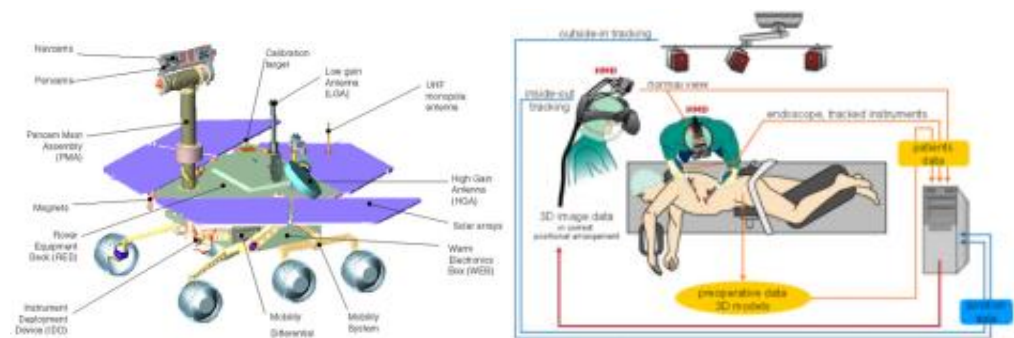


Figure 1.1 : Rovers on Mars and computer-aided surgery systems

Traditionally, different computer vision domains such as computer-aided surgery and surveillance were addressed separately due to the different nature of the application domains. However, these domains share many common problems related to their real-time, embedded-system characteristics. CV researchers have difficulties to implement their CV algorithms directly into a small embedded device for a specific CV application (Dana H. &

Christopher M. Brown, 1982). These situations occur because there are:

- No standardized framework or architecture in embedded system for CV
- No common interface for CV programming library

CV is also widely used in industrial embedded systems, taking part in production and inspection processes. Cameras find their way to everyday appliances such as cell phones, personal digital assistants (PDAs), presentation appliances and vehicles. Moreover, cameras are becoming “smarter” by gaining the capability of processing the acquired images internally (K.Y. Khoo & Y.H. Tay, 2006).

The field of CV can be characterized as immature and diverse. There is no standard formulation on how computer vision problems should be solved. Instead, there exists an abundance of methods for solving various well-defined computer vision tasks (Bernd Jahne & Horst HauBecker, 2000).

Combining CV and ES eventually creates a new field known as the embedded computer vision (ECV). The existence of a growing demand for ECV applications are used to tackle problems which can hardly be solved using traditional sensor systems, i.e. camera. However, due to the high computational complexity of most computer vision algorithms, applying of such applications on embedded systems is not a straight forward task.

This new field is gaining popularity as more people are showing interest in ECV, and attempt to bring more CV applications in the form of ES. Institute of Electrical and Electronics Engineers (IEEE) organization has also shown their interest by conducting the first embedded computer vision workshop in Jun 2005 at San Diego, United State (IEEE, 2005). Hence this ECV project tries to study and develop a possible platform for ECV using Advanced RISC Machine, Version 9 (ARM-9) processor.

## **1.2 Problem Statement**

There are many types of embedded systems platforms, such as microprocessor, digital signal processor (DSP), graphics processing unit (GPU), floating-point unit (FPU) and field-programmable gate array (FPGA) (Wayne Wolf, 2006). Each of them is possible to be used for CV application. However, there is not much study on platform suitability for CV application in term of processing power, speed and time to market.

There are no CV libraries designed for embedded systems. Furthermore, generalization of CV functions into proper library layout is difficult due to the CV files which are immature and diverse. There are many formulations to solve a particular CV problem. CV methods are often very task specific and seldom can be generalized over a wide range of applications.

Currently CV developers are not able to deploy any CV application in embedded board directly; this is due to the lack of hardware optimization and

the lack of possibility for real-time application execution. The possible outcome for the study of ECV is to design a platform where CV developers do not need to know the underlay of embedded systems, and the CV application can still be able to deploy on it.

Today, most of the ES are the system on chip (SOC), which is an application-specific integrated circuit, for which the central processing unit (CPU) was purchased as intellectual property to add integrated circuit (IC) design. For example:

- Integrate a video processing unit on chip for multimedia purposes, such as Sigma Design Inc.'s multimedia chip EM8600 (Sigma Designs Inc, 2004) which can decode high definition Windows media format 9. The equivalence processing power in this video encoding is only found in Intel Pentium 4 3.06GHz (Microsoft Corp, 2008).
- Integrate voice codec with Advanced RISC Machine (ARM) processor for VOIP purpose such as Octasic DSP chip which can handle 672 channels of G.971 audio codec at a time (Robert Keenan, 2003).

Intellectual property in the SOC is actually a unit to perform a specific task in real-time where the general purpose processor unable to perform. Currently there is no specific SOC for CV application. This is because the CV methods are often very task specific and can seldom be generalized over a wide range of applications.

Most of the embedded system for computer vision application is laid on platform of combination between microprocessor, DSP, coprocessor (FPU or GPU) and FPGA. Neither the co-processor nor coprocessor can perform special group calculation which can be between 1,000 to 10,000 times much faster than microprocessor. Further research must be conducted to identify a path of computer vision code which can take advantage of the embedded system (Graefe et al., 1991).

### 1.3 Scope of Work

Figure 1.2 illustrates that this project involves research protocol and life cycle study (Wendy Bergeru, 2002). It starts by identifying the need for embedded system and hardware-software design study, follows by establishing and maintaining the study, analysing the archive data, communicating results by submitting it as a paper for conference and lastly wrap-up the study and evaluates the outcome.



Figure 1.2 : Research protocol and life cycle study

Further development has to be done in order to conduct this research:

1. Investigate and identify suitable development board



2. Design a complete ECV system with input-output infrastructure
3. Establish an operating systems on the development board
4. Configure or rewrite input/output device driver for camera and sensor

The research approach includes:

1. Examine a miniature internet server for ECV
2. Research on computer vision library layout
3. Research on code optimization for hardware device
4. Analyse the sample computer vision application

Firstly, embedded systems design project is studied in order to determine the suitable hardware design that suites the CV application in term of flexibility and availability. Secondly, the project installs suitable operating system (OS) into the development board. Thirdly, as many as the input/output device is installed and attached with the development board. Forth steps, examination and finding of the miniature internet web server as a web infrastructure for input-output control.

Next, this project includes a few CV sample applications which were implemented on the ECV platform. The ECV library is built and grew through the development of these sample applications. Meanwhile a research on computer vision library layout is conducted to design a better ECV library by generalizing the CV functions and group them. Three methods of code optimization had been study and applied:

- General code optimization
- Code optimization for generic FPU
- Code optimization for specific FPU

At last, the analysis of these CV sample application has been conducted, where the sample application are compiled in few different platform to test the performance.

#### **1.4 Thesis Outline**

In the first chapter, embedded computer vision subject and the problem statement are introduced. Project scopes are initiated and the project contribution is listed.

Chapter 2 studies the computer vision application in embedded systems and the evolution of embedded systems in conjunction of popularity of CV application. It also discusses the issues of embedded systems in computer vision domain.

Chapter 3 is about the method that this project had been carry-out by answering the most important of “How”:

- How to determine embedded systems design
- How computer vision application take ES advantage
- How to select a suitable embedded processor

- How to generalize CV function into a specific library (ECV Library)

After knowing computer vision and embedded systems with ECV design idea in mind, Chapter 4.2 will show how the project had conducted a quantitative research to determine an embedded systems platform and peripheral device that is suitable for most of the computer vision.

Chapter 4.3 and Chapter 4.4 mainly explain the development of software. The basic operating systems and driver are developed. Internet server which acts as main interface of ECV board is pulled together and the important core software of ECV library is designed base on the research result in Chapter 3. Chapter 4.5 will provide details of compiler and code optimization on several ECV platforms.

Chapter 5 will cover a few CV sample applications which are working on new design of ECV platform. Finally Chapter 6 will analyse these CV sample application in term of execute speed in ECV platform and discuss the future work and potential ECV platform's product.

## **1.5 Thesis Contribution**

Minor contribution of the project is the development work on ECV platform. Such works are:

1. Searching for a suitable hardware platform

2. Operating systems installation and configuration
3. Input-output device driver modification and development

Another contribution is the finding of a miniature internet server for ECV platform which requires least processing power and memory consumption.

The study of various open source CV library has led to the concern of having a generalized CV library as most of them do not have common classification for particular a CV function.

This project had developed a new ECV library which is a CV library that has been designed in mind of embedded systems domain. It is an essential combination library from the study of various open source CV library layout. It has generic code which can later be optimized with other architecture of embedded system. By default it has been optimized with Cirrus Logic EP9315 microprocessor.

Immediate contribution is 5 CV samples application which had been optimized for Cirrus Logic EP9315 microprocessor by using a new ECV library.

During the course of research, a few papers were submitted for conference. They were:

1. Paper title : A Preliminary Study on Embedded Platforms for  
Computer Vision Applications  
Author : Kenny Khoo Kuan Yew, Tay Yong Haur  
Conference : Regional Computer Science Postgraduate Conference 2006  
Organizer : Universiti Teknologi Malaysia
  
2. Paper title : A Study of Digital Colour Imaging Systems Design for  
Embedded Systems  
Author : Kenny Khoo Kuan Yew, Tay Yong Haur  
Conference : MMU International Symposium on Information and  
Communication Technologies (M2USIC 2007)  
Organizer : Multimedia University, Malaysia
  
3. Paper title : Study and Implementation of Embedded Computer Vision  
Library  
Author : Kenny Khoo Kuan Yew, Tay Yong Haur  
Conference : 3rd International Conference on Postgraduate Education  
(ICPE3 2008)  
Organizer : Malaysian Deans Graduate Studies Council (MYDEGS)

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Overview

The literature review is a study on embedded systems for computer vision application and via versus study on computer vision application in embedded systems.

#### 2.2 Study on Embedded Systems for Computer Vision Application

The definition of embedded computer vision system means a hardware which is able to perform real-time computer vision algorithm. The diagram below shows an ideal block system of it (Dana H. & Christopher M. Brown, 1982).



Figure 2.1 : Basic structure of embedded computer vision

In the future, electronic gadgets will be equipped with powerful processing power. Every 18 months, the number of transistor is doubled up

inside a single die of chip (David C. Brock, 2006). The speed of processor is limited at some point due to the difficulty to keep power consumption and reliability (Aleksandr Mitrofanov, 2006). The size of processor is limited at some point due to the quantum tunnelling effect in wafer manufacturing processes (Michael Kanellos, 2003). The doubling of transistor, limitation of the speed of processor and limitation of the size of processor caused manufacturer to:

- Multiply processor core on single die of chip, or
- Enable single instruction and multiple data (SIMD) processing on single die of chip.

Computer vision application is always dealing with a group of data (Such as image or array of data). Processor with SIMD ability will boost application speed such computer vision (CV) application (Herb Sutter, 2005).

### **2.2.1 Imaging Input**

Figure 2.1, shows that imaging input as a device that produce array of data that represent a real world environment. Basically the device sensing mechanism is touch sensing or seeing. The examples of touching mechanism sensor device are push button, fingerprint reader. In general seeing mechanism sensor device can detect 2 types of spectrum:

- Visible spectrum
- Invisible spectrum

Visible spectrum is a portion of the electromagnetic spectrum which can be seen by human eyes. Sometimes it is also called as optical spectrum. The wavelength of it is between 380nm to 750nm (Bir Bhanu & Ioannis Pavlidis, 2004).

Invisible spectrum is an electromagnetic spectrum that human eye unable to see or detect. Some of it is useful because it can pass through certain object, such as X-ray (around 1nm wavelength) which can display inner structure of human body. Due to this imaging device ability, most of them produced the image by tomography method. It is rarely used in CV application, because both systems are normally targeted for far object which is invisible to human eye (Bir Bhanu & Ioannis Pavlidis, 2004).

Table 2.1 : List of invisible spectrum and the type of sensor

Physical Phenomenon		Approximately Wavelength (in Meter)	Type of Sensor
Below Visible Spectrum	Sound	2 E-02	Sound Navigation and Ranging (SONAR)
	Ultrasound	5 E-05	Medical Sonography (Ultrasonography)
Beyond Visible Spectrum	Laser	6 E-07	Light Detection and Ranging (LIDAR)
	X-Rays	1 E-07	Computed Tomography (CT)
	Airport Radar	1 E-10	Radio Detection and Ranging (RADAR)
	Gamma Rays	1 E-11	Single Photon Emission Computed Tomography (SPECT)



To detect signal within visible spectrum, a seeing mechanism sensor device, or so called colour image sensor will be used. The major constraints in image sensor are sensitivity, resolution, progressive/interlaced scan image, and interval between sequence images (video frame per second). These constraints will determine the size of information that CV systems receive for analysis. An ECV hardware platform must select the right image sensor as it will provide sufficient information for CV systems to calculate accurate result.

Camera is the most popular image sensor in CV application. Nowadays camera is part of every day's appliances such as cell phones, PDAs, presentation appliances and vehicles. Moreover, cameras is becoming "smarter" by gaining the capability of processing the acquired images internally.

Common criteria of all seeing mechanism sensor devices are that it senses the reflection of the electromagnetic spectrum source. For example, colour image camera sense the reflection of light from an environment object; RADAR and LIDAR image systems use radio/laser waves to determine the distance to an object by measuring the time delay between transmission pulse and detection of the reflected pulse.

### **2.2.2 Processor**

As imaging input produces array of data and the algorithm always applied on the set of data, the processor must be able to handle parallel

processing. In a case study, a simple brightness adjustment operation of an image requires to perform 1 million times of colour.brightness function for each pixel in 1280 x 1024 pixel images. Figure 2.2 shows the example of colour.brightness function mathematic.

$$\sum_{X=0}^{1280} \sum_{Y=0}^{1024} color.brightness (RGB, -10\%)$$

Figure 2.2 : Example for calling colour.brightness function to reduce 10% brightness

The sequence of images and sound is called video. Processor itself might not be able to process this video data in real-time. Integrated DSP/GPU in the systems helps to process this video data in real-time. DSP/GPU is able to do parallel processing, and a single instruction for multiple data operations to solve the large data calculation (Ian Buck, 2005).

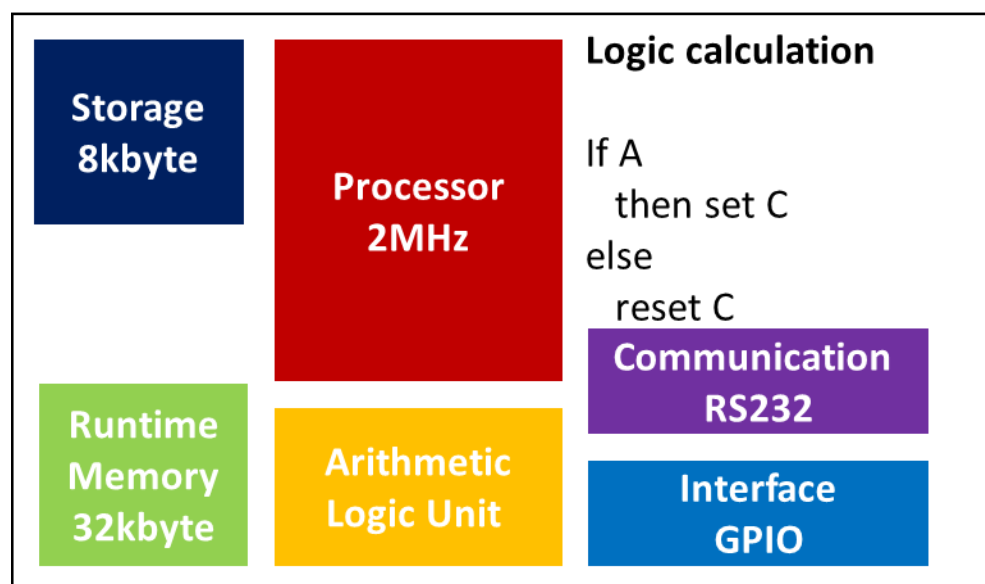


Figure 2.3 : Early state of ES evolution

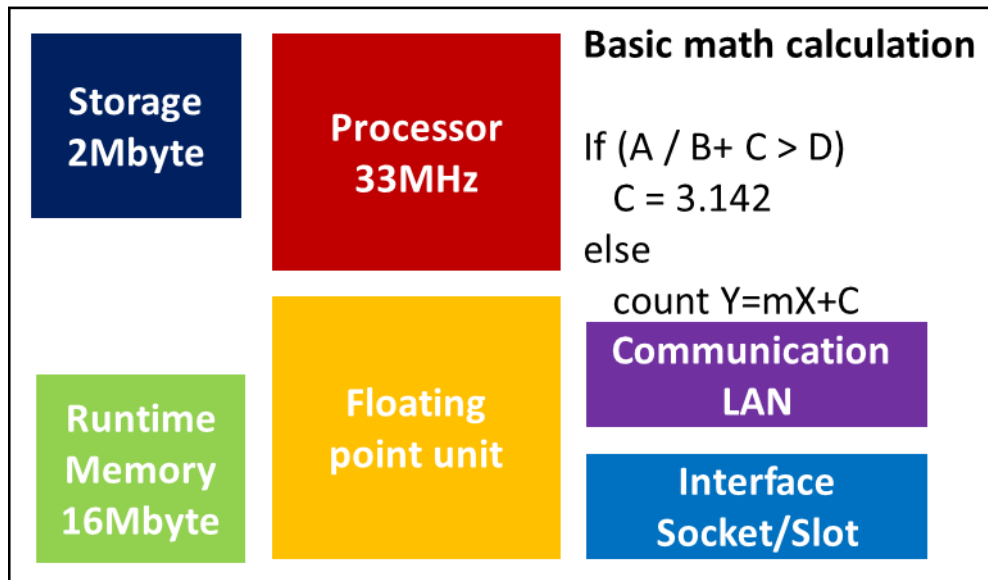


Figure 2.4 : Middle state of ES evolution

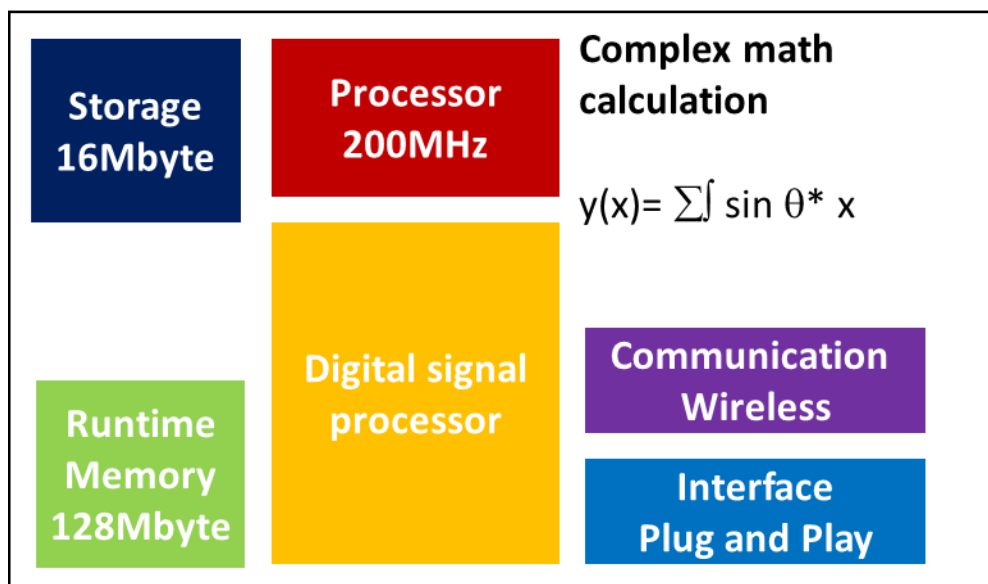


Figure 2.5 : Current state of ES evolution

Figure 2.3, Figure 2.4, and Figure 2.5 show embedded systems evolution. As time passes, embedded systems are capable to contain an OS and cooperate with DSP/GPU to calculate complex mathematic (Wayne Wolf, 2006). With the high processing power of these embedded platforms and ever

increasing size of both volatile and non-volatile storage, it opens up a great opportunity to integrate CV applications to these relatively low-cost and standalone devices.

In the beginning of embedded systems evolution (Figure 2.3), ES is just an input output device which gave electrical output base on incoming command, or a variable, the software architecture just a control loop systems.

At the time when microprocessor has become fast and arithmetic logic unit (ALU) has become more advanced, ES itself can collect data, and perform calculation to determine the decision and output. At this stage (Figure 2.4), ES can contain a simple operating system and software structure.

Current state of embedded systems evolution (Figure 2.5), shows that as time goes on, embedded systems are capable to contain an OS and cooperate with DSP/GPU to calculate complex math (K. Rath & P. K. Meher, 2006).

### **2.2.3 Result Represents**

Usage of embedded computer vision can be a standalone device, or a reporting/monitoring device. ES can attach input/output peripheral for mobility. Such peripheral are liquid crystal display (LCD), input button, motor, and servo motor.

Outcome for standalone device will be immediate action. For example, iris recognition door access systems will unlock the door if the user permission is granted.

