IMPLEMENTATION OF A LOW-COST SMART CAMERA APPLICATION ON A COTS SYSTEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HAYRİ KEREM BAYKENT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2011

Approval of the thesis:

IMPLEMENTATION OF A LOW-COST SMART CAMERA APPLICATION ON A COTS SYSTEM

submitted by **HAYRİ KEREM BAYKENT** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Gözde Bozdağı Akar
Supervisor, **Electrical and Electronics Engineering Dept, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Ece Güran Schmidt
Electrical and Electronics Engineering Dept., METU _____

Dr. Fatih Kamışlı
Electrical and Electronics Engineering Dept., METU _____

Yusuf Bediz, Msc
Electrical and Electronics Engineering Dept., METU _____

Date: 07/12/2011

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : HAYRİ KEREM BAYKENT

Signature :

ABSTRACT

IMPLEMENTATION OF A LOW-COST SMART CAMERA APLLICATION ON A COTS SYSTEM

Baykent, Hayri Kerem

M.Sc., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Gözde Bozdağı Akar

December 2011, 153 pages

The objective of this study is to implement a low-cost smart camera application on a Commercial off the Shelf system that is based on Texas Instrument's DM3730 System on Chip processor. Although there are different architectures for smart camera applications, ARM plus DSP based System on Chip architecture is selected for implementation because of its different core abilities. Beagleboard-XM platform that has an ARM plus DSP based System on Chip processor is chosen as Commercial off the Shelf platform. During this thesis, firstly to start-up the Commercial off the Shelf platform the design steps of porting an embedded Linux to ARM core of System on Chip processor is described. Then design steps that are necessary for implementation of smart camera applications on both ARM and DSP cores in parallel are given in detail. Furthermore, the real-time image processing performance of the Beagleboard-xM platform for the smart camera applications is evaluated with simple implementations.

Key words: Low cost smart camera design, real time image processing, ARM and DSP architectures, embedded system design, performance evaluation of Beagleboard-xM

ÖZ

DÜŞÜK MALİYETLİ BİR AKILLI KAMERA UYGULAMASININ TİCARİ KULLANIMA HAZIR BİR SİSTEM ÜZERİNDE GERÇEKLENMESİ

Baykent, Hayri Kerem

Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Gözde Bozdağı Akar

Aralık 2011, 153 sayfa

Bu çalışmanın amacı, Texas Instrument'ın DM3730 elektronik yonga üzerinde sistem işlemcisine dayalı ticari kullanıma hazır bir sistem üzerinde düşük maliyetli bir akıllı kamera uygulamasını gerçekleştirmektir. Akıllı kamera uygulamaları için çeşitli mimariler olmasına rağmen, uygulama için farklı çekirdek yetenekleri sebebiyle ARM artı DSP tabanlı elektronik yonga üzerinde sistem mimarisi seçilmiştir. ARM artı DSP tabanlı elektronik yonga üzerinde sistem işlemcisine sahip Beagleboard-XM platformu ticari kullanıma hazır platform olarak seçilmiştir. Bu tez boyunca, ilk olarak ticari kullanıma hazır platformu ayağa kaldırmak için elektronik yonga üzerinde sistem işlemcinin ARM çekirdeğine gömülü Linux işletim sistemi uyarlamasının tasarım basamakları tanımlanmıştır. Daha sonra akıllı kamera uygulamalarının ARM ve DSP çekirdekleri üzerinde paralel olarak gerçeklenmesi için gerekli tasarım basamakları detaylı olarak verilmiştir. Ayrıca, akıllı kamera uygulamaları için Beagleboard-xM platformunun gerçek zamanlı görüntü işleme performansı basit uygulamalarla değerlendirilmiştir.

Anahtar kelimeler: Düşük maliyetli akıllı kamera tasarımı, gerçek zamanlı görüntü işleme, ARM ve DSP mimarileri, gömülü sistem tasarımı, Beagleboard-xM performans değerlendirmesi

To my family

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor Prof. Dr. Gözde Bozdağı Akar for his guidance, support and valuable suggestions in preparing this thesis.