If ECV systems act as reporting/monitoring device, this device may report the result to the user through user interface. In the era of internet, an embedded computer vision system is able to link up and send any message to user through web.

High capacity integration in SOC enables ES to have much functionality, such as Ethernet connection or wireless communication. Due to the trend of web interface, and ability of network access in ES, the most suitable control interface today will be web-based control. User can control the ES from web browser at anywhere. To enable this web-based control, we will later study the internet server for ES. Basic type of server which might be related is web server, video server, scripting server and database server. Figure 2.6 shows the building block of internet server (James Kurien et al., 2002).

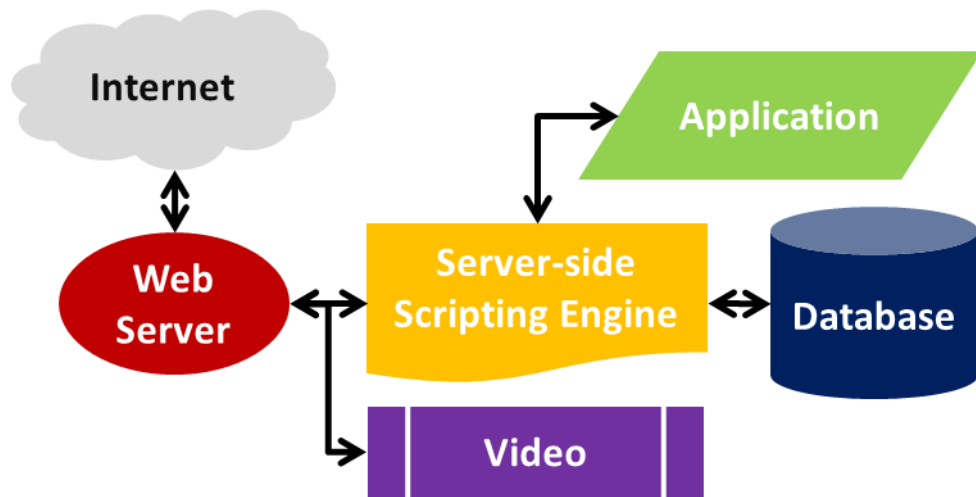


Figure 2.6 : Web-based control service building block

## 2.2.4 Embedded Systems Platform

The core of processing will determine the overall ECV design and performance and such CV application can be categorised into 2 different platforms of embedded system design (Wayne Wolf, 2006) which is:

1. Basic embedded systems
2. Embedded systems with coprocessor

### 2.2.4.1 Basic Embedded Systems

An embedded processor only processes single thread of program and handle basic input/output. A fine example of such computer vision application that runs on this kind of systems is “White surface and black colour line tracking systems to guide the movement of robot”. This simple computer vision application will grab the input data from black and white detection

sensor. The input image data is a single array of data. Each pixel is a binary data that represents either black or white colour. This CV application will detect the position of black colour, and will decide the robot movement to ensure the black line is always in the middle of the array in real-time (Pakdaman M. & Sanaatiyan M., 2009). It is illustrated in Figure 2.7.

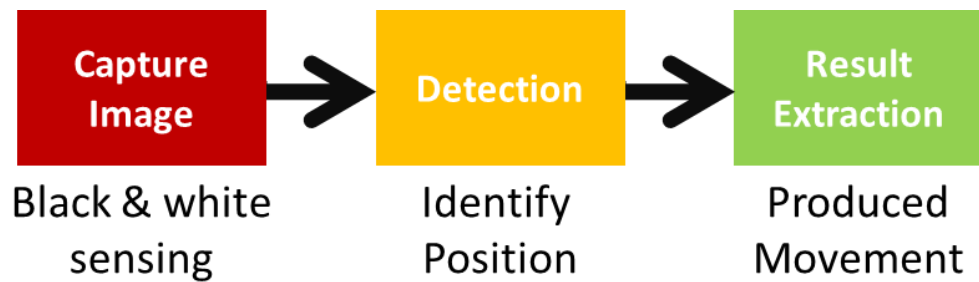


Figure 2.7 : Simple CV processing block for basic embedded systems

#### 2.2.4.2 Embedded Systems with Coprocessor

An embedded processor connects to a coprocessor with a high speed bus interface such as direct memory access (DMA) bus. This embedded processor processes a thread of program and the coprocessor processes a group of floating-point/fix-point data in memory. It combines domain specific execution clusters to produce high performance low-power accelerators for perception applications in the embedded space (Binu Mathew et al., 2003).

Most of the silicon on chip (SOC) do combine an embedded processor with coprocessor in a single die of chip where it allows user to implement algorithms without hardware knowledge. A descriptor method in which the central processing unit gave instructions to coprocessor through a register

array enabled task-level parallel processing as well as pixel-level parallel processing in the processing units (Komuro T. et al., 2010).

Examples of computer vision application that can run on this kind of systems are:

- Finger print verification
- Palm print recognition
- Text recognition
- Human face detection
- Motion detection and etc.

For example in a motion detection of security monitoring, in order to ensure a real-time processing, the programming code must not be  $O(N^2)$  style. Input image data is a two dimensional (2D) array of data and each pixel might be 24 bits of colour data. But to apply CV algorithm, the input image data needs to be in correct colour space and standard. To do so, a pre-processing function needs to de-noise, convert or normalise input image data. The Figure 2.8 shows the interconnection of processing function in the CV processing block.

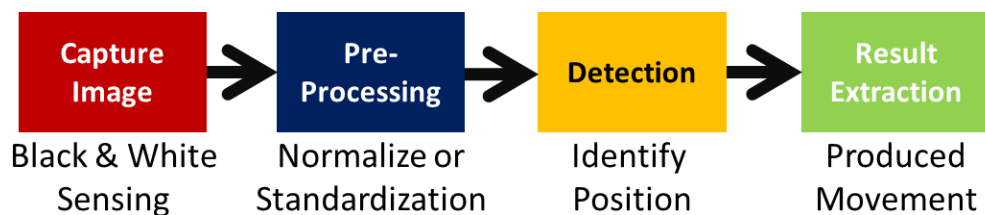


Figure 2.8 : Basic CV processing block



### 2.2.5 Comparison of Development Cycle

The Figure 2.2 shows the function block according to the type of embedded system platform.

Table 2.2 : The function block according to the type of embedded system platform

Type of Embedded System Platforms	Basic Embedded Systems	Embedded Systems with Coprocessor	
		System On Chip (SOC)	Configurable Chip
Function Block	Fix-point Arithmetic Unit (FAU)	Floating Point Unit (FPU)	Customized Floating Point Unit
		Graphics Processing Unit (GPU)	Customized Graphics Processing Unit
		Specific Coprocessor	Customized Coprocessor

User develops software on top of basic embedded systems has no concern of API software. All the programming code is translated into machine code by compiler.

A SOC is an integrated circuit that integrates all components of a computer or other electronic system into a single chip. It may contain digital, analogue, mixed-signal, and often radio-frequency functions. A typical application is in the area of embedded systems. Electronic system, such FAU, FPU, GPU, and specific coprocessor had a set of advance function in SOC for use.

Most of SOC come with a software distribution and the general software library where both of it are optimized with the FPU, GPU or theirs specific coprocessor. User develops software on top of it with:

- Guided API to use the specific coprocessor
- Optimized general software library

Programming style of the algorithm is adjusted according to the guided API for the specific coprocessor. Adjusted programming style can gain speed from that specific SOC.

CPLD, ASIC, FPGA is a configurable chip. It contains logic blocks that can be programmed to perform the function of basic logic gates such as AND gate, and XOR gate, or more complex combinational functions such as decoders or mathematical functions. For it an algorithm is a hardwired connection in the chip (Proshanta Saha, 2007). A programmable CPLD or FPGA has the ability to update the functionality. One-time programmable ASIC is more cost effective in large volume production.

Customized function block provides flexibility to user to customise coprocessor. However, with such flexibility, user will have to spend more time on:

- Design the customized function block
- Test the customized function block

- Manage the programming API
- Optimize general software library for the customized function block

Table 2.3 : The development step according type of embedded systems platform

Development Step	Type of Embedded Systems Platform		
	Basic Embedded Systems	Embedded Systems with Coprocessor	
		System On Chip (SOC)	Configurable Chip
1 CV Algorithm Design	Same		
2 Algorithm Strip Down	Same		
3 Hardware Optimization	None	Profiling	Register Transfer Level
		Using Coprocessor Extension in Critical Loop	Design Synthesis
			Gate Level Simulation
			Logic Synthesis

Table 2.3 shows the development step of generic algorithm for different types of embedded system platforms. The only different development step among them is hardware optimization. Note that embedded systems with coprocessor for profiling performance might need to use coprocessor extension in critical loop. The optimization for configurable chip requires knowledge of hardware description language (Tammy Noergaard, 2005). The adoption of FPGA in high performance computing is currently limited by the complexity of FPGA design as compared to conventional software and the extremely long turn-around times of current design tool where 4 to 8 hours of waiting time is necessary even after a minor change is made to the source code.

The development cycle for basic embedded systems is straightforward but it hardly increases the runtime speed. And the development cycle for SOC is much faster than customizable circuit. This is mainly due to SOC:

- Does not need to test the customized function block
- Does not need to optimize the general software library for the customized function block

However SOC does not support dynamic reconfiguration in runtime. Instead, it adapts itself to a specific functionality.

### **2.3 Study on Computer Vision Application in Embedded Systems**

The CV field can be characterized as immature and diverse, and there are many formulations to solve a particular CV problem. CV methods often are very task specific and seldom can be generalized over a wide range of applications.

Computer vision can also be described as a complement (but not necessarily the opposite) of biological vision. In biological vision, the visual perception of humans and various animals are studied, resulting in models of how these systems operate in terms of physiological processes (Dana H. & Christopher M. Brown, 1982). Computer vision, on the other hand, studies and describes artificial vision system that is implemented in software and/or hardware. Interdisciplinary exchange between biological and computer vision

has proven increasingly fruitful for both fields.

### 2.3.1 Typical Function of Computer Vision Applications

The organization of a computer vision system is highly application dependent. Some systems are standalone applications which solve a specific measurement or detection problem. While other systems constitute a subsystem of a larger design which, for example, also contains subsystems for control of mechanical actuators, planning, information databases, human machine interfaces, and etc. The specific implementation of a computer vision system also depends on if its functionality is pre-specified or if some part of it can be learned or modified during operation. There are, however, typical functions which are found in many computer vision systems (Gosta H. & Hans Knutsson, 1995).

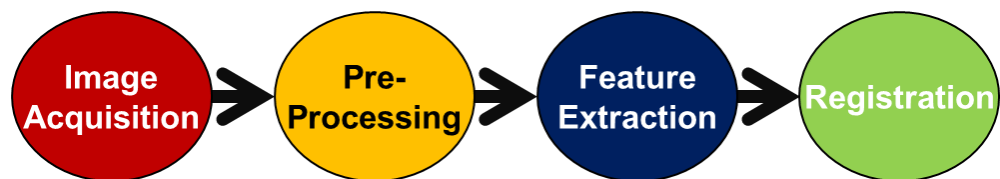


Figure 2.9 : Typical function of computer vision application.

Typical functions of computer vision system can be generalized into the following subsystems in Figure 2.9. The image acquisition subsystems are just a simple communication protocol to grab image or image sequence from imaging devices, e.g. camera, radar, tomography system.

Pre-processing function uses a lot of memory to do image processing for noise reduction on the image, but it also reduces the overall amount of data. Feature extraction function perform a lot of CV algorithms to further reduce the raw data to a set of features, which ought to be invariant to disturbances such as lighting conditions, camera position, noise and distortion. Examples of feature extraction function are:

- a. Perform edge detection or estimation of local orientation
- b. Extracting corner features
- c. Detect blob features
- d. Extract spin images from depth maps
- e. Acquire contour lines and curvature zero crossings
- f. Generate features with the scale-invariant feature transform

After obtaining a set of features, CV researchers are able to apply their own idea/algorithm for specific CV application. The registration function has to bring up a final hypothesis. It is a high level processing, where at this point the input is typically a small set of data for example a set of points or an image region which is assumed to contain a specific object. The remaining processing deals with, for example:

1. Verification that the data satisfy model-based and application specific assumptions
2. Estimation of application specific parameters, such as object poses or object size

### 3. Classifying a detected object into different categories

Table 2.4 : Computing systems resources usage according to typical computer vision function, in terms of I/O, DSP, memory and processing time

Typical CV Function	Computing Systems Resources			
	I/O	Coprocessor	Memory	Processor
Image Acquisition	Yes		Low	Low
Pre-processing		Yes	Very High	High
Feature Extraction		Yes	Very High	Very High
Registration	Yes		High	High

Based on each typical CV function, there are different amount of computing systems resource used, as shown in the Table 2.4. Some typical CV function always run in critical time, any data which cannot be processed in real-time will be ignored. Bottleneck of any CV subsystems might cause real-time processing fail. Pre-processing subsystems always deal with adjustment of large amount of images. The feature extraction subsystems processing time is the worst, because it extracts a set of attribute from each static image or a sequence of images. Both subsystems are computation intensive and deals with large amount of data. Only GPU which have SIMD capability can process all large amount of data faster and probably in real-time.

#### 2.3.2 Typical Task of Computer Vision Application

Typical tasks of CV systems application that currently realize can be categorized as (Linda G. Shapiro & George C. Stockman, 2001):

- Object recognition task - Detect the presence of known objects or living things in an image, possibly together with estimating the pose of these objects.
- Optical character recognition (OCR) task - Takes pictures of printed or handwritten text, and then converts it into computer readable text such as American Standard Code for Information Interchange (ASCII) or Unicode.
- Tracking task - Known objects movement and direction through an image sequence.
- Scene interpretation task - Create a model from an image/video of scenes.
- Ego motion task - Determine the motion of the camera itself.

Thus typical task reflect the design of embedded systems. Such task will directly reflect the embedded systems input device and memory design example of such task are:

1. Object recognition tasks needs at least a single imaging device to capture object, or a fingerprint reader device.
2. Tracking task and scene interpretation task also need imaging device but there need to keep track previous data with some large memory device.



### 2.3.3 Study of Available Computer Vision Library

This project is to build an ECV library which is optimized with the selected ECV platform design. An examination on all available CV libraries is conducted.

Table 2.5: List of proprietary computer vision library

No	Product	Company	Country	Website
1	Common Vision Blox	Stemmer Imaging	Germany	commonvisionblox.de
2	Halcon	MVTec	Germany	mvtec.com
3	LabView-NI Vision	National Instruments	USA	ni.com
4	Matlab Image Processing Toolbox	The MathWorks	USA	mathworks.com
5	Matrox Imaging	Matrox Electronic Systems	Canada	matrox.com
6	NeuroCheck	NeuroCheck	Germany	neurocheck.com
7	Open eVision	Euresys	Belgium	euresys.com

Table 2.5 shows the list of proprietary computer vision library. But this project will not concern about proprietary computer vision library as it is:

1. Close source and hard to evaluate inner code.
2. Software lockup with high maintenance cost.
3. Fully depend on vendor for available function, new function upon request might be slow.
4. Operating systems dependent.

Study on available open source computer vision related library had been conducted. The result is that there are at least 16 open source computer vision libraries for CV developer to use; Table 2.6 is the list of it. In open source community, there are still many CV related projects, as many of it are not intended to be designed as library, so it is not discussed. Main concern in open source software reliability is:

- Development status - Active or pending
- License - Free to use?
- Number of steady/registered developers
- Operating systems platform

Table 2.6 : List of open source computer vision library

No	Library	Core Language	Platform	Current Version	Release Date	Size
1	Blepo	C	Linux	0.6.2	24-Aug-2007	9,095kB
2	Camellia	C	Linux	2.24	24-Jan-2005	373kB
3	Diamond3D	C++	Win/Linux	0.4	13-June-2003	452kB
4	Gandalf	C/C++	Win/Linux	1.6	21-Sep-2006	6,574kB
5	JavaVis	Java	JRT	3.51	16-Dec-2005	1,700kB
6	LibCV	Java	Linux	0.1	16-Jun-2006	49kB
7	LTI-Lib	C	Linux	1.9.15	25-Nov-2005	105kB
8	Mimas	C/C++	Linux	2.1	30-Oct-2006	45,756kB
9	NokiaCV	C/C++	Symbian	1.0	19-Jun-2008	3,585kB
10	OpenCV	C/C++	Win/Linux	1	6-Nov-2006	10,866kB
11	OpenVIDIA	C	Win/Linux	0.8	27-Jun-2005	1,299kB

12	TLIB	C/C++	Win/Linux	0.3	10-Dec-2003	2,626kB
13	Torch-CV	C/C++	Linux	2.1	2-Apr-2007	7,347kB
14	VIGRA	C/C++	Win/Linux	1.5.0	8-Dec-2006	12,554kB
15	VXL	C++	Win/Linux	1.10.0	6-Jan-2008	21,553kB
16	WebCamXtra	C++	Win/Linux	0025	16-May-2006	475kB

The Table 2.6 shows the sixteen of open source computer vision libraries which been analysed. In a detail preview of the source code for each CV library, the library classes are categorized according to their functions in Table 2.7. We found that most of them do not have internet service, hardware driver, wavelet domain processing function, artificial neural network (ANN) and support vector machine (SVM) support.

Table 2.7 : List of open source computer vision library according library

Library	File Handling	Internet Service	Desktop GUI	Feature Extraction		Image Processing			Image Manipulate	Camera Calibration	Hardware Driver	Math				Machine Learning		
				2D	3D	Time	Frequency	Wavelet				Linear Algebra	Matrix	Vector	Statistic	ANN	SVM	
1	Blepo	Yes	-	Yes	Yes	-	Yes	Yes	-	Yes	Yes	Yes	Yes	Yes	-	-	-	
2	Camellia	-	-	-	Yes	Yes	Yes	-	-	Yes	-	-	Yes	-	-	Yes	-	-
3	Diamond3D	-	-	-	Yes	-	Yes	-	-	Yes	-	-	Yes	Yes	-	-	-	-
4	Gandalf	Yes	-	Yes	Yes	Yes	Yes	Yes	-	Yes	-	-	Yes	Yes	Yes	-	-	-
5	JavaVis	Yes	-	Yes	Yes	-	Yes	-	-	Yes	-	-	-	-	-	-	-	-
6	LibCV	-	-	Yes	Yes	-	Yes	-	-	-	-	-	Yes	-	-	-	-	-
7	LTI-Lib	-	-	Yes	Yes	-	Yes	-	-	Yes	-	-	Yes	Yes	-	-	-	-
8	Mimas	Yes	-	-	Yes	-	Yes	Yes	-	Yes	Yes	-	Yes	Yes	-	-	-	-
9	NokiaCV	Yes	-	-	Yes	-	Yes	-	-	Yes	-	-	Yes	Yes	Yes	-	-	-
10	OpenCV	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
11	OpenVIDIA	Yes	-	-	Yes	-	Yes	Yes	Yes	Yes	-	-	Yes	Yes	Yes	Yes	-	-
12	TLIB	Yes	-	Yes	Yes	Yes	Yes	Yes	-	Yes	-	-	Yes	Yes	-	-	-	-
13	Torch-CV	Yes	-	Yes	Yes	-	Yes	Yes	-	Yes	-	-	Yes	Yes	-	-	-	-
14	VIGRA	Yes	-	-	Yes	-	Yes	Yes	-	Yes	-	-	Yes	Yes	Yes	Yes	-	-
15	VXL	Yes	-	Yes	Yes	Yes	Yes	Yes	-	Yes	Yes	-	Yes	Yes	Yes	-	-	-
16	Myron	Yes	-	Yes	Yes	-	Yes	-	-	Yes	-	-	-	-	-	-	-	-

#### **2.3.4 Examination of CV Library Layout**

The typical CV function (Chapter 2.3.1), typical task of CV (Chapter 2.3.2) and CV library classes (Chapter 2.3.3) can be a reference point to generalize all the CV method/function over a wide range of applications to be a standard CV library.

Based on the type of CV application usage, the runtime of it can be categorised as:

1. One short runtime - This kind of CV application can tolerate delay, such application is fingerprint recognition (can be delay more than few seconds).
2. Continues runtime - A kind of CV application which needs real-time processing. Such as motion detection where it needs to process 15 frames of image data per second.

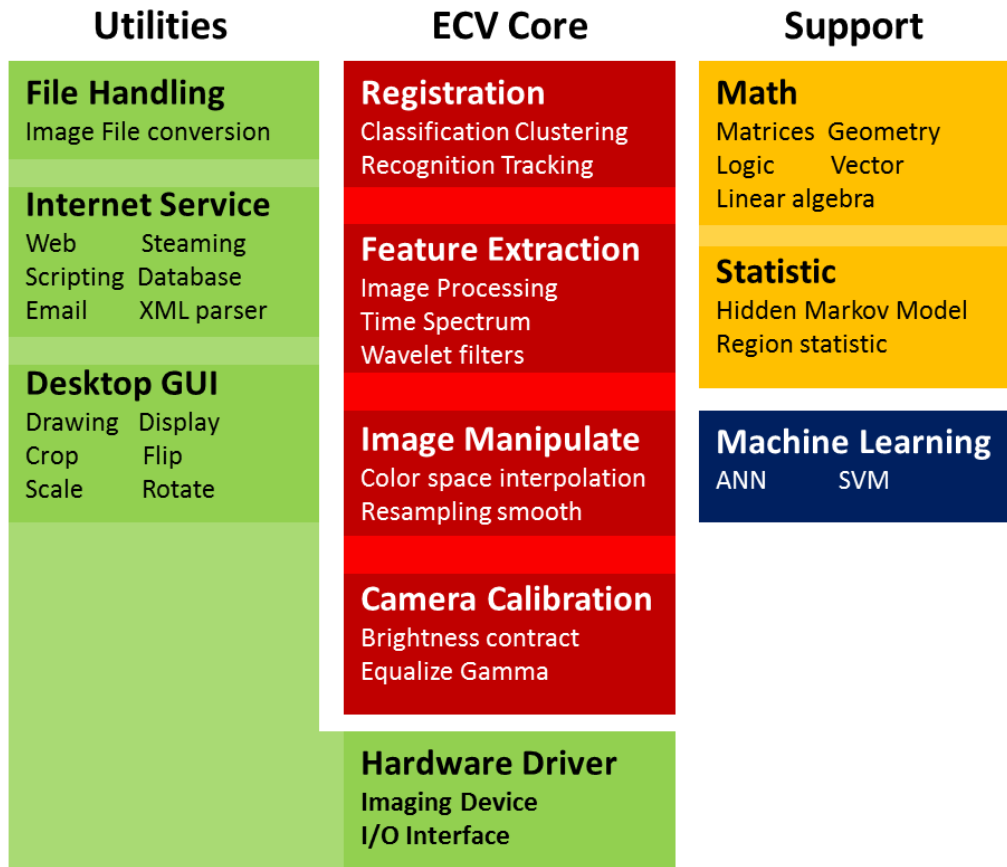


Figure 2.10 : ECV library layout and class diagrams

Figure 2.10 shows the possible ECV library layout. It covers most of the functions which can be found in open source CV library. There are 3 main groups of CV library classes:

1. Utilities - It is handling mostly operating systems task.
2. Core - Main CV library which generalize according to CV subsystems and CV typical task.
3. Support - It's fundamental function routine, such as math and machine learning algorithm.

The core class can be divided into a few sub-groups according to typical CV function and typical CV task. For instance, image acquisition function can group into hardware driver section, pre-processing function is grouped into image processing section, and feature extraction is grouped into registration section. Such also as feature extraction function is divided into 2D or three dimensional (3D) CV task.

ECV library is not intended to reinvent/rework any existing code, such as hardware driver and basic math library. Some of third party libraries are inherited to ECV library.

## **2.4 Summary**

When the processing speed is to be considered in an ES hardware design, the parallelism of the logic resources and coprocessor are the main aspects in this consideration. The inherent parallelism of the logic resources on the CPLD, ASIC, or FPGA allows for a considerable computation throughput at a sub-500 MHz clock rate. For example, the current (2007) generation of FPGA can implement around 100 single precision floating point units, all of which can compute a result every single clock cycle. The flexibility of the CPLD, ASIC, or FPGA provides a better performance by trading off the precision and range in the number format for an increased number of parallel arithmetic units. However modern coprocessor with DMA and SIMD capability is able to achieve similar high performance by trading off customized function block. Development of high performance embedded

systems consist of algorithm and hardware design.

ES development process is affected by the development cost, knowledge, and tools. This process involves CV application developer workforce and CPLD, ASIC, or FPGA engineers. The extra electronic design automation (EDA) software tools are required to do the hardware description language (HDL) process. Embedded processor with coprocessor is more preferable as CV application developer does not need to know the underlay of circuit design. Furthermore, comparing with high performance embedded systems, the total cost is much lower.

Propose a design of embedded computer vision platform that supports a wide range of CV applications with a shortest development cycle. The project will enable a touch sensing input device (Fingerprint reader, a capacitive touch sensing input device) and visible spectrum sensor device (Colour image sensor, a light spectrum sensor device) to be capable for wide range of CV applications. To achieve the shortest development cycle, all the CV algorithms will be implemented on top of embedded systems with coprocessor support. The emerging internet era, all the ES outcome, alert or result will be able to be accessed by user from web browser.

Continuous studies and researches have shown the possibility of grouping all the CV functions according to the typical CV task and the main group of CV library classes.



## CHAPTER 3

### EMBEDDED COMPUTER VISION DESIGN METHODOLOGY

#### 3.1 Introduction

This project methodology involves 3 major steps: start-up preparation, platform establishment by integration of sample applications, and research on ECV platform by benchmarking the performance.

Start-up preparation is to choose the suitable embedded systems board for research and then enable operating systems and several imaging devices.

Platform establishment is to apply common CV algorithm in ECV library through a few sample CV applications. Each sample CV application is broken down into smaller functions. Smaller functions are then intergraded into ECV library according to the library layout finding. Platform establishment takes place at personal computer (PC) platform and embedded systems platform. The sample CV application code is as generic as portable in PC platform, and it is optimized according to the specific executed instructions in embedded systems platform (Musser D. & Stepanov A., 1994). ECV library source code is written from scratch during the implementation of CV application.

Next research on ECV platform by benchmark the performance of the sample CV application is performed. Performance and algorithm optimization issue will then be studied.

### **3.2 Overall Methodology**

Through this study, it had identified six main variables/tasks. Thus, the main task needs to be completed for research methods are:

1. Apply CV algorithm in ECV library
2. Integration and optimization between ECV library and hardware
3. Sample CV application development
4. Determine the best layout of ECV library
5. Choose the suitable embedded systems board
6. Benchmark tools

The main task's interconnection is shown in Figure 3.1. It shows that the main focus of applying CV algorithm in ECV library is the determination of best layout to build the ECV library. The determination of best layout task is in the middle of other tasks. Any adjustment for any others task will affect this task.

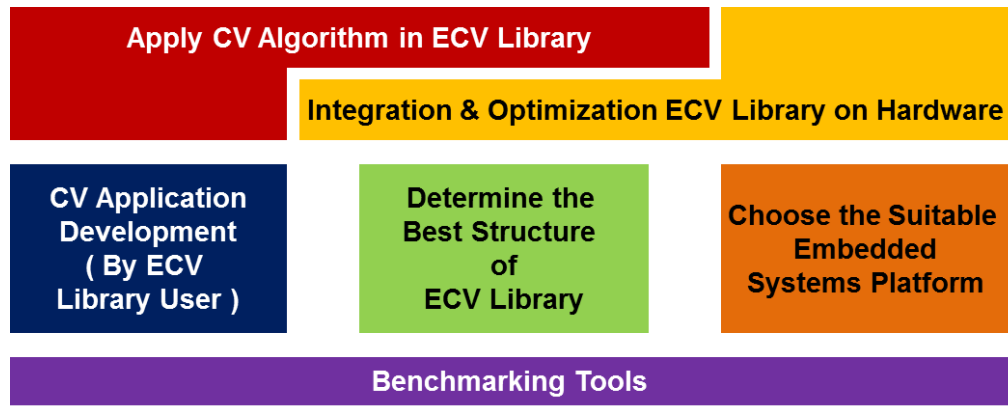


Figure 3.1 : Interconnection between project main tasks

Integration and optimization between ECV library and hardware take place after CV function is placed on ECV library layout. Integration and optimization step will depend on the chosen embedded systems while code optimization will depend on additional function that embedded systems had.

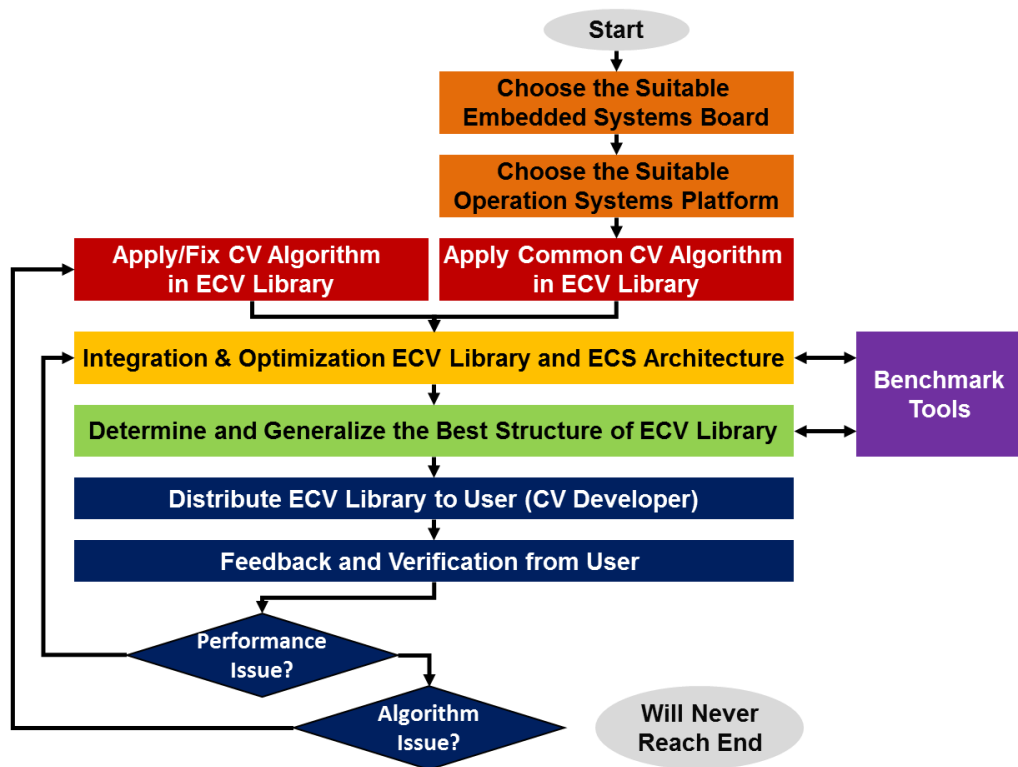


Figure 3.2 : ECV library project's working flow

Referring to Figure 3.2, choosing a suitable embedded systems board is the first step for this research project. Next, the multiple installation option is tested in order to determine the suitable operating systems and hardware driver. Several imaging device are also tested.

After step 1, the hardware and software-based systems are built. Step 2 is to apply a common/generic CV algorithm in ECV library onto the base systems and to check if the CV algorithm produces the same result for PC, and embedded platform.

When a CV algorithm is working, the optimization is performed to speed up the program in step 3. Step 4 is to simplify and generalize the functions into ECV library. Step 3 and step 4 are always monitored using a benchmark tools.

Step 5 is to distribute ECV library for developer to program CV application on top of it. We will go back to step 2 to fix/create new algorithm based on the developer feedback for any lack of functionalities, performance and algorithm. After several CV application developments, the whole research process is repeated for a few times. Literature review in Chapter 2 had help to identify these research problems as listed below:

1. Generalization of CV function
2. Determination of library layout
3. Optimization code in embedded systems

### **3.3 Specific Methodology**

This section explains specific methodology of this project. Literature review helps to identify this research problem and formulate the problem into procedure 4 sections:

1. Generalization of CV function
2. Application programming interface (API) design
3. Optimization code in embedded systems
4. Application runtime benchmark

#### **3.3.1 Generalization of CV Function into ECV Library**

This generalization process can preserve the information about the former level of specialization and to allow round-tripping between specialized and unspecialized forms of the same content. Generalization is important to simplify and to make sure the proper categorization of the software library. Image processing and interchange standard has been referred during designing of API (ISO/IEC Copyright Office, 1998) (Pratt & William K, 1996).

Generalization of CV function into software library form is a specific task for computer vision subject only. Referring to Chapter 2.3.1 and Chapter 2.3.2, CV function can be categorized according to typical CV function and typical task of CV. Both can be used to create a matrix map for CV software library.

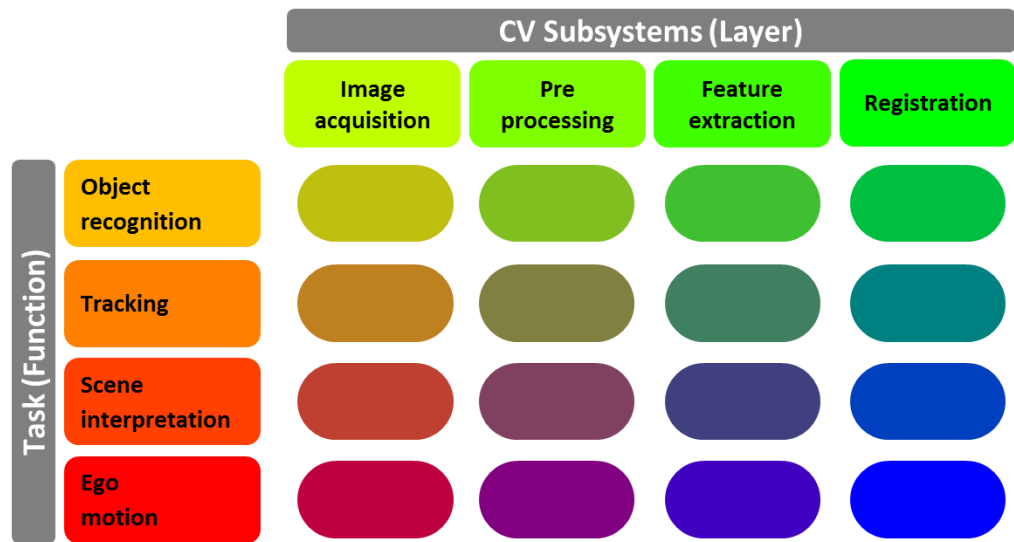


Figure 3.3 : Proposed matrix map for ECV library layout.

Figure 3.3 above is a proposed matrix map for ECV library layout. It is similar to Linux kernel matrix map. The Linux kernel matrix map consists of function versus user layer (Constantine Shulyupin, 2008). The matrix map is a guide for all the Linux developer to properly design their code in proper section of code (Daniel P. Bovet & Marco Cesati, 2005). Finding the ECV matrix map is a first method to generalize CV function according to the natural of CV subject (The CV function and CV typical task).

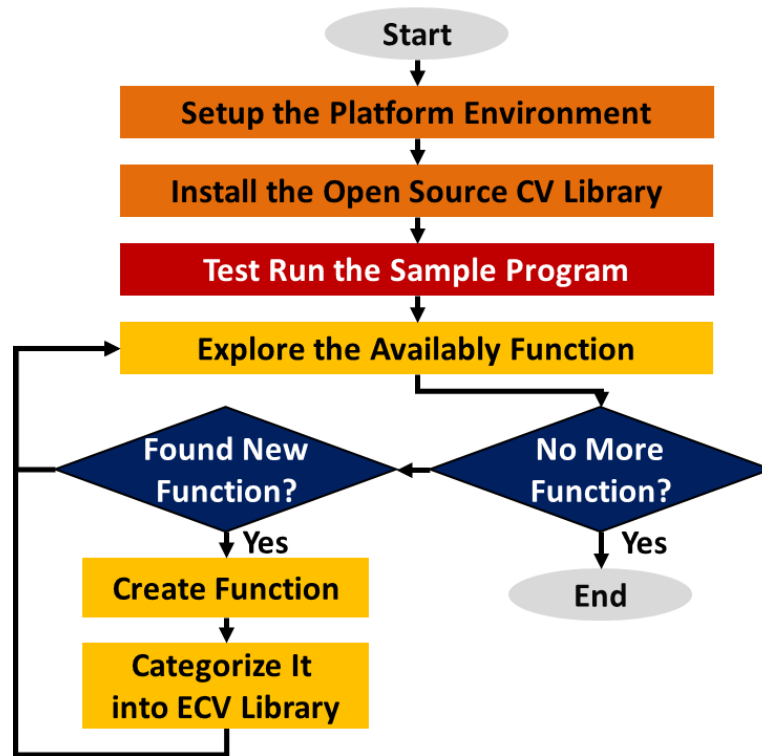


Figure 3.4 : Processes to examine of open source CV library layout

Second method is to study the generalization of CV function with the aim to study the current available open source CV library. Chapter 2.3.3 had done such studies. Figure 3.4 shows the process during the examination of open source CV library layout:

At Chapter 2.3.4, project found that commonly CV library function can be grouped into 3 main groups of CV library classes:

- Utilities - It is handling mostly operating systems task.
- Core - Main CV library which generalize according to CV subsystems and CV typical task.
- Support - It's fundamental function routine, such as math and machine learning algorithm.

Considering building ECV matrix map by combination of CV function, CV typical task and CV library classes, Figure 3.5 show the 3D matrix map for ECV library layout. It will be a guide for ECV developer during CV algorithm development, so that the piece of source code is in place accordingly.

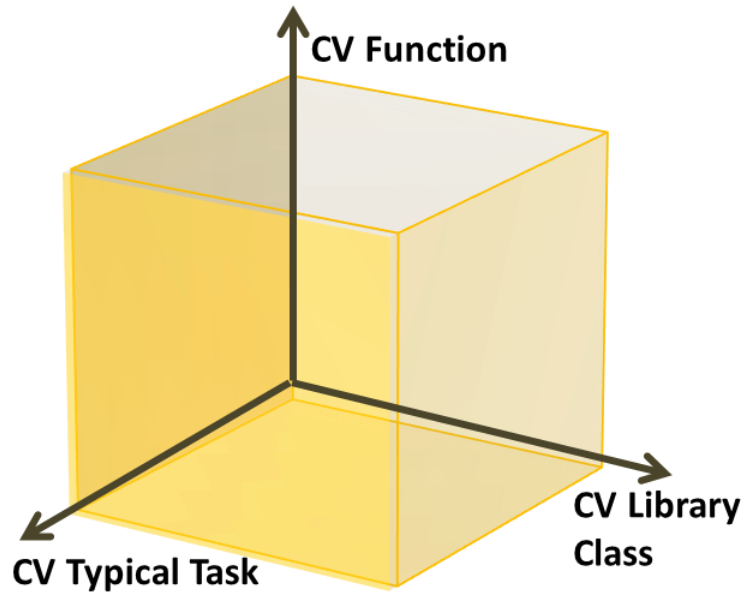


Figure 3.5 : Proposed 3D matrix map for ECV library layout

### 3.3.2 API Design

API design is about the function call of the ECV library. It is important as user will learn the library through API and a successful public APIs will capture user (David R. Hanson, 1997) (Joshua Bloch, 2006) and it can be among ECV greatest liabilities.



### Characteristics of a good API:

1. Easy to learn
2. Easy to use, even without documentation
3. Hard to misuse
4. Easy to read and maintain code that uses it
5. Sufficiently powerful to satisfy requirements
6. Easy to extend
7. Appropriate to audience

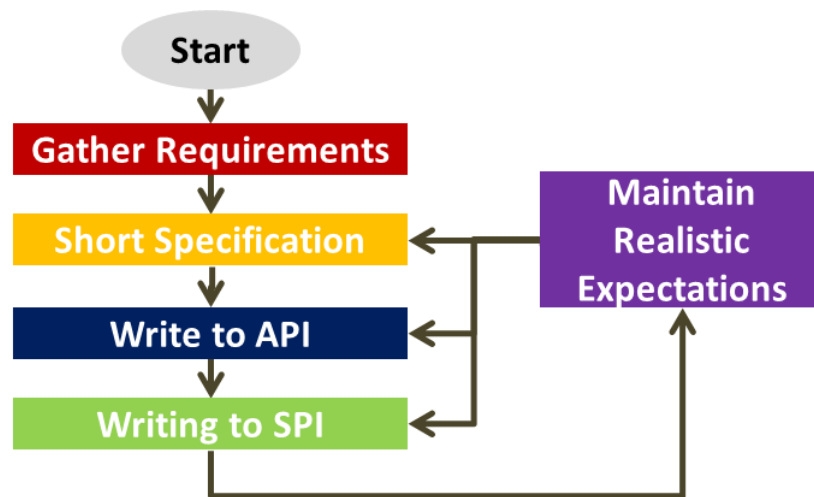


Figure 3.6 : The ECV library API design flow chart

Referring to Figure 3.6, the ECV library API design flow chart, the process of ECV library API design flow is executed in 5 steps. The API is always reviewed by repeating the design flow.

At first, gather all the requirements by conducting the required analysis at Chapter 3.3.1 in order to generalize the CV function. Next, we need to

conduct survey with user regarding their desired features.

Step 2 is to implement the functionality, design API, and release to test it. This way of API design tends to reflect the structure of the underlying code, rather than what the application programmer who will use the API.

Step 3, writing a few typical application snippet codes based on short specification list, as writing the code snippets, and the API takes shape. As implementing the API or write unit tests for the implementation, found many flaws or undefined corner cases in the original short specification design.

Finally, the API evolves by constantly repeating the API design flow. The API design flow is updated base on developer's feedback. It won't be able to please everyone, but it aims to displease everyone equally.