I am also thankful to my committee members Assoc. Prof. Dr. Cüneyt Bazlamaçcı, Assoc. Prof. Dr. Ece Güran Schmidt, Dr. Fatih Kamışlı and Yusuf Bediz, Msc for their time and for their efforts in helping to improve the quality of this manuscript.

I acknowledge the financial support of TUBITAK (BIDEB-2210 Fellowship) during my master studies.

I feel great appreciation to Alper Ünsoy, Mehmet Aydın, Mehmet Umut Demirçin, Süleyman Alpay Aslangül, Hüseyin Ogün Şen, Ali Erdem Özcan, Semih Çakıl, Mustafa Morca, Mehmet Fatih Karagöz, Kubilay Pakin, Murat Gevrekci and Hamza Ergezer for their helpful advices, support and encouragement.

I am deeply thankful to my colleagues for their friendship, moral support and suggestions throughout this study.

I would like to express my endless gratitude to my parents for their love, endless support and trust throughout my life.

Last but not least, I would like to thank to my wife Handan Baykent for her precious support, care, love and patience.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

ABBREVIATIONS

ALU : Arithmetic Logic Unit
AMD : Advanced Micro Devices
ANSI : American National Standards Institute
API : Application Programming Interface
ARGB : Alpha Red Green Blue
ARM : Advanced Risc Machine
ASIC : Application Specific Integrated Circuit
BIOS : Basic Input/Output System
CD : Compact Disc
CDT : C/C++ Development Tool
CE : Codec Engine
CIF : Common Intermediate Format
CISC : Complex Instruction Set Computer
CMEM : Contiguous Memory Management
CMOS : Complementary Metal Oxide Semiconductor
CODEC : Coder-Decoder
COTS : Commercial off the Shelf
CPLD : Complex Programmable Logic Device
CPU : Central Processing Unit
DDR : Double Data Rate
DISPC : Display Controller
DMA : Direct Memory Access
DMAI : Davinci Multimedia Application Interface
DMIPS : Dhrystone Million Instructions per Second
DRAM : Dynamic Random Access Memory
DSP : Digital Signal Processor
DSPLIB : Digital Signal Processor Library
DVD : Digital Versatile Disc
DVI : Digital Video Interface
DVSDK : Digital Video Software Development Kit
ECC : Error-Correction Code
EEPROM : Electrically Erasable Programmable Read-Only Memory
EDMA : Enhanced Data Memory Access
EPROM : Erasable Programmable Read-Only Memory
ETB : Embedded Trace Buffer
ETM : Embedded Trace Macrocell
EVM : Evaluation Module
FAT : File Allocation Table
FIFO : First In First Out
FFT : Fast Fourier Transform
FPGA : Field Programmable Gate Array

FPS : Frame per Second
FPU : Floating Point Unit
GCC : GNU Compiler Collection
GMM : Gaussian Mixture Model
GNOME : GNU Network Object Model Environment
GPS : Global Positioning System
GPRS : General Packet Radio Service
GPU : Graphics Processing Unit
HD : High Definition
HDL : Hardware Description Language
HDTV : High Definition Television
IC : Integrated Circuit
ICE : In-Circuit Emulator
IDE : Integrated Development Environment
IMGLIB : Image Library
INTC : Interrupt Controller
IP : Internet Protocol
IPC : Interprocessor Communication
ISP : Image Signal Processor
IOCTL : Input Output Control Logic
IVA : Image Video Audio
IEEE : Institute of Electrical and Electronics Engineers, Inc.
JTAG : Joint Test Action Group
LCD : Liquid Crystal Display
LPM : Local Power Manager
MAC : Multiply-Accumulate
MBX : Mailboxes
MDDR : Mobile Double Data Rate
MIPS : Microprocessor without Interlocked Pipeline Stages
MMC : Multimedia Card
MMU : Memory Management Unit
MoG : Mixture of Gaussians
MPEG : Moving Picture Experts Group
MPU : Microprocessor Unit
MSGQ : Message Queues
NFS : Network File Server
NTSC : National Television System Committee
NVDK : Network Video Development Kit
OMAP : Open Multimedia Application Platform
OMAPDSS2 : Open Multimedia Application Platform Display-Subsystems Two
OMAPFBdev : Open Multimedia Application Platform Frame Buffer Device
OS : Operating System
OSD : On Screen Display
PAL : Phase Alternating Line
PC : Personal Computer
PCI : Peripheral Component Interconnect
PDA : Personal Digital Assistant
PMIC : Power Management Multi-Channel Integrated Circuit