### **3.3.3 Code Optimization**

Choosing the fastest speed and highest accuracy CV algorithm can be a difficult task and there is no definitive comparison process for choosing the right one for ECV library. The code optimization process flow in Figure 3.7 shows that choice of programming language which reflect subsequent step in compiler optimization. And the correct choice of CV algorithm will dramatically help to speed up the code. Benchmark or profiling can be the performance determination. If there is a need to improve the benchmark results, hand-on code optimization can be done in inner looping code or

implementation of assembly code.

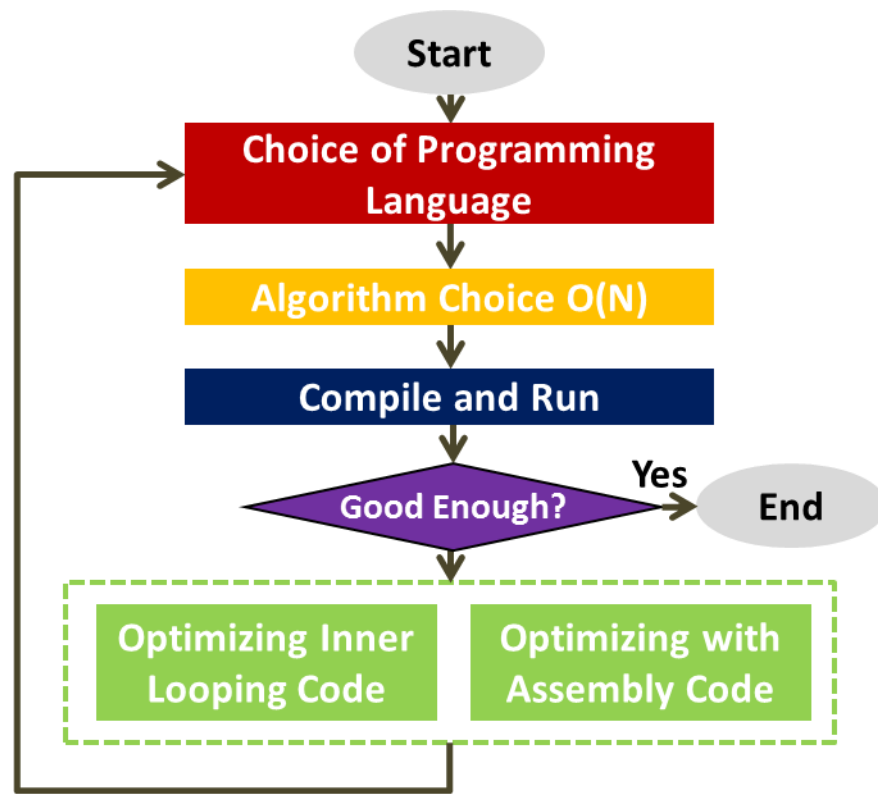


Figure 3.7 : Code optimization flow

By reducing an  $O(N^2)$  algorithm to  $O(N)$  algorithm, the program can be speeded up, especially for a large amount of data (Robert L. Kruse & Alex Ryba, 1998), such as image data. Table 3.1 showing number operations that would be performed for various values of  $N$  in algorithms analysis notation. Logarithms to base 2 (as used here) are proportional to logarithms in other base, so this doesn't affect the big-oh formula.

Table 3.1 : Comparison of time complexities versus number operations for an algorithms analysis notation

Operations n	Time Complexities					
	Constant Notation	Logarithmic Notation	Linear Notation		Quadratic Notation	Cubic Notation
	O(1)	O(log N)	O(N)	O(N log N)	O(N <sup>2</sup> )	O(N <sup>3</sup> )
1	1	1	1	1	1	1
2	1	1	2	2	4	8
4	1	2	4	8	16	64
8	1	3	8	24	64	512
16	1	4	16	64	256	4,096
1,024	1	10	1,024	10,240	1,048,576	1,073,741,824
1,048,576	1	20	1,048,576	20,971,520	1.1 E+12	1.15 E+18

For example, the camera images are captured, processed and displayed on the screen. The algorithms that do this must not be  $O(N^2)$ . If it took one microsecond (1 millionth of a second) to process each pixel, an  $O(N^2)$  algorithm would take more than a week to finish processing a 1 megapixel image, and more than three months to process a 3 megapixel image (note the rate of increase is definitely not linear).

The ECV library implementation was written in C/C++. This was a good choice because the GNU compiler collection (GCC) has an option that control code optimization. Turning on code optimization options makes the compiler attempt to improve the performance and/or program size at the expense of compilation time.

After settling on an algorithm choice and compiler optimization options, if the program still does not execute fast enough, there are a number of hand-on optimizations that can be performed. Optimizing inner looping code will reap the greatest benefit, because most of the processing time is spent in an inner loop. For code that needs to be extremely streamlined, assembly language is a good choice. C programming allows assembly code to be inserted directly into the code. It is also possible to write an entire section of code in assembly.

### **3.3.4 Application Runtime Benchmark**

Benchmarks provide a method of comparing the performance of EVC program in various optimizations across different system architectures. Benchmarking can be performed using 2 types of profiler (Thorsten Grotker et al., 2007):

1. Flat profilers compute the average call times, from the calls, and do not breakdown the call times based on the callee or the context.
2. Call graph profilers show the call times, and frequencies of the functions, and also the call-chains involved based on the callee. However context is not preserved.

Statistical profiler GNU gprof is used as flat profilers and call-graph profiler. Step by step to do benchmark for ECV library function are:

1. Compiling a program for profiling
2. Generate profile data
3. Analysis flat profile and call graph
4. Rewrite the critical path of program

Each ECV application is benchmarked in 2 platforms, the PC platform and embedded systems platform. Each sample CV application will be benchmarked 3 times using the combination below, as shown in Table 3.2:

- a. Code optimization of ECV application with microprocessor capability only
- b. Code optimization of ECV application with microprocessor and generic coprocessor capability only
- c. Code optimization of ECV application with microprocessor and specific coprocessor capability only

Table 3.2 : The platform combination for ECV application is benchmarking

ECV library	Combination 1	Combination 2	Combination 3
	Generic Compilation	Generic Compilation	Optimized Compilation
Software Block Illustration			
Hardware Involvement	With Microprocessor Only	With Microprocessor and Generic Coprocessor	With Microprocessor and Specific Coprocessor

### **3.4 Summary**

The study of CV and ES domain in Chapter 2 are helping to determine the overall project methodology. The overall project methodology helps to design the Gantt chart and time line.

A specific methodology is needed as a guide, because the overall project scope is too big. The specific methodology is helpful when meet some technical problem, it can be review all over the work flow to find the problem.

## **CHAPTER 4**

### **DEVELOPMENT OF EMBEDDED COMPUTER VISION PLATFORM**

#### **4.1 Introduction**

The development of ECV platform splits into hardware design, software design, ECV library design, and code optimization.

#### **4.2 Examination to Identify Suitability of Hardware**

This project aims to design an embedded computer vision platform for a wide range of CV applications within the duration of 2 years. Based on the analysis from Chapter 2, embedded systems with coprocessor were chosen as platform due to:

- a. Development cost, knowledge, and tools
- b. Processing speed

To determine the availability of ES board, an analysis on the core of embedded processor is first conducted. The market reliability among the chosen core processor is then checked. Determination of ES board is based on the criteria which are able to match the criteria of the suggestion in Chapter 2 as shown in Figure 4.1.



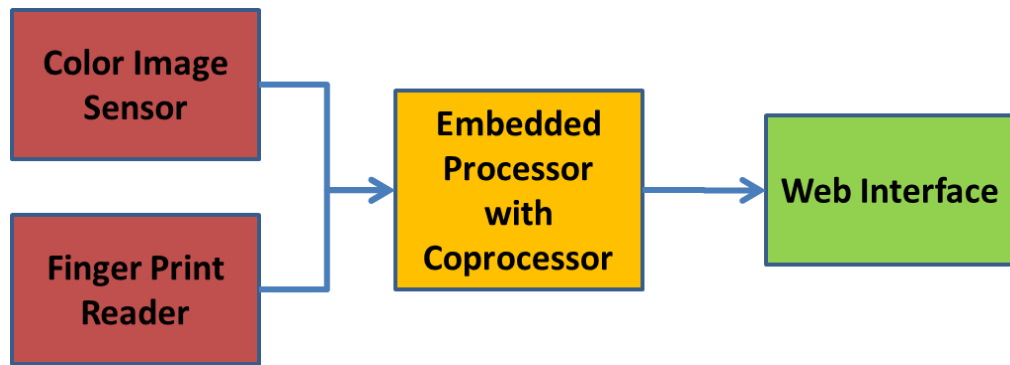


Figure 4.1 : Suggestion of embedded computer vision platform for this project

#### 4.2.1 Comparison of Core for Embedded Processor

Currently both Linux and Windows are battling among each other on the embedded systems platform (Jerry Krasner, 2007). This is due to the continuous embedded hardware growth which is currently stand at the aggregate rate of 14.2% and had reached USD 78.7 billion in 2009. As for the embedded board revenues, it will increase by an aggregate tare of 10% (Ravi Krishnan, 2005). There are many different types of CPU architecture which are currently used in embedded design where the most popular embedded processor architectures are ARM, MIPS, PowerPC, and x86 (Rick Lehrbaum, 2001).

To compare the core of embedded processor for general purpose is a tricky work. This is due to the difficulties in make an apple-to-apple comparison. The processor performance is more complicated than just CPU clock speed (Hertz) or CPU throughput (Dhrystone). Some vendors quote

maximum CPU clock speed, others quote typical CPU throughput while others do not include CPU performance at all. Some processors include useful system peripherals while others don't.

Based on an independent analyst (Linley Gwennap, 2005), they choose four general-purpose processors with clock speeds of between 400 and 600MHz. for the analysis. The four processors are:

1. Microprocessor without interlocked pipeline stages (MIPS) based reduced instruction set computer (RISC) processors
2. PMC-Sierra's popular RM7065C chip
3. Broadcom's low-end BCM1122
4. AMD's newest Alchemy Au1550

The analysis found that a new entrant of embedded processor from Raza Microelectronics, the XL5105 is pin-compatible with the RM7065C footprint, which means existing circuit can be upgraded easily. As most of the PowerPC chips in this segment are aging and uncompetitive, the analysis chose Freescale MPC8349. For variety, the analysis further included Intel's 80219 which is a general-purpose XScale processor, and Via's Eden C3, a low-cost x86 chip for embedded systems.

The analysis report states some low speed grades processor with system peripherals support are performed equally to high speed grades processor. Some useful system peripherals in embedded processor could boost

some of the CPU speed. Such useful system is DMA or cache memory. For example, the XScale chip has a scalar (single-instruction) CPU of level-two (L2) cache. So it used a 600MHz speed grade to boost the performance. Conversely, the Broadcom chip is a four-way super scalar CPU with 128kB of integrated L2 cache which should deliver a similar performance at 400MHz. The other processors are all at 500 or 533MHz. The analysis would have preferred to bump the speed of the AMD chip to 600MHz, making up for its scalar architecture, but that device is not yet available at the faster speed. The RM7065C, XL5105, and Eden C3 are all traditional standalone CPUs, whereas the other four processors integrate a memory controller, PCI interface, and other common peripherals. To even the comparison, the analysis also added Marvell's new Discovery LT companion chip into the two MIPS processors.

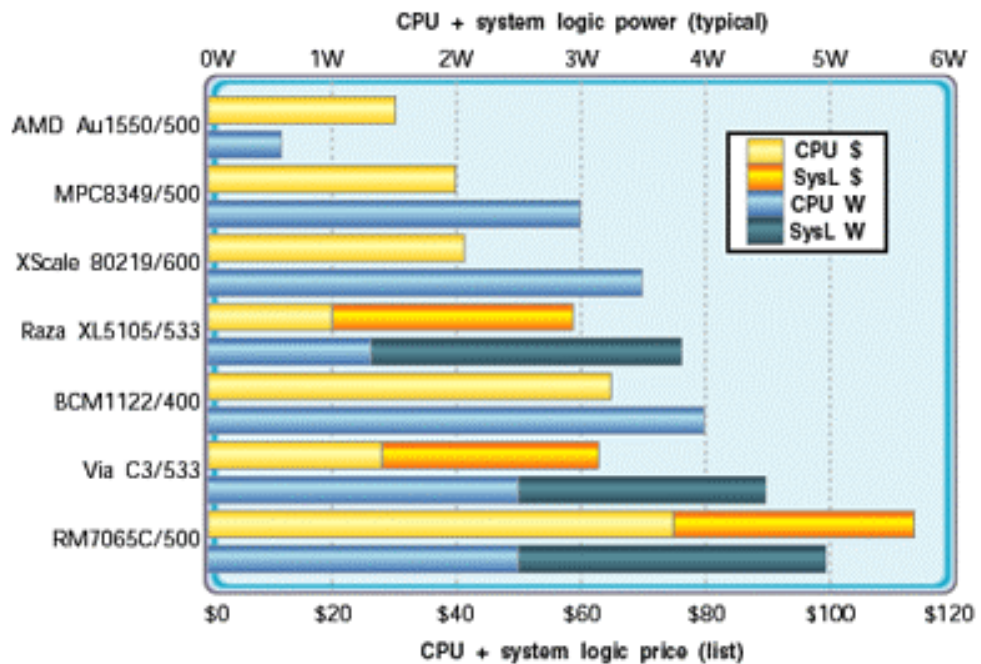


Figure 4.2 : Price and power comparison

The graph in Figure 4.2 shows the resulting comparison of price and power consumption for this group of processors. At USD 20 (list), the XL5105 is the least expensive of the group, but when the cost of the Discovery LT is added, the Raza product ends up costing nearly USD 60. Via's Eden C3 is similarly priced. The lowest-cost solutions are instead the integrated processors that don't require an external support chip. At USD 30, the Au1550 looks to be the best deal, but its performance is slightly worse than the others in this group. The MPC8349 and XScale 80219 offer a bit more performance and still cost much less than the others in this group.

The analysis report shows that integrated processors are becoming popular not just because they reduce board area but because they deliver real savings in system cost and power. Engineer looking for a processor in the 500MHz range should consider AMD's new Au1550, Intel's XScale and Freescale's MPC8349 embedded processor (Linley Gwennap, 2006). These three suggested embedded processor share a common characteristic where all of them are RISC based embedded processor.


Currently ARM processor accounts for approximately 75% of all embedded 32 bits RISC CPU (Jim Turley, 2002) and thus making it one of the most prolific 32 bits architectures in the world. The important branches in this family include Marvell's XScale and the Texas Instruments OMAP series. In January 2008, ARM achieves 10 billion processor milestones. Annual run rate now are three billion processor shipments across very diverse markets. (ARM Ltd, 2008). All data here had shown the availability and reliability of ARM

processor and therefore, ARM embedded processor will be the choice for this project.

#### 4.2.2 Variety of ARM Embedded Processor

ARM processor is better than other embedded processor in terms of size of code, run time, product reliability, and to perform more than a single instruction single data processing. We need a coprocessor which can perform a single instruction multiple data function to speed up image data (array of data) calculation. As we mentioned early in Chapter 2, computer vision is a chunk of array that perform almost the same operations for the whole junk of data.

Table 4.1 : Available ARM processor development board and the specification

Processor	Manufacture	Cirrus Logic	Intel	Texas Instruments
	CPU Model	EP9315	IXP425	TMS320C6202
	CPU Speed	200MHz	533MHz	600MHz
	ARM Core	ARM-9	XScale	OMAP
	Coprocessor	Maverick Crunch	Multiply-Accumulate	VelociTI
	MMU	Yes	Yes	Yes
Interface	Serial Port	Yes	Yes	Yes
	Ethernet	Yes	Yes	No
	Wireless	No	No	No
	USB Host	Yes	Yes	No
	Compact Flash	Yes	Yes	No
Hardware	Development Board	EP9315	VULCAN	dspModule
	Form Factor	PC104	PC104	PC104
	Price	USD 450	USD 995	USD 895
	Website	atmark-techno.com	arcom.com	rtd.com
Software	Development Tools	GCC	GCC	Code Composer Studio
	Price	Free	Free	USD 495
	Support Linux	Yes	Yes	Yes
	Support WinCE	Yes	Yes	No
Image				

Based on the cost factor and the lack of systems interface facility, the Texas Instruments Inc development board is not chosen. Instead, Cirrus Logic development board is chosen because of:

- a. USB host controller which enable wide range of plug-n-play device
- b. Free referent circuit design
- c. Low-cost

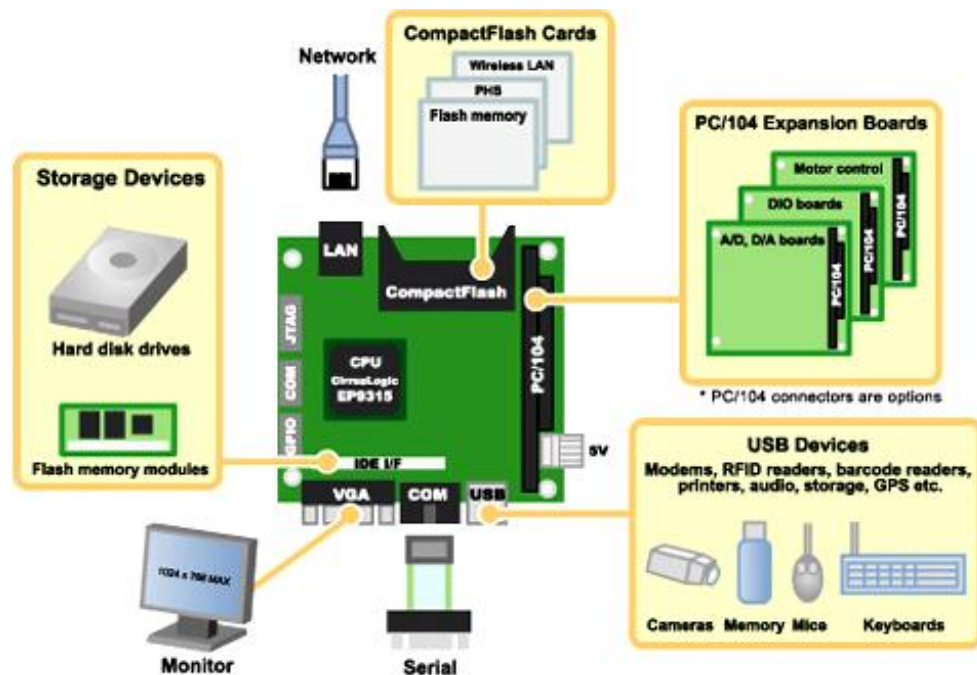


Figure 4.3 : Cirrus Logic EP9315 development board input/output

Cirrus Logic EP9315 is a SOC that contains ARM920T embedded processor, Maverick Crunch floating point math coprocessor and other input/output peripheral interface (Cirrus Logic, 2007). Please refer to Figure 4.3, EP9315 development boards I/O, where the development board contain:

- Compact flash
- USB host controller
- Serial communication
- VGA video output
- PC/104 expansion slot
- Mini integrated drive electronics (IDE) interface slot
- Random access memory (RAM)
- Fast Ethernet port

### 4.2.3 Coprocessor

The Maverick Crunch is a floating point math coprocessor core by Cirrus Logic, currently implemented in silicon alongside an ARM920T main core in their 200MHz EP9302 EP9307 EP9312 and EP9315 system on chip integrated circuits. Please refer to Figure 4.4.

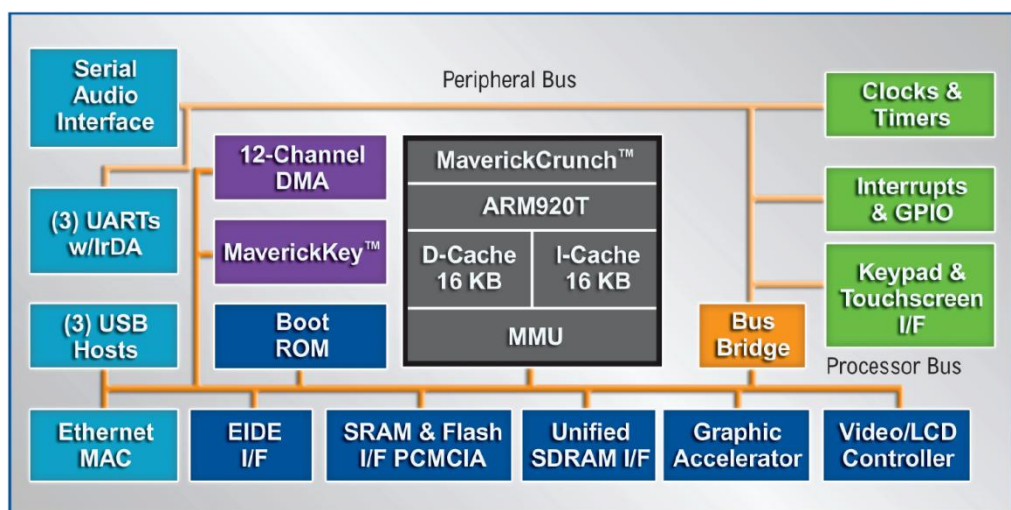


Figure 4.4 : Cirrus Logic EP9315 systems on chip processor

It has its own instruction set which implements 32 bits and 64 bits integer and IEEE-754 floating point addition, subtraction, multiplication, negation, absolute value and comparison, a set of multiply-and-accumulate functions, conversion between integer and floating point values, and instructions to move data between itself and the ARM registers or memory (Brett Davis, 2004). It has 16 x 64 bits registers, four 72 bits multiply-and-accumulate registers and a status register.