PRCM : Power, Reset, and Clock Manager
POP : Package on Package
RAM : Random Access Memory
ROM : Read-Only Memory
RF : Radio Frequency
RFBI : Remote Frame Buffer Interface
RGA : Running Gaussian Average
RGB : Red Green Blue
RGBA : Red Green Blue Alpha
RISC : Reduced Instruction Set Computer
RTOS : Real Time Operating System
SD : Standard Definition
SDMA : System Direct Memory Access
SDK : Software Development Kit
SDRAM : Synchronous Dynamic Random Access Memory
SIMD : Single Instruction Multiple Data
SoC : System on Chip
SPEC : Standard Performance Evaluation Corporation
SRAM : Static Random Access Memory
SYSC : System Controller
TI : Texas Instruments
TLB : Translation Look-aside Buffers
TV : Television
UNIX : Uniplexed Information and Computing System
USB : Universal Serial Bus
V4L2 : Video for Linux Two
VGA : Video Graphics Array
VFP : Vector Floating Point
VLIW : Very Long Instruction Word
VLSI : Very Large Scale Integration
VM : Virtual Machine
WLAN : Wireless Local Area Network
WUGEN : Wake-Up Generator

CHAPTER 1

INTRODUCTION

1.1. Background

During recent years, visual surveillance has become an important topic of research due to the increased need for both military and public safety sectors. The need for surveillance in large areas such as shopping centers and borders increase the number of surveillance cameras connected to the main video analyzer performing dedicated real time image processing algorithms to extract the valuable information. These increase the processing and storage load of main video analyzer and result with development of smart cameras. The aim of smart camera design is extracting the valuable information at camera site and sending only this information to decrease the bandwidth and cost of communication channels. However smart cameras power consumption, volume and cost should not be high like main video analyzers. Therefore these smart cameras must be developed on low cost, low power and low volume embedded platforms and real time visual surveillance applications developed and investigated for high processing power computers and platforms must be ported to these embedded platforms. Real time implementation of visual surveillance applications can be applied on different platforms and it is not tied to a specific hardware platform. However hardware platform must be capable of video capturing, video streaming and image processing operations in the limitations of computational performance, cost and power constraints.

Advances in embedded processors, bring news platforms with the low size, low weight, low power and low cost. This results in a great opportunity in developing real-time systems. Also rich peripheral environments and different architecture capabilities of System on Chip (SoC) platforms make easy the design of smart and complex systems such as smart cameras.

1.2. Scope of Thesis

This thesis work defines how to implement a smart camera application on a low-cost COTS system that is based on TI ARM + DSP SoC, DM3730 processor. We also present a summary of the advantages of ARM and DSP based SoC architectures together with a review on different architectures. In this thesis Beagleboard-XM platform is chosen as a COTS platform and its SoC architecture is investigated. The design steps of porting an embedded Linux to ARM core of SoC processor to start-up the COTS platform is given in detail. The algorithm implementation steps on ARM + DSP cores of the Beagleboard-XM platform are described and the real-time image processing performances of the ARM and DSP cores for smart camera applications are evaluated.

Beagleboard-XM platform is a commercial off the shelf platform without official software support to design a specific application on it. Therefore to design a specific application on the platform, the embedded system design procedures must be investigated and implemented on that platform. In this work, ARM core is used as operating system (OS) core to convert an analog detector to a smart camera with the advantages of an OS. OS provides integration of different sensors to the system by using their open source drivers. OS includes the network and external memory management operations that are also necessary for implementation of smart camera application. DSP core is used as image processing core and it provides enough processing power for necessary image processing applications. In this thesis work Angstrom OS is implemented on ARM core and necessary drivers to implement a simple smart camera application are added to implement Angstrom OS. Communication structures and common RAM memory usage infrastructure of cores are investigated for parallel algorithm implementation. To make necessary input

output connections between algorithms and video capture and video display drivers, platform specific video capture and video display driver infrastructures are investigated. Design steps of the implementation an algorithm on ARM and DSP cores are investigated and described. A simple compile and run procedure is defined to evaluate the performance of algorithms on the target platform. Furthermore, a Running Gaussian average model background subtraction algorithm that needs high processing speed for real time operation was implemented on ARM processor, NEON coprocessor and DSP processor of DM3730 and processor performances of the platform were evaluated separately through experiments.

1.3. Organization of Thesis

Chapter 2 of this thesis gives a brief explanation of smart camera systems. First part of this chapter gives information about example platforms of smart cameras. Second part of this chapter describes the implementation architectures for smart cameras. The chapter ends with a summary of performance comparisons of smart camera implementation platforms.

Chapter 3 introduces the hardware of chosen system for a low cost smart camera design. An overview about the chosen target board namely Beagleboard-xM is given and ARM and DSP core architectures of processor used in the selected hardware are described. Furthermore, important peripherals of selected hardware for the smart camera design are introduced.

Chapter 4 introduces the embedded systems design steps of smart camera application on Beagleboard-XM platform. The embedded OS implementation procedures on the ARM and DSP core are described. Configuration of HOST and TARGET setup described. Necessary driver installation to make necessary input output connections between algorithms and video capture and video display drivers are introduced Interprocessor communication protocol is introduced used for parallel programming on ARM and DSP core. Algorithm development environments for the target board are introduced. The procedures to create a software development kit for Beagleboard-XM board by using DVSDK 4.01 are described in this chapter. This

chapter also defines how to add a DSP algorithm to a codec server and how you can implement an algorithm by using both ARM and DSP cores.

Chapter 5 describes implemented algorithms on the cores and reports the analysis of the data collected. This chapter compares the performance of the Beagleboard-XM platform with 2.8 GHz Host PC and shows the contribution of DSP core and NEON co-processor on the performance of the system. Finally, chapter 6 concludes the thesis.

CHAPTER 2

SMART CAMERAS

Smart Cameras are devices consisting of image sensing device, an image processing module and a communication module. Image sensing device can be a single or multiple image sensors. Image processing module provides the application-specific information processing and communication module connects the smart camera to the external world (host or network) [19]. Smart cameras do not only capture the frames from the scene. They also make a sense of what is happening in the image and in some cases these cameras take the action defined by camera user. Smart cameras offer low power consumption and low physical size, compared to usual PCs. In addition to this, they provide the transmission over low-cost, low-bandwidth communication channels by allowing the extraction of valuable information at site. In the following sections we give a short introduction about the current state of smart camera platforms and implementation architectures.

2.1. Smart Camera Platforms

Recently the new technology Microcontrollers and SoC platforms reduce their power consumption and size while increasing their processing power. A number of Microcontroller and SoC based vision systems have been developed especially dedicated to the task of image processing. The iOne from Ambarella is a smart camera based on Android embedded system [47]. This camera has a SoC with triple ARM cores and an advanced media DSP. The availability of the SoC processor provides the advanced HD camera and multimedia capabilities to that platform. Another commercially available vision platform is a BOA Vision System by

Teledynedalsa [48]. BOA combines a GPU, a DSP and a FPGA core on a one platform and uses the specific advantages of these architectures to design this advanced smart camera. An additional indication for the suitability of SoC concepts for smart camera design is the variety of developments of smart camera platforms reported in the literature. Table 2.1 shows the leading companies and research organizations on smart cameras.