The coprocessor operates in parallel with the main processor, both processors receive their instructions from a single 32 bits instruction stream. Thus, to use it efficiently, integer and floating point instructions must be interleaved so as to keep both processors busy.

#### **4.2.4 Imaging Device**

As a suggestion from Chapter 2.2.1, a fingerprint reader and a colour image sensor will act as imaging input.

There are four types of finger print capture mechanism; optical, ultrasonic, passive capacitance, and active capacitance. However, this project will use active capacitance capture mechanism in the Microsoft fingerprint reader (model number DG2-00002). The Microsoft fingerprint reader is used with “libdpk”, an open source driver. It allows us to capture 400 x 200 resolutions of digital image of the fingerprint pattern.



Two most popular colour image sensor technologies today are charge-coupled device (CCD) and complimentary metal-oxide semiconductor (CMOS) where there are some noticeable differences between these sensors (Nicolas Blanc & Zurich, 2001):

- CCD sensors create high-quality, low-noise images.
- CCDs use a process that consumes more power.
- CCDs consume as much as 100 times more power than an equivalent CMOS sensor.
- CMOS chips can be fabricated on just about any standard silicon production line, so they tend to be extremely inexpensive compared to CCD sensors.



Figure 4.5 : Imaging device (From left: fingerprint, 2 of colour imaging sensor)

A Logitech and Z-star CMOS colour imaging sensor is chosen because of the availability of hardware driver. Both sensors can capture 620 x 480 of YUV colour image at 20 frames per second.

## 4.2.5 Interface Peripheral

To make the ECV systems possible as a standalone device, an input output controller is needed for light emitting diode (LED) blinking, switch buttons and servo motor. Microcontroller PIC18F4550 is used for this purpose. It can perform digital input, digital output and analogue input. Figure 4.6 is a schematic of the peripheral interface controller.

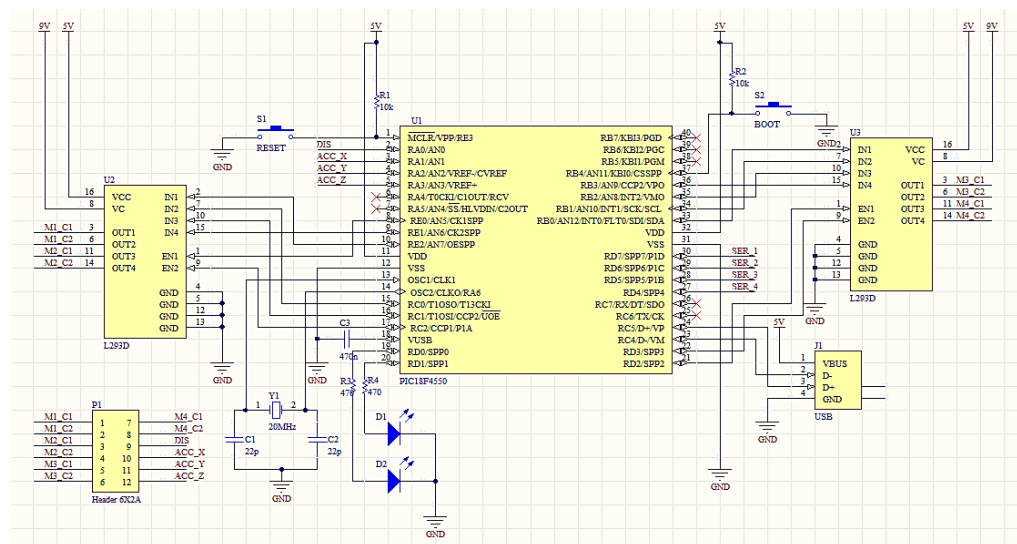


Figure 4.6 : Schematic of add on input/output peripheral interface controller

#### 4.2.6 Overall Hardware Design

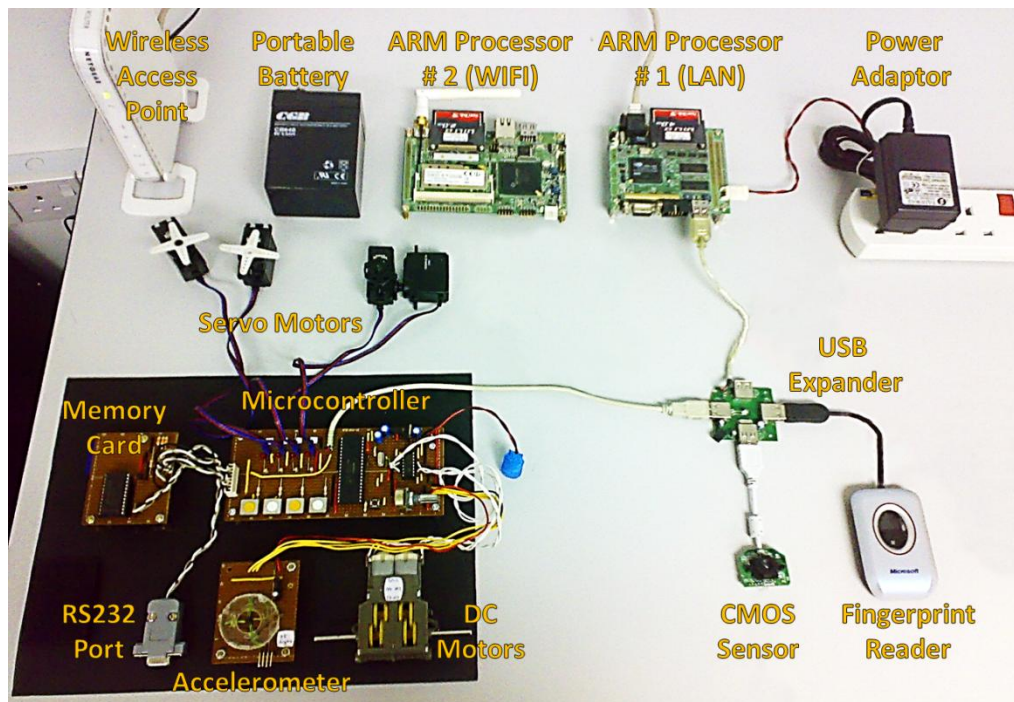


Figure 4.7 : Overall ECV platform hardware design

Figure 4.7 shows the overall embedded computer vision platform hardware design. The overall ECV hardware design specification is shown in Table 4.2.

Table 4.2 : The overall ECV platform hardware design specification

Development Board	
Processor	ARM-9 at 200MHz
Coprocessor	Maverick Crunch
RAM	64MB
Storage	1GB
Connection	Fast Ethernet
Input/output Peripheral	
Button	4
LED Indicator	4
Servo Motor	4

Colour Imaging Device	
Resolution	640 x 480
Frame per Second	25
Colour Level	32 bits
Fingerprint Imaging Device	
Resolution	394 x 289
Colour Level	8 bits

### 4.3 Examination to Identify Software Layout

#### 4.3.1 Operating Systems and Hardware Driver

In a design idea, in order to hide the underlay of hardware from CV developer, a CV library is provided for the developer. This CV library must be optimized with either an embedded processor or a coprocessor by the CV library maintainer. Figure 4.8 shows the basic software block diagram.

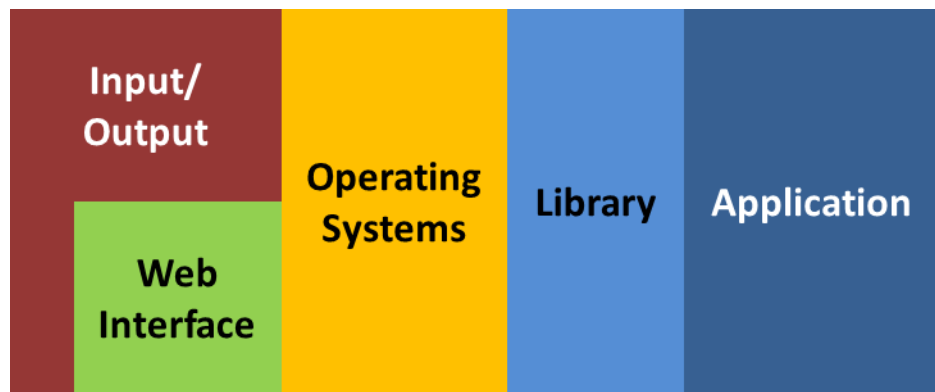


Figure 4.8 : Basic software block diagram

Study of the existing open source software is conducted to identify the software specifications that make up the software block diagram as shown in Table 4.3. In order for most of the open source software to execute on EP9315 development board a Linux distribution is installed.

Table 4.3 : Software package that work in EP9315 development board

Package		Software	Version
Operating Systems		Linux	v2.6.22
Utilities	File Sharing	samba	v3.0.26
Imaging Input Driver	Camera	gspca	v1.0.16
	Fingerprint Reader	libdppf	v0.2.1
General I/O Interface	Custom USB Driver	libUSB	v0.1.4

Result of the software engineering work on the EP9315 development board is shown in Table 4.3. The main challenge of this engineering work is to identify a small footprint of software which is able to run on the chosen embedded systems.

#### 4.3.2 Web Server Development

Web interface technology is studied and developed for this ECV platform. It will exclude physical output display and physical input switch from the hardware design. The development is aiming for minimum program footprint and processing power. In the evolution of web technology, today's rich interactivity of the web based interface can be created by a group of inter-related web development techniques. Examples of such group are:

- Adobe Flash, Adobe Flex and Adobe AIR
- Sun Java applets, Sun Java and JavaScript
- Microsoft ActiveX, Microsoft ASP and Microsoft Silverlight
- JavaScript and XML

Each group of the inter-related web development can be derived into few software components such as shown in Figure 4.9.

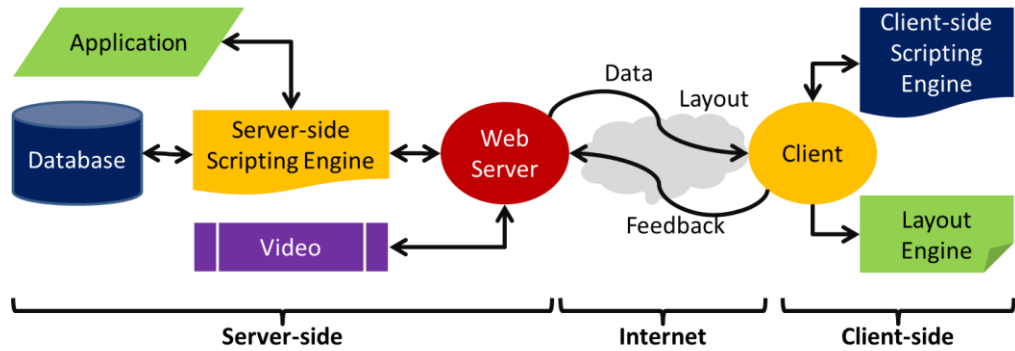


Figure 4.9 : Inter-related web server and client software component

Table 4.4 shows the example of software for each software component. In any combinations of the software component, there are standard web technologies which involve the following:

- Database communication protocol, such as SQL and LINQ
- Server-side scripting, such as JSP, PHP, ASP and LUA
- Client-side scripting, such as JavaScript, and VBScript
- Data representation format, such as XML, JSON and YAML
- Layout representation format, such as HTML, XHTML, CSS, and SVG
- Client feedback protocol, such as POST, CGI, and XMLHttpRequest

Table 4.4 : Inter-related web server and client component

<b>Component</b>		<b>Example of Software</b>
Server-side	Web Server	IIS, Apache
	Database Server	MySQL, Microsoft SQL
	Server-side Scripting Engine	PHP, ASP, LUA
	Video Server	IceCast, Apple Streaming Server
Client-side	Client-side Scripting	JavaScript, VBScript
	Layout Engine	Gecko, Trident, WebKit

According to W3School continuous survey (Refsnes Data, 2008), it shows that the most popular client side software (better known as internet browser) is the Internet Explorer and Firefox. Each of it gains 53.8% and 39.1% respectively for the internet browser's market shared by the end of April 2008. However, Firefox does not support VBScript or ActiveX, which are both proprietary to Microsoft. Therefore, JavaScript is the only possibility for achieving Internet Explorer and Firefox compatibility (Danny Goodman & Michael Morrison, 2004).

Referring to Figure 4.9, only the server-side component is needed in the embedded systems platform. Client side software will only require the Internet Explorer or Firefox internet browser for Windows and Unix platform.

Server-side scripting engine will be examined first before looking for a suitable web server that supports it. Database server is important to log and keep data that the web server needs to represent it later. Video server is an add-on tool to view the recorded video or live capture video from the embedded systems board. The main concerns during the evaluation of each internet server component are:

- Open source licensing and free to use
- Small size of source code or program
- Cross platform compilation
- Cross platform of client side software

#### 4.3.2.1 Server-side Scripting Engine

Table 4.5 is a comparison of server-side scripting engine based on fractal benchmark that is conducted (Erik Wrenholt, 2005). It shows that LUA is the fastest server-side scripting engine available.

Table 4.5 : Comparison of server-side scripting engine base on fractal benchmark

Language	Version	Time (Sec)	Relative Speed
C (Using GCC)	4.0.1	0.05	1.00 x
Java	1.4.2	0.40	8.00 x
LUA	5.1.0	1.50	30.00 x
Perl	5.8.6	21.75	435.00 x
PHP	5.1.4	23.12	462.40 x
Python	2.5.1	9.99	199.80 x
Ruby	1.8.4	34.31	686.18 x

Table 4.6 : Compares the size of server-side scripting engine

Language	Version	Source Code Size (kB)
LUA	5.1.0	830
Perl	5.10.0	65,910
PHP	5.2.6	63,960
Python	2.5.2	48,900
Ruby	1.8.7	21,180



Referring to Table 4.5 and Table 4.6, LUA is the fastest and most lightweight server-side scripting language. It consists of all the basic server-side scripting functionalities. In general, LUA strives to provide flexible meta-features that can be extended when needed, rather than supply a feature-set specific to one programming paradigm (Roberto Ierusalimschy, 2006). As a result, the base language is light. In fact, the full reference interpreter size is only about 150kB and easily adaptable to a broad range of applications.

#### 4.3.2.2 Web Server

Assessment found out that there are 48 open source web servers. However, only 26 of the open source web server software are using C/C++ as development language. The C/C++ development language enables the web server software for cross-platform compilation. Finally, it is found that there are only 9 of the C/C++ open source web server software that support CGI function. Table 4.7 lists out nine of the C/C++ open source web server software.

Table 4.7 : List of open source web server which using C/C++ as development language

No	Web server	License	Operating Systems	Source Code Size (kB)	LUA Support
1	Apache HTTP Server	Apache	Unix, Windows	31,610	Yes
2	Bozohhttpd	BSD Variant	Unix	32	-
3	Cherokee HTTP Server	GPL	Unix, Windows	686	Yes
4	Hiawatha	GPL	Unix, Windows	100	-

5	Mini httpd	BSD Variant	Unix	41	-
6	Null httpd	GPL	Unix, Windows	52	-
7	SHTTPD	BSD Variant	Unix, Windows	50	-
8	Thttpd	BSD	Unix	129	Yes
9	Xavante	GPL Variant	Unix	260	Yes

Table 4.7 also shows that C/C++ open source web server software with LUA support that has the least memory size are the Thttpd web server, Xavante web server and Cherokee HTTP web server.

#### 4.3.2.3 Database Server

Structured query language (SQL) is a recognizable standard language which designed to organize, manage, and retrieve data from a database. SQL is essentially a programming language for relational databases, and over the past decade SQL server has consistently delivered a reliable, scalable, cost-effective data management platform (C.J. Date & Hugh Darwen, 1996).

SQL provides a standard for database interoperability and LUA scripting support SQL interface. Hence, the database server must support relational databases since SQL is used as the database programming language in the miniature internet server.

Table 4.8 : List of open source database server

Database Server	License	Development Language	Platform
Apache Derby	Apache	Java	Unix, Windows
Firebird	IPL and IDPL	C++	Unix, Windows
H2	MPL	Java	Unix, Windows
HSQLDB	BSD	Java	Unix, Windows
Ingres	GPL and Proprietary	C/C++	Unix
MonetDB	MonetDB Public License	C/C++	Unix, Windows
MySQL	GPL and Proprietary	C/C++	Unix, Windows
PostgreSQL	BSD	C	Unix, Windows
SmallSQL	LGPL	Java	Unix, Windows
SQLite	Public Domain	C	Unix, Windows

Table 4.8 is a list of free and open source database server. C/C++ development language is required for platform portability and the small code size of database are important for miniature systems. Table 4.9 shows the analysis for the size of C/C++ database server.

Table 4.9 : List of open source database server using C/C++ as development language

Database	Version	Source Code Size (kB)
Firebird	2.1.0	86,800
Ingres	2006 R2	39,000
MonetDB	Feb-08	343,240
MySQL	5.0.51b	110,870
PostgreSQL	8.3.3	74,153
SQLite	3.5.9	9,390

It was found that SQLite has a relatively smaller code size among others, with the basic SQL functionalities. This program uses SQLite's

functionality through simple function calls, which reduces latency in database access as function calls are more efficient than inter-process communication. The entire database is stored as a single cross-platform file on a host machine (Chris Newman, 2004). This simple SQLite architecture and the code size making it possible to be implemented in miniature internet server.

#### 4.3.2.4 Streaming Media Server

A streaming media server is a specialized application which runs on an internet server. It is able to allow user for streaming audio and video content from internet using media player or embedded media player inside the internet browser (Roy S. et al., 2003). Table 4.10 shows the available open source streaming media servers.

Table 4.10 : Open source streaming media server

<b>Streaming Media Server</b>	<b>Licence</b>	<b>Operating Systems</b>	<b>Language</b>	<b>Source Code Size (kB)</b>	<b>Video Format</b>
Darwin	APSL	Unix	C	31,846	Mpeg4, 3GP
FFmpeg	GPL	Unix	C	14,123	ASF, AVI, FLV
Flumotion	GPL	Unix	C, Python	8,400	OGG, WMV, FLV
FreeCast	GPL	Unix, Windows	Java	26,130	OGG
IceCast	GPL	Unix, Windows	C	4,240	OGG
PeerCast	GPL	Unix	C	840	OGG
VLC/VLS	GPL	Unix, Windows	C	85,280	Mpeg4, OGG, WMV, FLV

PeerCast and IceCast is the smaller code size streaming media server. However both of it only support OGG video format. Which the video format is not support by default in internet browser.

Most of the portable video format is FLV. Currently Adobe Flash is widely use, and is portable either in Windows systems or Unix systems and either on Desktop platform or mobile platform. Thus Flumotion server and FFmpeg server has the smaller code size streaming media server which supports FLV format.

The Flumotion server requires Python. Since the Python standard library is large and comprehensive, the Flumotion server is not suitable to be fitted into the embedded systems. Only FFmpeg has the smallest code size and it supports the FLV format.

#### **4.3.2.5 Development and Examination**

The search of miniature internet server will start from open source web server platform, which is known as LAMP. The acronym LAMP's full meaning is as follow (Eric Rosebrock & Eric Filson, 2004):

- 'L' for Linux, referring to the Linux operating system
- 'A' for Apache, the web server
- 'M' for MySQL, the database server
- 'P' for Perl, PHP or Python, the scripting programming languages

Table 4.11 : List of evaluate of combination of open source internet server

At-tempt	Web Server	Scripting Language Engine	Database Server	Streaming Media Server	Total Program Size
1	Apache (6,124 kB)	PHP (13,200 kB)	MySQL (117,000 kB)	PeerCast (999 kB)	137,323 kB
2	Thttpd (389 kB)	PHP (13,200 kB)	SQLite (1,298 kB)	IceCast (1,483 kB)	16,370 kB
3	Thttpd (389 kB)	LUA (352 kB)	SQLite (1,298 kB)	IceCast (1,483 kB)	3,522 kB
4	Thttpd (389 kB)	LUA (352 kB)	SQLite (1,298 kB)	FFmpeg (7,000 kB)	9,039 kB
5	Xavante (623 kB)	LUA (352 kB)	SQLite (1,298 kB)	FFmpeg (7,000 kB)	9,273 kB

For the first attempt, the LAMP open source web server platform is recreated with smaller streaming media server, PeerCast. It is later discovered that the PeerCast does not feature BitTorrent-like swarming; if a point node fails, all others in the tree are mute and dead.

For the second attempt, the web server and database server are stripped down to Thttps and SQLite respectively and the Streaming media server is upgraded to IceCast. It is discovered that the Thttpd-PHP module is an extreme stripped down port with nothing is installed but the Thttpd binary, which has the PHP interpreter statically linked in. However, it does not work if external PHP modules are needed. Note that Thttpd-PHP is limited in its performance, simply because all the PHP requests are serialized, which means that a new connection is only accepted if the former has finished. The IceCast server is working well in this setup and it is capable of streaming content as OGG over standard HTTP.

For the third attempt, the scripting language engine is stripped down from PHP to LUA, which reduces 97.33% of the program size. The LUA remains to include the basic scripting functionalities.

For the fourth attempt, the streaming media server is upgraded to FFmpeg for supporting the FLV format. As explained in section 4.3.2.4, FFmpeg has the smallest code size and it supports the FLV format for a variety of systems and platform.

For the last attempt, the web server is upgraded to Xavante as it has the maximum integration with LUA. Xavante web server is support LUA scripting language through WSAPI interface. WSAPI is an API that abstracts the web server from LUA web applications.

In Table 4.11, LAMP open source web server platform with PeerCast has 137.323 MB of program size and research shows that it can be stripped down to 9.273MB after the 5 attempts while remaining the basic functionalities.

Currently the development shows that internet server with web, scripting, database and media streaming feature are able to be setup in embedded system which is less than 10MB program footprint. Table 4.12 shows the miniature internet server software package.

Table 4.12 : Miniature internet server software package that work in ARM920T embedded system

Package		Software	Version
Internet Server	Web Server	Xavenda	v2.0.0
	Scripting Language	LUA	v5.1.2
	Database	SQLite	v3.5.9
	Video Streaming	FFserver	v3.0.0

#### 4.4 ECV Library Design

From Figure 4.10, the actual software block diagram is resented based on software list in Table 4.3. It describes that CV application is able to access systems memory and gain the processor optimization through the ECV library function call.

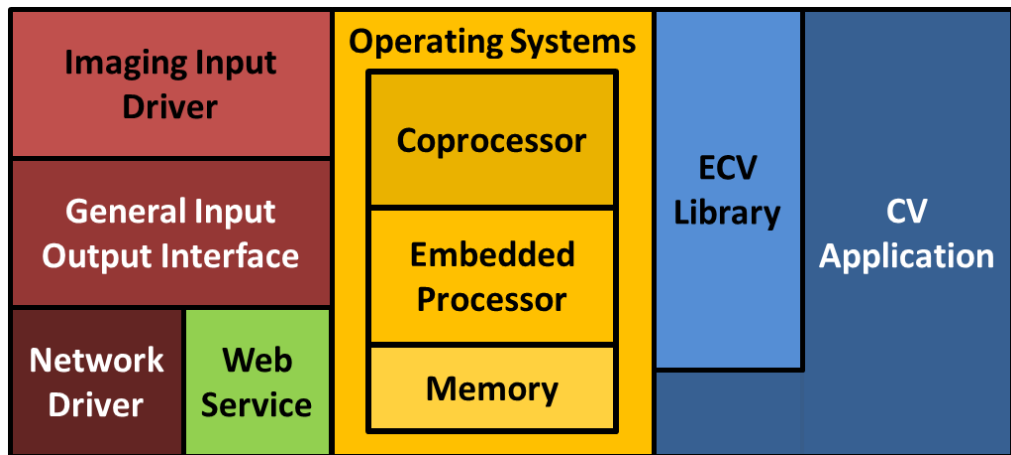


Figure 4.10 : Detail of software block diagram for embedded computer vision platform



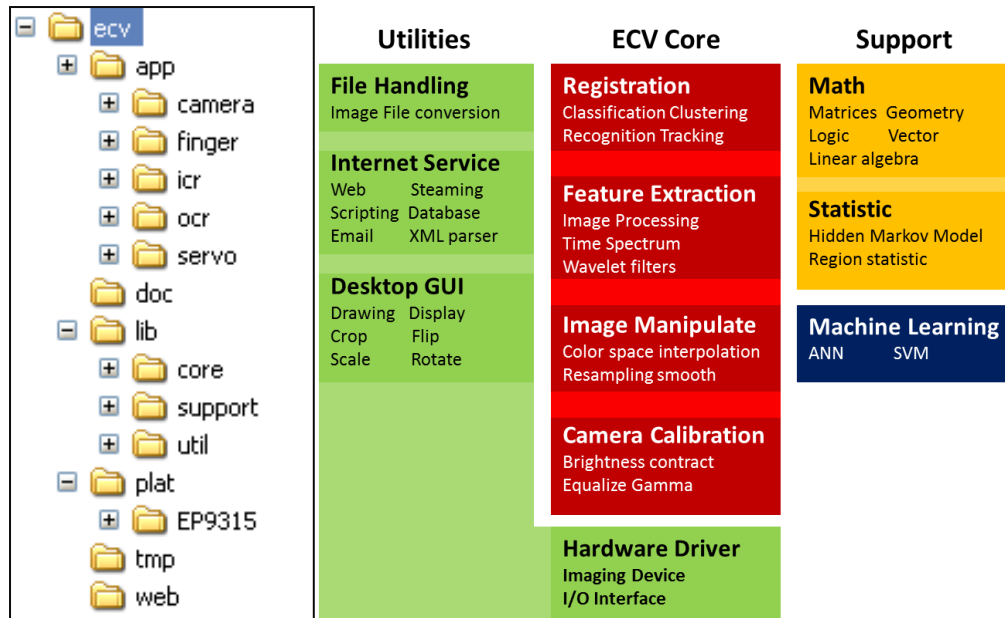


Figure 4.11 : ECV library file directory and the library layout

## 4.5 Code Optimization

### 4.5.1 Identifying the Critical Path

Profiler (GNU gprof) is used to learn on how the program spent its' time and which functions called which other functions while it is executing. This information is useful to identify the pieces of code which is slower and might be the candidates for rewriting and optimizing as to make it executes faster. It can also identify the function which has been called more or less often. This may help to spot unnoticed bugs (Brian J. Gough, 2004). Profiling has several steps which include:

- Compile and link the program with profiling enabled
- Execute the program to generate a profile data file

- Run GNU gprof to analyse the profile data

The result of the analysis is a file contains two listings, the flat profile and the call graph. The flat profile such as Figure 4.12 shows the total amount of time for the program spent executing each function. While the call graph such as Figure 4.13 shows detail of how much time was spent in each function and its children. The call graph also tells how many times that function was called. The GNU gprof profiler had been used to identify the:

- Section of code that are most frequently executed
- Section of code that take most CPU cycles to execute
- Inefficiencies in assembly code from the compilation

1.	Each sample counts as 0.01 seconds.						
2.	%	cumulative	self		self	total	
3.	time	seconds	seconds	calls	ms/call	ms/call	name
4.	33.34	0.02	0.02	7208	0.00	0.00	open
5.	16.67	0.03	0.01	244	0.04	0.12	offtime
6.	16.67	0.04	0.01	8	1.25	1.25	memcpy
7.	16.67	0.05	0.01	7	1.43	1.43	write
8.	16.67	0.06	0.01				mcount
9.	0.00	0.06	0.00	236	0.00	0.00	tzset
10.	0.00	0.06	0.00	192	0.00	0.00	tolower
11.	0.00	0.06	0.00	47	0.00	0.00	strlen
12.	0.00	0.06	0.00	45	0.00	0.00	strchr
13.	0.00	0.06	0.00	1	0.00	50.00	main
14.	0.00	0.06	0.00	1	0.00	0.00	memcpy
15.	0.00	0.06	0.00	1	0.00	10.11	print
16.	0.00	0.06	0.00	1	0.00	0.00	profil
17.	0.00	0.06	0.00	1	0.00	50.00	report

Figure 4.12 : Listing of flat profile output

```

1.  index % time self children called name
2.                                     <spontaneous>
3.  [1]   100.0  0.00  0.05                                     start [1]
4.                                     0.00  0.05  1/1  main [2]
5.                                     0.00  0.00  1/2  on_exit [28]
6.                                     0.00  0.00  1/1  exit [59]
7.  -----
8.                                     0.00  0.05  1/1  start [1]
9.  [2]   100.0  0.00  0.05  1  main [2]
10.                                     0.00  0.05  1/1  report [3]
11.  -----
12.                                     0.00  0.05  1/1  main [2]
13.  [3]   100.0  0.00  0.05  1  report [3]
14.                                     0.00  0.03  8/8  timelocal [6]
15.                                     0.00  0.01  1/1  print [9]
16.                                     0.00  0.01  9/9  fgets [12]
17.                                     0.00  0.00  12/34  strncmp[40]
18.                                     0.00  0.00  8/8  lookup [20]
19.                                     0.00  0.00  1/1  fopen [21]
20.                                     0.00  0.00  8/8  chewtime [24]
21.                                     0.00  0.00  8/16  skipSPACE [44]
22.  -----
23.  [4]   59.8  0.01  0.02  8+472  [4]
24.                                     0.01  0.02  244+260  offtime [7]
25.                                     0.00  0.00  236+1  tzset [26]
26.  -----

```

Figure 4.13 : Listing of call graph output

#### 4.5.2 Compiler Optimization

The GCC-ARM compiler was studied in order to identify the optimization option for increasing the speed of code and the optimization option for reducing the code size. In this experiment, the option to increase speed will be our priority. Methods below are the GCC general compiler optimization list (Heiko Falk & Peter Marwedel, 2004):

- a. Peephole - Combining several instruction into one simple instruction
- b. Local - Optimizing instruction in a serial code
- c. Loop optimization
- d. Intraprocedural - Optimizes the way control and data are passed between procedures

Table 4.13 : GCC optimizations option and optimizations level

Optimization Option	Optimization Level				Optimization Option	Optimization Level			
	O1	O2	OS	O4		O1	O2	OS	O4
defer-pop	✓	✓	✓	✓	strength-reduce		✓	✓	✓
thread-jumps	✓	✓	✓	✓	rerun-cse-after-loop		✓	✓	✓
branch-probabilities	✓	✓	✓	✓	rerun-loop-opt		✓	✓	✓
cprop-registers	✓	✓	✓	✓	caller-saves		✓	✓	✓
guess-branch-probability	✓	✓	✓	✓	force-mem		✓	✓	✓
omit-frame-pointer	✓	✓	✓	✓	peephole2		✓	✓	✓
align-loops		✓		✓	regmove		✓	✓	✓
align-jumps		✓		✓	strict-aliasing		✓	✓	✓
align-labels		✓		✓	delete-null-pointer-checks		✓	✓	✓
align-functions		✓		✓	reorder-blocks		✓	✓	✓
optimize-sibling-calls		✓	✓	✓	schedule-insns		✓	✓	✓
cse-follow-jumps		✓	✓	✓	schedule-insns2		✓	✓	✓
cse-skip-block		✓	✓	✓	inline-functions				✓
gcse		✓	✓	✓	rename-registers				✓
expensive-optimizations		✓	✓	✓					

Table 4.13 is the common set of GCC compiler optimizations option and the optimizations levels. By default each optimizations level includes certain optimizations option, where the included option is marked in the optimizations level column (William Von Hagen, 2006). Optimization level 1 let the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time. Optimization level 2 performs nearly all supported optimizations that do not involve a space-speed trade-off and does not perform loop unrolling or inline function. Optimization level S enables optimizations option that does not typically increase code size and performs further optimizations designed to reduce code size. Optimization level 3 turns on more expensive optimizations, such as function inlining, predictive-commoning and tree-vectorize in addition to all the optimizations option to increase the speed, but it can also increase the program size. In this project, optimization level 3 is a default compiler setting because the fast executes runtime is the concern to run CV application

in real-time.

### **4.5.3 Code Optimization without Coprocessor**

The sample CV application has generic source code and optimized source code. The generic source code means it contains no perform code optimization. The optimized source code was performed using the method below:

- Avoid redundancy - Store computations rather than re-computing them
- Serialize code - Minimum amount of branching, code branching is expensive
- Code locality - Code executed closely together in time should be placed closely together in memory, increasing spatial locality of reference and reducing expensive cache misses

## CHAPTER 5

### SAMPLE ECV APPLICATIONS

#### 5.1 Introduction

In this chapter, a sample application to prove the basic functionality of ECV platform and 5 samples of CV application to demonstrate the workable ECV library were conducted.

#### 5.2 Sample ECV Application

Sample application is chosen based on the finding of potential optimization part. For each sample application, the algorithm is first studied and then a generic sample program is integrated in ECV platform. Profiling tools is then used to identify the runtime of the code.

Sample application with potential optimization part is improved with optimization method. Five samples of CV application which was chosen to be integrated and optimized on the ECV platform are as shown below:

1. Motion detection
2. Face detection
3. Fingerprint matching

4. Optical character recognition
5. Intelligent character recognition

### **5.2.1 Basic ECV Platform Test Program**

The basic ECV platform shows the ability of ECV platform can be controlled through integrated mini internet server which is able to:

1. Capture image from camera and finger printed device
2. Receive and transmitted digital signal (LED light output and switch input)
3. Control multiple servo motor
4. Allow user to configure the ECV platform

Figure 5.1 shows the web interface of the ECV platform by using internet browser. The webpage is built from LUA server script, HTML markup language, and JavaScript. Section 4.3.2 clarifies more about this mini internet server.



Figure 5.1 : Web interface of the ECV platform

## 5.2.2 Motion Detection Test Program

It is a simple algorithm for motion detection by a fixed camera. It compares the current image with a reference image and simply counts the number of different pixels. Since images will naturally differ due to factors such as varying lighting, camera flicker, and charge coupled device (CCD) dark currents, pre-processing is useful to reduce the number of false positive alarms. Figure 5.2 shows the web interface for the motion detection test program in ECV platform.





Figure 5.2 : Motion detection application from the ECV's web interface

The profiling found the intensive part of motion detection in below function:

- Image thresholding during pre-processing
- Comparing two images using “for” loop
- Summation up array data
- Replicate image buffer

### 5.2.3 Face Detection Test Program

Face detection is a computer technology that determines the locations and sizes of human faces in arbitrary digital images (Shahrin Azuan Nazeer et al., 2007). It detects facial features and ignores anything else, such as buildings, trees and bodies. Face detection can be regarded as a specific case of object-class detection where it finds the locations and sizes of all objects in an image that belong to a given class. This face-detection application focused

on the detection of frontal human faces, whereas newer algorithms attempt to solve the more general and difficult problem of multi-view face detection. That is, the detection of faces that are either rotated along the axis from the face to the observer (in-plane rotation), or rotated along the vertical or left-right axis (out-of-plane rotation), or both. Figure 5.3 shows the web interface for the face detection test program in ECV platform.



Figure 5.3 : Face detection application from the ECV's web interface

The main intensive part of face detection application is in the ANN classification function.

#### 5.2.4 Fingerprint Matching Test Program

Among all the biometric techniques, fingerprint-based identification is the oldest method which has been successfully used in numerous applications.

Everyone is known to have a unique, immutable fingerprint. A fingerprint is made of a series of ridges and furrows on the surface of the finger. The uniqueness of a fingerprint can be determined by the pattern of ridges and furrows as well as the minutiae points (Anil Jain et al., 2001). Minutiae points are local ridge characteristics that occur at either a ridge bifurcation or a ridge ending. Fingerprint matching techniques can be placed into two categories techniques:

- a. Minutiae-based techniques first find minutiae points and then map their relative placement on the finger.
- b. Correlation-based techniques are able to overcome some of the difficulties of the minutiae-based approach.

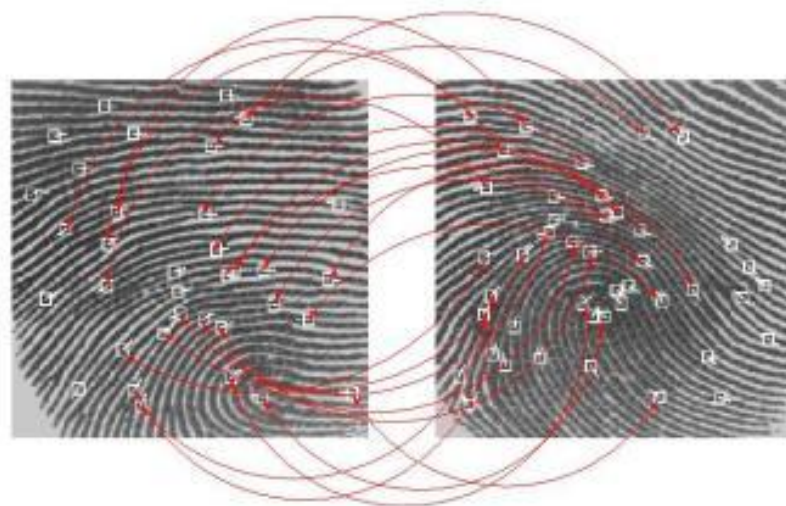


Figure 5.4 : Fingerprint matching illustration

This sample application will only focus on minutiae-based techniques, and this method does not take into account the global pattern of ridges and

furrows. The Figure 5.5 shows the web interface for the fingerprint matching program in ECV platform. The process of loading matching image from database and fingerprint matching processing consume idle time and CPU processing time.

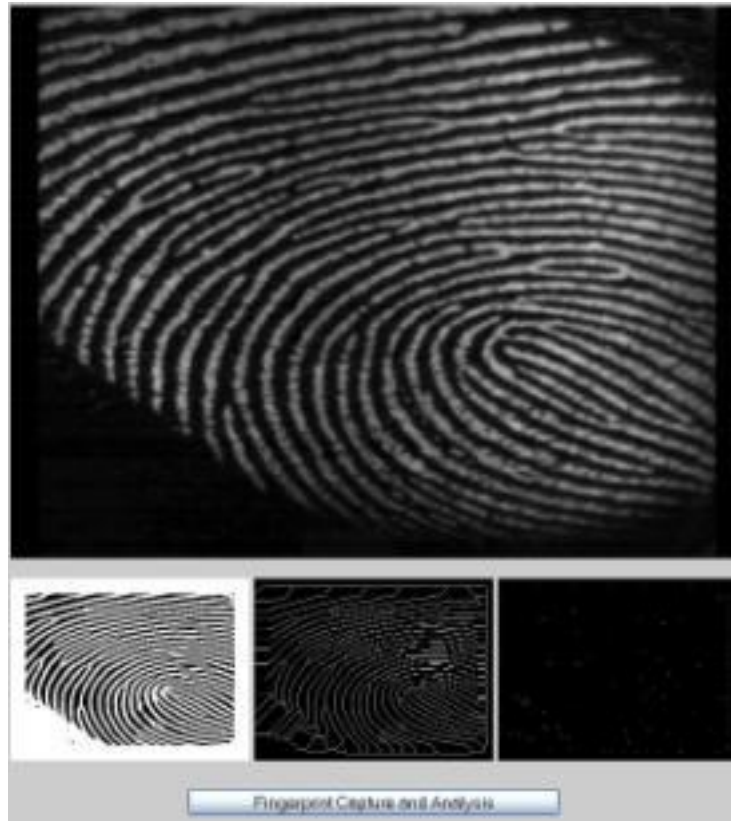


Figure 5.5 : Fingerprint matching application from the ECV's web interface

### 5.2.5 Optical Character Recognition Test Program

Optical character recognition (OCR) refers to a process whereby printed documents are transformed into ASCII files. This optical character recognition application is achieved by training a neural network (Mani N. & Srinivasan B., 1997), feed in an image, segmenting the image to detect a pattern, pre-processing the detected pattern, and applying the pre-processed

detected pattern to the trained neural network. The pre-processing includes determining a centroid of the pattern and centrally positioning the centroid in a frame containing the pattern. The training of the neural network includes randomly displacing template patterns within frames before applying the template patterns to the neural network.

Figure 5.6 shows the web interface for the OCR application in ECV platform. The training process of neural network is not run on ECV platform. The classification process of neural network is improved on ECV platform.

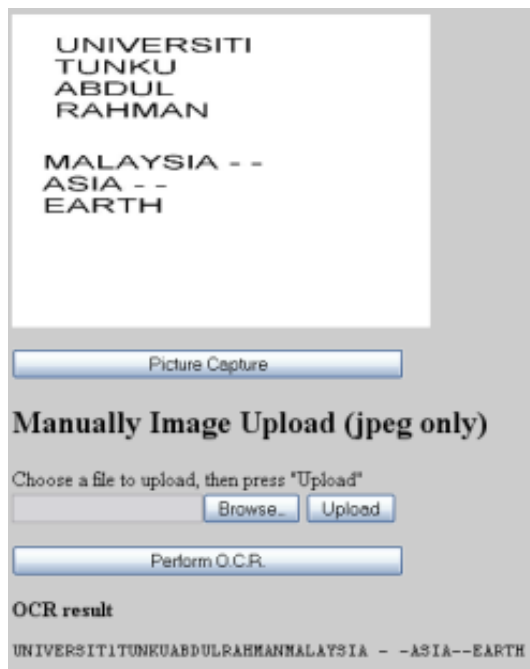


Figure 5.6 : Optical character recognition application from the ECV's web interface

### 5.2.6 Intelligent Character Recognition Test Program

Intelligent character recognition (ICR) is an advanced version of OCR

which is used to enhance the accuracy in recognition levels. It is a handwriting recognition system that allows fonts and different styles of handwriting. It extends the usefulness of scanning devices for the purpose of document processing, from printed character recognition to hand-written matter recognition.

Figure 5.7 shows the web interface for the ICR application in ECV platform. The training process of neural network is not run on ECV platform. Profiling found that the classification process of neural network and character segmentation is intensive and it is improved on ECV platform.



Figure 5.7 : Intelligent character recognition application from the ECV's web interface

## CHAPTER 6

### CONCLUSION AND FUTURE WORK

#### 6.1 Introduction

The entire tested CV application program for analysis is run on the same ECV platform with the same processor and coprocessor. The critical path of CV application program is identified by using a profiler. Next the code optimization is conducted and this method is explained in Chapter 4.5 “Code Optimization” and Chapter 4.5.2 “Compiler Optimization”. Computer vision applications are then compiled into 3 platforms for testing. These configured platforms on the same processor are:

1. Microprocessor only platform
2. Microprocessor with generic coprocessor platform
3. Microprocessor with specific coprocessor platform (Maverick Crunch coprocessor)

At the end of this experiment, there were 5 computer vision applications which were tested. For higher accuracy, this test was repeated 10 times or more in order to obtain the largest timing.

This project has successfully demonstrated a working embedded computer vision platform by using “Embedded with coprocessor” platform within a circumscribed development timeline. Many developers might still have concern regarding the CV algorithm on whether it needs more computational power than that the embedded systems platform can provide. However this project shows that Cirrus Logic embedded processor is able to perform several typical CV application task with optimization with coprocessor.

This project provides a learning ground for ECV development in:

1. Hardware circuit design
2. Building a Linux systems on embedded system
3. Build standard ECV library and the API
4. Code optimization with specific coprocessor

## **6.2 Results**

Table 6.1 to Table 6.5 show the experiment results of 5 CV tests application. The CPU runtime is the average of 10 experiments.



Table 6.1 : Motion detection test result

Compilation Setting		CPU Run Time (Second)	
		Default Code	Code Optimization
gcc-4.1.2	none (emulated FPU)	108	53
gcc-4.2.1	-mcpu=ep_9315 -mfast-math	12	5
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	7	2
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp -Os	8	3
gcc-4.1.2-cirrus	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	6	2
gcc-4.2.0	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	8	3

Table 6.2 : Face detection test result

Compilation Setting		CPU Run Time (Second)	
		Default Code	Code Optimization
gcc-4.1.2	none (emulated FPU)	425	376
gcc-4.2.1	-mcpu=ep_9315 -mfast-math	48	33
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	16	13
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp -Os	17	14
gcc-4.1.2-cirrus	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	16	13
gcc-4.2.0	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	17	13

Table 6.3 : Fingerprint matching test result

Compilation Setting		CPU Run Time (Second)	
		Default Code	Code Optimization
gcc-4.1.2	none (emulated FPU)	768	489
gcc-4.2.1	-mcpu=ep_9315 -mfast-math	78	47
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	31	20
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp -Os	32	21
gcc-4.1.2-cirrus	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	30	20
gcc-4.2.0	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	32	20

Table 6.4 : Optical character recognition test result

Compilation Setting		CPU Run Time (Second)	
		Default Code	Code Optimization
gcc-4.1.2	none (emulated FPU)	138	117
gcc-4.2.1	-mcpu=ep_9315 -mfast-math	13	11
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	5	3
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp -Os	6	4
gcc-4.1.2-cirrus	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	5	4
gcc-4.2.0	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	5	4

Table 6.5 : Intelligent character recognition test result

Compilation Setting		CPU Run Time (Second)	
		Default Code	Code Optimization
gcc-4.1.2	none (emulated FPU)	293	245
gcc-4.2.1	-mcpu=ep_9315 -mfast-math	31	28
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	13	11
gcc-4.1.2	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp -Os	14	11
gcc-4.1.2-cirrus	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	13	11
gcc-4.2.0	-mcpu=ep_9315 -mfpu=maverick -mfloat-abi=softfp	13	11

### 6.3 Analysis

GCC compiler version 4.1.2, version 4.2.0, and version 4.2.1 are available to compile the CV application for this arm processor (with Maverick Crunch coprocessor). Based on each CV application experiment result, it was found that different compiler version will produced a slight difference of CPU runtime for all CV programs. Hence, the compilation setting is sum-up as:

- GCC compilation without enabling any FPU option. It means the code only run on the main processor, in this case ARM processor.
- GCC compilation by enabling common FPU option (“gcc -mcpu=ep\_9315 -mfast-math”). It means the code is for a generic FPU coprocessor.
- GCC compilation by enabling specific FPU option (“gcc -mcpu=ep\_9315 -mfpu=maverick -mfloat-abi=softfp”). It means the code is the best optimization for this specific FPU coprocessor, in this case ARM-9 processor with Maverick Crunch coprocessor.

Table 6.6 : Summary table of all test application CPU run time in all type of compilation setting

Compilation Setting	CPU Run Time (Second)									
	1.Motion Detection		2.Face Detection		3.Finger -print Matching		4.Optical Character Recognition		5.Intelligent Character Recognition	
	Default Code	Code Opt	Default Code	Code Opt	Default Code	Code Opt	Default Code	Code Opt	Default Code	Code Opt
Without FPU	108	53	425	376	768	489	138	117	293	245
With FPU	12	5	48	33	78	47	13	11	31	28
With Specific FPU	6	2	16	13	30	20	5	3	13	11
Improvement “Without FPU” Over “With FPU”	9.00	10.60	8.85	11.39	9.85	10.40	10.62	10.64	9.45	8.75
Improvement “With FPU” Over “With Specific FPU”	2.00	2.50	3.00	2.54	2.60	2.35	2.60	3.67	2.38	2.55

## 6.4 Conclusion

Based on Table 6.6, we conclude that:

- a. Average improvement “without FPU” over “with FPU” is 9.96 times.
- b. Average improvement “with FPU” over “with specific FPU” is 2.62 times.
- c. The overall average improvement “without FPU” over “with specific FPU” is 26.07 times.

The code optimization and compiler optimization, had achieved:

- a. The code optimization for the Improvement “without FPU” over “with FPU” shows 81% improvement.
- b. The code optimization for the Improvement “with FPU” over “with specific FPU” shows 20% improvement.
- c. The overall code optimization had 50.5% improvement.

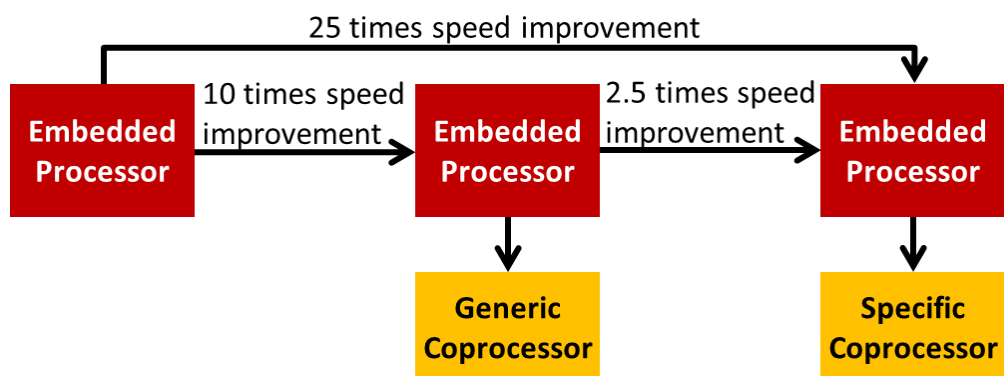


Figure 6.1 : Illustrated of speed improvement between 3 platforms

Embedded systems with specific coprocessor can dramatically help to

improve CV application runtime.

## **6.5 Future and Trend**

As mentioned in abstract, currently silicon chips are becoming more and more powerful despite the reduction in size. Various general processors, digital signal processors (DSPs) and graphic processing units (GPUs) were embedded in electronic gadgets. With the high processing power of these embedded platforms, it is a great opportunity to integrate CV applications to these relatively low-cost and standalone devices. Therefore, the ECV engineering will be the next major topic.

Figure 6.2 shows that convergence is happening in industry and consumer product (Zamani Zakariah, 2004), for example telecom industry and computer industry converge as new telecoms industry. From this research, we foresee that ECV industry will be the next converged industry between ES industry and emerging CV industry.

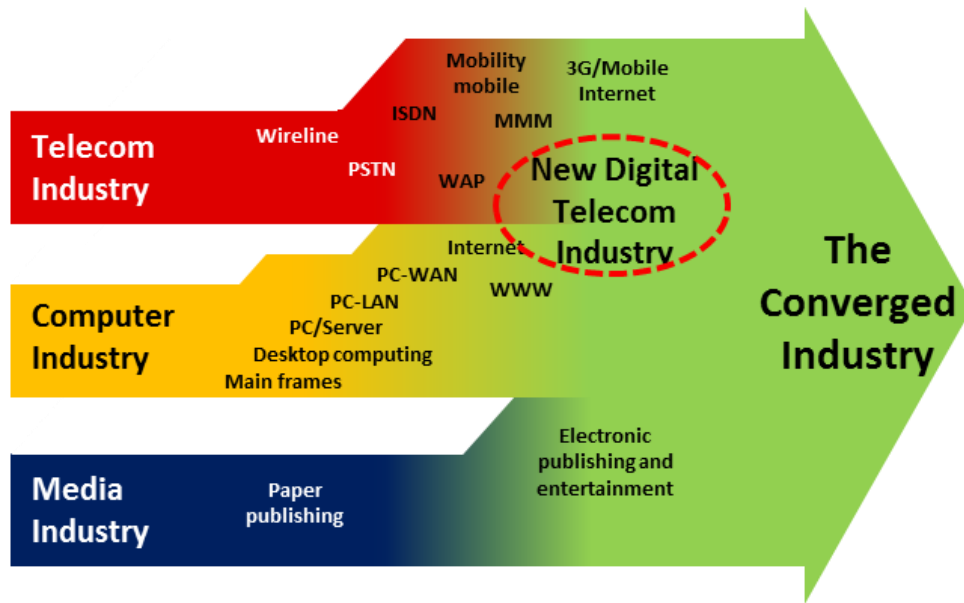


Figure 6.2 : Industry integration and convergence

## 6.6 Future Work

In the future, further studies can be conducted on an open API for computer vision algorithm so that SOC manufacture and ECV maintainer can provide optimized ECV library according to their SOC chip. The SOC industrial support and standard CV organization body are needed in order to serve two main purposes which are:

1. To hide the complexities of interfacing with different CV accelerators by presenting the programmer with a single and uniform interface.
2. To hide the differing capabilities of hardware platforms by requiring all implementations in order to support the full ECV feature set (Using software emulation if necessary).

Next, research on speed improvement between FPGA with coprocessor will be conducted. This research is crucial as it will justify whether a coprocessor platform is worth more than FPGA platform in terms of total development cost. In order to do so, we need to identify whether the equivalence FPGA and coprocessor can be determined according to:

1. Number of processing cell, or
2. Same speed of processing step, or
3. Number of transistor or else.

Among the future research and development for the project are:

1. Research on an open API for computer vision algorithm.
2. Research on total of cost between FPGA and coprocessor for justification.
3. Development on the ECV library so that it can fulfil near 100% function that all CV developer need.
4. Development to enhance current ECV platform as to ensure better integration of coprocessor with the embedded processor such as:
  - a. Single SIMD instruction which is able to perform with the size of a single image data.
  - b. Possibility of multi coprocessor on a single SOC chip.
5. Development on hardware peripheral so that no waiting time is spent on peripheral driver.

## REFERENCES

- Aleksandr Mitrofanov (August 15, 2006), *Dual-core Intel Conroe Processor*. Retrieved from <http://www.digital-daily.com/cpu/intel-conroe/print>
- Anil Jain, Arun Ross and Salil Prabhakar (Oct 7 - 10, 2001), *Fingerprint Matching Using Minutiae and Texture Features*. Proceedings of Int'l Conference on Image Processing (ICIP), Pages 282-285
- ARM Ltd (22 January 2008), *ARM Achieves 10 Billion Processor Milestone*. Retrieved from ARM Ltd. Website: <http://www.arm.com/about/newsroom/19720.php>
- Bernd Jahne and Horst HauBecker (2000), *Computer Vision and Applications A Guide for Students and Practitioners*. Retrieved from Academic Press. ISBN 0-13-085198-1.
- Binu Mathew, Al Davis, and Ali Ibrahim (2003), *Perception Coprocessors for Embedded Systems*. Retrieved from IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia (ESTIMEDIA 2003)
- Bir Bhanu and Ioannis Pavlidis (November 12, 2004), *Computer Vision Beyond the Visible Spectrum*. Retrieved from Springer. ASIN B000UJB8TI
- Brett Davis (Jan 2004), *Optimizing Code Speed for the Maverick Crunch Coprocessor*. Retrieved from Cirrus Logic Inc. Web site: <http://www.cirrus.com/en/pubs/appNote/AN253-1.pdf>
- Brian J. Gough (30 Mar 2004), *An Introduction to GCC - for the GNU Compilers gcc and g++*. Retrieved from Network Theory Limited. ISBN-10: 0954161793. Chapter 10.2 Using the profiler gprof.
- Chris Newman (November 19, 2004), *SQLite*. Retrieved from Sams Publishing. ISBN-10: 067232685X



- C.J. Date, and Hugh Darwen (November 18, 1996), *Guide to Sql Standard, 4 Editions*. Retrieved from Addison-Wesley Professional. ISBN-10: 0201964260
- Cirrus Logic (September 2007), *EP93xx User's Guide*. Retrieved from: [http://www.cirrus.com/en/pubs/manual/EP93xx\\_Users\\_Guide\\_UM1.pdf](http://www.cirrus.com/en/pubs/manual/EP93xx_Users_Guide_UM1.pdf)
- Constantine Shulyupin (2008), *Linux Kernel Map*. Retrieved from [http://www.makelinux.net/kernel\\_map](http://www.makelinux.net/kernel_map)
- Dana H. Ballard and Christopher M. Brown (1982), *Computer Vision*. Retrieved from Prentice Hall. ISBN 0131653164.
- Daniel P. Bovet and Marco Cesati (Nov 2005), *Understanding the Linux Kernel, Third Edition*. Retrieved from O'Reilly Media. ISBN-10: 0596005652
- Danny Goodman and Michael Morrison (2004), *JavaScript Bible, 5th edition*. Retrieved from John Wiley and Sons Inc. ISBN 0-7645-5743-2. Chapter 47 Cross-browser dynamic HTML issues
- David C Brock (2006), *Understanding Moore's Law: Four Decades of Innovation*. Retrieved from Chemical Heritage Press. ISBN-10 0941901416
- David R. Hanson (1997), *C Interfaces and Implementations: Techniques for Creating Reusable Software*. Retrieved from Addison-Wesley Professional Computing Series. ISBN 0201498413
- Eric Rosebrock and Eric Filson (July 22, 2004), *Setting Up LAMP: Getting Linux, Apache, MySQL, and PHP Working Together*. Retrieved from Sybex Publisher. ISBN-10 0782143377

- Erik Wrenholt, (20 September 2005), *Ruby, Io, PHP, Python, Lua, Java, Perl, Applescript, TCL, ELisp, Javascript, OCaml, Ghostscript, and C Fractal Benchmark*. Retrieved from <http://www.timestretch.com/FractalBenchmark.html>
- Gosta H. Granlund and Hans Knutsson (1995). *Signal Processing for Computer Vision*. Retrieved from Kluwer Academic Publisher, ISBN 0-7923-9530-1.
- Graefe, V., Fleder K, and Fuer Messtech (28 Oct 1991), *A Powerful and Flexible Co-processor for Feature Extraction in a Robot Vision System*. Retrieved from University Der Bundeswehr Muenchen, Neubiberg, Germany, Proceedings of Industrial Electronics, Control and Instrumentation. ISBN: 0-87942-688-8. Pages: 2019 - 2024 vol.3
- Heiko Falk and Peter Marwedel (December 20, 2004), *Source Code Optimization Techniques for Data Flow Dominated Embedded Software*. Retrieved from Springer Publisher, ISBN-10: 1402028229
- Herb Sutter (March 2005), *The Concurrency Revolution*. Retrieved from C/C++ Users Journal. Issues 23(2), February 2005. Pages 8
- Ian Buck (June 2005), *Programming GPU for General Computing*. Retrieved from Graphics Laboratory, Stanford University. Web site [http://www.eetasia.com/ART\\_8800367818\\_480100\\_NT\\_221862cd.HTM](http://www.eetasia.com/ART_8800367818_480100_NT_221862cd.HTM)
- IEEE (June 25 2005), *The First IEEE Workshop on Embedded Computer Vision*. Retrieved from <http://computervisioncentral.com/sites/all/files/images/ecvw2005.pdf>
- ISO/IEC Copyright Office (1998), *Image Processing and Interchange Standard*. Retrieved from ISO/IEC Document 12087-5: Image Processing and Interchange Standard (1998)

- James Kurien, Xenofon Koutsoukos, and Feng Zhao (2002), *Distributed Diagnosis of Networked, Embedded Systems*. Proceedings of the 13th International Workshop on Principles of Diagnosis, 2002
- Jerry Krasner (December 2007), *Embedded Linux Total Cost of Development Analyzed*. Retrieved from Embedded Forecast Web site  
[http://embeddedforecast.com/images/Embedded\\_Linux\\_TCD\\_Analyzed\\_120507.pdf](http://embeddedforecast.com/images/Embedded_Linux_TCD_Analyzed_120507.pdf)
- Jim Turley (December 18 2002), *The Two Percent Solution*. Retrieved from Embedded Systems Design Web site  
<http://www.embedded.com/story/OEG20021217S0039>
- Joshua Bloch (2006), *How to Design a Good API and Why It Matters*. Retrieved from Dynamic Languages Symposium archive. Publisher ACM New York. ISBN: 1-59593-491-X, Pages: 506 - 507
- K. Rath, and P.K.Meher (2006), *Design of a Merged DSP Microcontroller for Embedded Systems using Discrete Orthogonal Transform*. Retrieved from Journal of Computer Science 2006. ISSN 1549-3636. Pages 388-394
- K.Y. Khoo and Y.H. Tay (2006), *A Preliminary Study on Embedded Platforms for Computer Vision Applications*, Regional Computer Science Postgraduate Conference 2006, Universiti Teknologi Malaysia
- Komuro T, Tabata T, and Ishikawa M (2010), *A Reconfigurable Embedded System for 1000 f/s Real-Time Vision*, Retrieved from IEEE Transactions on Circuits and Systems for Video Technology (TCSVT.2009)
- Larry Matthies (2005), *Vision Systems for Mars Rovers and Landers*. Retrieved from Jet Propulsion Laboratory. Web site:  
<http://www.scr.siemens.com/ecv05>

- Linda G. Shapiro and George C. Stockman (2001), *Computer Vision*. Retrieved from Prentice Hall. ISBN 0-13-030796-3.
- Linley Gwennap (2005), *A Guide to High-Speed Embedded Processor, 3rd Edition*. Retrieved from Linley Group report.
- Mani N. and Srinivasan B. (October 12 1997), *Application of Artificial Neural Network Model for Optical Character Recognition*. Retrieved from IEEE International Conference on Computational Cybernetics and Simulation. ISBN: 0-7803-4053-1. Issue volume: 3. Pages: 2517 - 2520
- Michael Kanellos (December 1 2003), *Intel Scientists Find Wall for Moore's Law*. Retrieved from CNET Networks Inc. Web site <http://news.cnet.com/2100-1008-5112061.html>
- Microsoft Corp (2008), *WMV HD Content Showcase-High Definition Quality with WMV9*. Retrieved from Microsoft Corp Web Site: <http://www.microsoft.com/windows/windowsmedia/musicandvideo/hdvideo/contentshowcase.aspx#sysreq>
- Musser D. and Stepanov, A. (1994), *Algorithm-Oriented Generic Libraries*. Retrieved from Software-Practice and Experience. Issues Vol. 24. Pages 623-642
- Nicolas Blanc and Zurich (2001), *CCD versus CMOS - has CCD imaging come to an end?*. Retrieved from Photogrammetric Week Issue 01, 2001, Page 131-137.
- Pakdaman M. and Sanaatiyan M. (2009), *Design and Implementation of Line Follower Robot*, Retrieved from Computer and Electrical Engineering, 2009 (ICCEE '09)
- Pratt and William K. (1996), *Overview of the ISO/IEC Image Processing and Interchange Standard*. Retrieved from Proceeding of Standards for Electronic Imaging Technologies, Devices, and Systems, (SPIE). Volume CR61. Pages 29-53

Proshanta Saha (2007), *Automatic Software Hardware Co-Design for Reconfigurable Computing Systems*. Retrieved from 17th International Conference on Field Programmable Logic and Applications (FPL 2007)

Ravi Krishnan (June, 2005), *Future of Embedded Systems Technology*. Retrieved from BCC Reseach Publication ID: WA1124992

Refsnes Data (2008), *Browser Statistics*. Retrieved from Refsnes Data Company Web site:  
[http://www.w3schools.com/browsers/browsers\\_stats.asp](http://www.w3schools.com/browsers/browsers_stats.asp)

Rick Lehrbaum (December 2001), *Embedded Processor and System-on-Chip Quick Reference Guide*. Retrieved from LinuxDevices.com Web site  
<http://www.linuxdevices.com/articles/AT4313418436.html>

Robert Keenan (Apr 07, 2003), *Startup Octasic Delivers Scalable Echo Cancellor*. Retrieved from CommsDesign.com. Web site  
<http://www.commsdesign.com/news/OEG20030407S0070>

Robert L. Kruse and Alex Ryba (October 3, 1998), *Data Structures and Program Design in C++*. Retrieved from Prentice Hall. ISBN-10: 0137689950

Roberto Ierusalimschy (March 2006), *Programming in Lua, 2nd Edition*. Retrieved from Lua.org. ISBN13 9788590379829

Roy, S., Ankcorn, J. and Wee, S. (2003), *Architecture of a Modular Streaming Media Server for Content Delivery Networks*. Retrieved from Multimedia and Expo 2003, ICME apos; 03. Proceedings. 2003 International Conference on Volume 3, Issue 6-9 July 2003, Page: III - 569-72

Shahrin Azuan Nazeer, Nazaruiddin Omar, Khairol Faisal and Jumari,Marzuki Khalid (March 27, 2007), *Face Detecting Using Artificial Neural Network Approach*. Retrieved from First Asia International Conference on Modelling & Simulation (AMS'07). ISBN: 0-7695-2845-7

- Sigma Designs Inc (August 24, 2004), *I-O Data Launches Networked DVD Media Player with WMV9 Support*. Retrieved from Sigma Designs Web site  
[http://www.sigmadesigns.com/news/press\\_releases/040824.htm](http://www.sigmadesigns.com/news/press_releases/040824.htm)
- Tammy Noergaard (February 24, 2005), *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*. Retrieved from Newnes Publisher. ISBN-10: 0750677929
- Thorsten Grotker, Ulrich Holtmann, Holger Keding, and Markus Wloka (November 30, 2007), *The Developer's Guide to Debugging, 1st Edition*. Retrieved from Springer. ASIN: B001VNC98G
- Wayne Wolf (September 25, 2006), *High-Performance Embedded Computing: Architectures, Applications, and Methodologies*. Retrieved from Morgan Kaufmann Publisher. ISBN-13: 978-0123694850
- Wendy Bergeru (April 15, 2002), *Planning and Implementing a Research Study*. British Columbia and Ministry of Forests Research Branch, Web site <http://www.for.gov.bc.ca/hre/forprod/researchprotocols.pdf>
- William Von Hagen (August 11, 2006), *The Definitive Guide to GCC, Second Edition*. Retrieved from Apress Publisher. ISBN-10: 1590595858
- Zamani Zakariah (March 2004) *Transition into Next Generation Networks*. Retrieved from Malaysia Communication and Multimedia Commission 2004 report. Proceeding of APEC Telecommunications and Information Working Group. Web site:  
<http://www.cu.ipv6tf.org/casos/unpan017962.pdf>

## APPENDIX A

### Manuals of ECVLIB Systems

This manual is to guide user to setup ECV base systems. Users can either choose PC or hardware version of ECV base systems. The only different on setup procedure is in section 2. The section 3 is common for either version

## **Section 1. Prepare Development Environment**

### **Install Debian 5.0 from CD**

Link : <http://cdimage.debian.org/cdimage/lenny/i386/iso-cd/debian-testing-i386-netinst.iso>

### **Update systems**

```
HostPC# apt-get update
HostPC# apt-get dist-upgrade
```

### **Install GUI systems**

```
HostPC# apt-get install gnome
```

### **Prepare installation source**

```
HostPC# Vi /etc/apt/source.list
deb http://download.atmark-techno.com/debian\_etch/
```

### **Install cross compile software**

```
HostPC# dpkg -i genext2fs_1.3-7.1-cvs20050225_i386.deb
Link: http://download.atmark-techno.com/armadillo-9/
tools/genext2fs_1.3-7.1-cvs20050225_i386.deb
HostPC# vi /etc/apt/preferences
Package: genext2fs
Pin: version 1.3-7.1*
Pin-Priority: 1001
HostPC# apt-get install atde-essential-arm
HostPC# apt-get install a9-development-environment
HostPC# apt-get install libpcap0.8-arm-cross
HostPC# apt-get install libpcap0.8-dev-arm-cross
HostPC# apt-get install libnet0-arm-cross
HostPC# apt-get install libnet0-dev-arm-cross
```

## **Section 2. Choose Version of ECV Base Systems**

Go to section 2.1, if user chooses to setup ECV base systems on VMware simulator.

If you have *Armadillo-9* or *Armadillo-300* embedded systems board, you might refer section 2.2 for the guide to install ECV base systems on the embedded systems board.

## **Section 2.1. PC Version Using VMware as Emulator**

### **Step 1 - Install VMware player**

Please refer to VMware player to create an empty virtual PC, or  
Download ready VMware blank PC File from:

<ftp://cvislab.com/project/kykhoo/ECVsystem-i386-VMW-001-Blank.7z>

### **Step 2 - Obtain latest CD image from Debian website**

Example link:

[http://cdimage.debian.org/debian-cd/4.0\\_r3/i386/iso-cd/debian-40r3-i386-netinst.iso](http://cdimage.debian.org/debian-cd/4.0_r3/i386/iso-cd/debian-40r3-i386-netinst.iso)

### **Step 3 - Install the Debian mini systems**

Booting from the CD to the systems and follow the instruction menu  
During installation it asks about networkmirror, please DON'T use any  
network mirror.

### **Step 4 - Tuning ECV base systems**

1. Turn off annoying beep sound

```
HostPC# vi /etc/rc.local
        modprobe -r pcspkr.
```

2. Fast boot up time by adjust grub

```
HostPC# vi /boot/grub/menu.lst
        timeout = 1
```

## **Section 2.2. Hardware Version Using Embedded System Board**

### **Step 1 - Prepare typical on board systems. Currently for Armadillo-9 /300 board only**

Burning Bootloader, Kernel, Userland flash image using downloader program  
(Note: Jumper 1= SET, Jumper 2 = CLEAR)

- Downloader = Hermit\_WIN32 program
- Bootloader = "loader-armadillo9-eth-v1.1.13.bin" image
- Kernel = "linux-2.6.12.3-a9-13.bin.gz" image
- Userland = "romfs-20071214-2.6.12.3-a9-13.img.gz" image

### **Step 2 - Prepare network and mount CF Card**

(Note: Jumper 1= CLEAR, Jumper 2 = SET)

```
Board.rs232 # clearenv
Board.rs232 # boot
Board.rs232 # fdisk /dev/hdc // create only 1 partition, type
83
Board.rs232 # mke2fs -O -filetype /dev/hdc1
Board.rs232 # mount /dev/hdc1 /mnt
Board.rs232 # mount -t ramfs none /home/ftp/pub
Board.rs232 # chmod 777 /home/ftp/pub
Board.rs232 # ifconfig eth0 192.168.88.7 netmask 255.255.255.0
up
```



### Step 3 - Install by unzip Debian into ARM-CF card

Transfer each Debian zip file using FTP protocol, and then unzip it into CF card. Repeat following step for 5 of the Debian zip file

```
HostPC# ftp 192.168.88.7
HostPC# ftp> cd pub
HostPC# ftp> bin
HostPC# ftp> put debian-etch-a9-X.tgz

Board.rs232 # tar zxvf /home/ftp/pub/ debian-etch-a9-X.tgz -C
/mnt
Board.rs232 # rm -f /home/ftp/pub/debian-etch-a9-X.tgz
Board.rs232 # sync && umount /mnt
```

### Step 4 - 1<sup>st</sup> time boot Debian from CF card

(Note: JP1 = CLEAR, JP2 = SET)

```
Hermit v1.3-armadillo9-2 compiled at 22:13:37, Mar 11 2005
Board.rs232 # setenv noinitrd root=/dev/hdcl
```

(ECVsystem-A300-CFcard-001-Basic)

### Step 5 - Boot Debian from ARM-CF card (JP1 = Clear, JP2 = Clear)

```
Board # Login : root (no password)
Board # passwd
Board # useradd -D -s /bin/bash
Board # useradd -c user -m user
Board # passwd user
```

### Step 6 - Enable Debian networking

```
Board # apt-get install wireless-tools
Board # vi /etc/network/interfaces
# Used by ifup(8) and ifdown(8). See the interfaces(5)
manpage or
# /usr/share/doc/ifupdown/examples for more information.

auto lo ath0 eth0
iface lo inet loopback
iface ath0 inet static
    address 192.168.88.7
    gateway 192.168.88.1
    netmask 255.255.255.0
    network 192.168.88.0
    broadcast 192.168.88.255
    wireless-essid CVIS-Wifi_2_(Test)
    wireless-key qwertyuiop
iface eth0 inet static
    address 192.168.88.11
    gateway 192.168.88.1
    netmask 255.255.255.0
    network 192.168.88.0
    broadcast 192.168.88.255
Board # vi /etc/resolv.conf
search local-network
nameserver 202.188.0.133
nameserver 202.188.1.5
Board # vi /etc/hostname
ECVsystem
Board # /etc/init.d/networking restart
```

### Step 6 - Fix APT and update system time

```
Board # mkdir /var/cache
Board # mkdir /var/cache/debconf
Board # mkdir /var/cache/apt
Board # mkdir /var/cache/apt/archives
Board # mkdir /var/cache/apt/archives/partial
Board # apt-get update
Board # apt-get install ntp-server ntp-simple
Board # vi /etc/ntp.conf
Board # /etc/init.d/ntp restart
Board # date -s "12/2/2008 21:18:00"
```

(ECVsystem-A300-CFcard-002-MiniSystem)

## Section 3. Prepare Debian as ECV Base Systems

### Step 1: Update Debian systems

update all package to very latest package. Purpose: Point to the latest (APT)'s software list

```
HostPC# vi /etc/apt/sources.list
deb http://http.us.debian.org/debian lenny main
HostPC# apt-get update
HostPC# apt-get -u dist-upgrade
```

Repeat step 1 until terminal show

0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded

(ECVsystem-A300-CFcard-003-Update-20080808)

### Step 2: Install necessary package

For any package, please use bellow command to find the actual name

```
HostPC# apt-cache search [name_of_package]
```

Then install the package by

```
HostPC# apt-get install [name_of_package]
```

wireless-tools

#### For Development Tools Package

Description	Name of Package	Note
Communication	openssh-server	Secure shell server, an rshd replacement
Compiler	build-essential	Informational list of build-essential packages
	locate	Maintain and query an index of a directory tree
	libreadline5-dev	GNU readline and history libraries, development files
	pkg-config	Manage compile and link flags for libraries
	libssl-dev	SSL development libraries, and header files

Share	samba	<p>A LAN manager - file/printer server for Unix</p> <pre> Board # smbpass -a user; Board # vi /etc/samba/smb.conf         [homes]             comment = Home Directories         browseable = yes         writable = yes         create mask = 0775         directory mask = 0775 </pre>
-------	-------	--

(ECVsystem-A300-CFcard-004-DevTools)

### For Internet Server Package

Description	Name of Package	Note
Web	xavante	<p>Lua HTTP 1.1 Web server</p> <pre> vi /etc/xavante/xavante_start.lua     server = {host = "*", port = 80}, vi /etc/rc.local     lua /etc/xavante/xavante_start.lua &amp;  mkdir /home/user/www/ &amp;&amp; chmod 777 /home/user/www/ vi /etc/xavante/sites-available/localhost.lua     local webDir = '/home/user/www/'     params = {"index.lp"} </pre>
	Kepler 1.1 Beta 2	<p>Lua HTTP 1.1 Web server - kepler module</p> <pre> Board # wget <a href="http://luaforge.net/frs/download.php/2720/kepler-1.1beta2.tar.gz">http://luaforge.net/frs/download.php/2720/kepler-1.1beta2.tar.gz</a> Board # tar -xzvf kepler-1.1beta2.tar.gz Board # cd kepler-1.1 Board # ./configure --launcher=xavante         --enable-lua --prefix=/usr Board # make &amp;&amp; make install </pre>
Scripting	lua5.1	Simple, extensible, embeddable programming language
	liblua5.1-0	LUA shared library
	liblua5.1-sql-sqlite3-2	luasql library for the lua language version 5.1
Database	sqlite3	A command line interface for SQLite 3
	libsqlite3-0	SQLite 3 shared library
Video	ffmpeg	multimedia player, server and encoder

### Configure internet server

```
Board # vi /usr/etc/kepler/1.1/xavante/config.lua >>> port=80
Board # ln -s /usr/htdocs /home/user/ecv
Board # vi /etc/rc.local
        xavante_start &
Board # chmod 777 -R /usr/htdocs
```

### For Hardware Device Driver Package

Description	Name of Package	Note
WebCam	gspca-source	GSPCA video for Linux (v4l) driver modules
	qc-usb-source	source code for QuickCam Express kernel module
	qc-usb-utils	Utility programs for the qc-usb kernel module
Finger Print	libdpfp	Board # wget http://download.berlios.de/dpfp/ libdpfp-0.2.2.tar.gz Board # tar -xzvf libdpfp- 0.2.2.tar.gz Board # cd libdpfp-0.2.2 Board # ./configure Board # make && make check && make install
USB	libusb++-0.1-4c2	userspace C++ USB programming library
	libusb++-dev	userspace C++ USB programming library development files
	libusb-0.1-4	userspace USB programming library
	libusb-dev	userspace USB programming library development files
	usbutils	Linux USB utilities

Configure GSPCA driver

NWFPE math emulation

FastFPE math emulation (EXPERIMENTAL)

### Compile Kernel

```
HostPC# #tar xzvf linux-2.6.12.3-a9-8.tar.gz
HostPC# #cd linux-2.6.12.3-a9-8
HostPC# #make armadillo9_defconfig
HostPC# #make menuconfig
        Device Drivers -> Multimedia devices -> <M>Video For
Linux
        Device Drivers -> Graphics support -> <>Support for
frame ...
        Device Drivers -> USB Support ...
HostPC# #make
HostPC# #make modules
HostPC# #make modules_install
```

arch/arm/boot/Image

arch/arm/boot/zImage

### Compile GSPCA driver

```
cd /usr/src
tar -xjvf gspca-source.tar.bz2
```

```

cd /usr/src/modules/gspca/
HostPC# #vi Makefile
        KERNEL_VERSION=2.6.12.5-at5
make CC=arm-linux-gnu-gcc LD=arm-linux-gnu-ld AR=arm-linux-gnu-
ar NM=arm-linux-gnu-nm OBJCOPY=arm-linux-gnu-objcopy
make install CC=arm-linux-gnu-gcc LD=arm-linux-gnu-ld AR=arm-
linux-gnu-ar NM=arm-linux-gnu-nm OBJCOPY=arm-linux-gnu-objcopy

```

### **Compile QC-USB driver**

```

cd /usr/src
tar -xzvf qc-usb-modules.tar.gz
cd /usr/src/modules/qc-usb-source
HostPC# #vi Makefile
        MODULE_DIR := /lib/modules/2.6.12.5-at5
        KVER      := 2.6.12.3-a9-16
        KMISC     := /lib/modules/$(KVER)/kernel/drivers/media/video/
make all CC=arm-linux-gnu-gcc LD=arm-linux-gnu-ld AR=arm-linux-
gnu-ar NM=arm-linux-gnu-nm OBJCOPY=arm-linux-gnu-objcopy
make install CC=arm-linux-gnu-gcc LD=arm-linux-gnu-ld AR=arm-
linux-gnu-ar NM=arm-linux-gnu-nm OBJCOPY=arm-linux-gnu-objcopy

```

### **Copy Kernel Module**

```

cd /lib/modules/
tar -cjvf 2.6.12.5-at5.bz2 2.6.12.5-at5
mv 2.6.12.5-at5.bz2 /home/user/Desktop/

```

### **Auto Load driver module**

```

HostPC# #tar zxvf gspcav1-20070508.tar.gz
HostPC# #cd ../gspcav1-20070508

HostPC# #vi Makefile
        KERNEL_VERSION=2.6.12.5-at5

HostPC# #cd
HostPC# #make install

HostPC# cd /lib/modules
HostPC# tar czvf 2.6.12.3-a9-8.tar.gz 2.6.12.3-a9-8
Board # cd /lib/modules
Board # tar zxvf 2.6.12.3-a9-8.tar.gz

qc-usb-modules.tar.gz

m-a prepare
cd /usr/src
tar -xjvf qc-usb.tar.bz2
m-a i-a qc-usb

```

### **Prepare Kernel for Hardware Driver Compilation**

```

apt-get install libreadline5-dev
cd /usr/src/ && tar -xzvf linux-2.6..tz
cd linux-2.6
make armadillo300_defconfig
make menuconfig
        Device Drivers -> Multimedia devices -> <M>Video For Linux
        Device Drivers -> Graphics support -> <>Support for frame ...
make CC=gcc LD=ld AR=ar NM=nm OBJCOPY=objcopy
make install

```

```
make modules
make modules_install
```

**Re-compile kernel as optimize using cross-compiler from Atmark Corp**

```
make armadillo300_defconfig
make menuconfig
    Device Drivers -> Multimedia devices -> <M>Video For Linux
    Device Drivers -> Graphics support -> <>Support for frame ...
make
make install
make modules
make modules_install
```

**Compile hardware driver**

```
ln -s cpp-4.3 arm-linux-cpp
ln -s g++-4.3 arm-linux-g++
ln -s gcc-4.3 arm-linux-gcc
Cd /usr/src/
m-a a-i qc-usb
make install CC=gcc LD=ld AR=ar NM=nm OBJCOPY=objcopy

make CC=arm-linux-gnu -gcc LD= arm-linux-gnu -ld AR= arm-
linux-gnu -ar NM= arm-linux-gnu -nm OBJCOPY= arm-linux-gnu-
objcopy
```

**Prepare hardware node**

```
mknod /dev/video0 c 81 0
```

**Multimedia Manipulate Tools**

Description	Name of Package	Note
Image	netpbm	Graphics conversion tools
	libjpeg-progs	Programs for manipulating JPEG files
	libjpeg62	The independent JPEG group's JPEG runtime library
Video	libsdl1.2-dev	Simple DirectMedia layer development files
	mjpegtools	Board # wget http://optusnet.dl.sourceforge.net/ sourceforge/mjpeg/ mjpegtools-1.9.0rc3.tar.gz Board # tar -xzvf mjpegtools- 1.9.0rc3.tar.gz Board # cd mjpegtools-1.9.0rc3 Board # ./configure Board # make && make check && make install

**Section 4. Install UTAR-ECV Library**

**Only Steps: Install UTAR-ECV library**

To install source code, demo and application template of UTAR-ECV library  
Please request access key from author, and download it from:

<http://cvislab.com/project/kykhoo/utarecv.zip.php>

## APPENDIX B

### How to Add Your Own Function in UTAR-ECV Library

A new function has to be added in the ECVfunc package. To do that, we create a new class. This class inherits from ECVFunction abstract class. As an abstract class, it has to implement some methods. Once created, the file containing the class is inserted in the ECVfunc directory. That's all. Once compiled, it can be used from the GUI and command line. The following code is an example of a function (ECVbinarize):

```
// This option indicates that this function owns to the ecvfunc package
package ecvfunc;
import ecv.ECV;
import ecv.ECVFunction;
import ecv.ECVImage;
import ecv.ECVParameter;
import ecv.ECVFunctionList;

//A function allways inherits from ECVFunction
public class FBinarize extends ECVFunction {
    public FBinarize() { // Constructor
        name = "FBinarize"; // Function name
        //Description
        description = "Transforms a BYTE image to binary";
        //Group to which this function is included
        groupFunc = ECVFunctionList.TRANSFORM;
        //First parameter
        ECVParameter p1 = new ECVParameter("u1", ECV.pINT, false);
        p1.setDefault(128);
        p1.setDescription("Lower bound of the range to consider as 1");
        //Second parameter
        ECVParameter p2 = new ECVParameter("u2", ECV.pINT, false);
        p2.setDefault(255);
        p2.setDescription("Upper bound of the range to consider as 1");

        params.addElement(p1);
        params.addElement(p2);
    }

    // A function allways implements this method. It is where our code is
    placed
    public ECVImage processImg(ECVImage img) {
        ECVImage res = null;
        //We obtain the value of the parameters
        int p1 = getParamValueInt("u1");
        int p2 = getParamValueInt("u2");
        // This function is only applied to BYTE images
        if (img.getType() == ECV.tBYTE) {
            int w = img.getWidth();
            int h = img.getHeight();
            int b = img.getNumBands();
            //The result image is created
            res = new ECVImage(b, w, h, ECV.tBIT);
            // For each band
            for (int nb = 0; nb < b; nb++) {
                // We obtain all the pixels, as we don't need spatial
relationships
                int[] bmp = img.getAllPixel(nb);
                int[] bin = new int[w * h];
                for (int i = 0; i < w * h; i++)
                    bin[i] = (bmp[i] >= p1 && bmp[i] <= p2) ? 1 : 0;
                // Once the band is processed, it is assigned to the output
image
                res.setAllPixel(nb, bin);
            }
        }
        else {
            error=new String("Binarize only defined for BYTE images");
            return img;
        }
        return res;
    }
}
```