Table 2.1. Some Leading Companies and Research Organizations on Smart Cameras (Adapted from [19])

Organization/company	Project/Product	Country
SCS - Carnegie Mellon University	CMUCam Vision Sensors	USA
WSNL - Stanford University	Mesh Eye architecture	USA
LASMEA - Blaise Pascal University/CNRS	SeeMOS project	France
Vision Components	VC series	Germany
Smart Systems - Ausrian Research Centers	Smart eye sensors	Austria
Le2i - Bourgogne University/CNRS	High-speed smart camera	France
Intellio	ILC series	Hungary
Sony	XCI series	Japan
National Instruments	NI 17 xx series and CVS	USA
SICK IVP	IVC 2D and IVC 3D	Sweden
Philips/NXP Research	WiCa wireless mote	Netherlands
NeuriCam	VISoc Vision System-on-Chip	Italy
ITI - Graz University of Technology	SmartCam project	Austria

In section 2.3, we will give the details of some of the smart camera implementations. However, before this we will first concentrate on the architectures for smart cameras.

2.2. Architectures for Smart Camera Implementation

As already mentioned before, smart cameras are based on embedded system technology. The group of embedded systems is diverse and varies with respect to its level of flexibility between each category. Field Programmable Gate Arrays (FPGAs), Reduced Instruction Set Computers (RISCs), Digital Signal Processors (DSPs), and Application Specific Integrated Circuits (ASICs) are the main categories. On the other hand, two other groups having other special properties,

namely Microcontrollers and Graphics Processing Units (GPUs), share the same features as the embedded systems. Microcontrollers include the set of System-on-Chip platforms (SoCs). Graphics Processing Units (GPUs) is a recently emerging area. A short characterization of each single domain is given hereafter, some notes on general selection criteria are explained in the following section. For a more elaborate introduction to processors and embedded computing, the book of Wolf [15] is a reference for the interested reader. Note that it is not possible to mention all types of methodologies, design issues, development groups and reference implementations here, thus we aim at giving a general overview and only focus on aspects relevant for our own work.

2.2.1 Field Programmable Gate Array (FPGA)

Field Programmable Gate Arrays are semiconductor devices which contain a huge number of logic elements and connections in-between. The elements are called logic blocks and can be programmed to perform a logic operation with limited complexity, from simple AND gates up to full multipliers or even complexer functions. The name field programmable denotes the possibility to program interconnections and logic blocks after manufacturing in the field. Many different types of FPGAs exist, which can be chosen according to design and security issues. FPGAs based on Static Random Access Memory (SRAM) are programmed at power-on and are booted from some external functionality. Erasable Programmable Read-Only Memory (EPROM) devices can be programmed multiple times but is usually programmed during manufacturing. The program can be erased by exposing the device to ultra-violet light, and can be re-programmed afterwards. Electrically Erasable Programmable Read-Only Memory (EEPROM) technology based FPGAs can be erased and re-programmed electrically which makes the need for ultra-violet light exposure obsolete.

Fuse and Anti-Fuse based FPGAs are working on opposite electrical principles. In the first case programming is done by breaking conductive connections and in the latter case connections are established if the applied current is exceeding a specified limit. The latter technology is much more common in the world of Integrated

Circuits (ICs); however, FPGAs based on both techniques can be programmed only once.

The programs for FPGAs are usually written in a Hardware Description Language (HDL) such as ABEL, VHDL or Verilog. Given the description of the desired functionality a number of steps have to be taken to finally place the functionality on the device. To accelerate this process different development tools can be used such as SystemC. An important feature is the libraries of function blocks and macros that can be used to further accelerate the design of programs. The list of manufacturers of FPGAs contains companies like Altera, Atmel, Actel and Xilinx among others. FPGAs can be used to create complete devices such as DSPs or to perform computationally expensive tasks in hardware, such as Fast-Fourier-Transform (FFT) or video en- and decoding. However, FPGAs are also mostly used for prototyping ASICs, and as their capabilities and speed increase, nowadays complete SoC outlines can be fabricated on FPGA technology.

2.2.2. Reduced Instruction Set Computer (RISC)

The term RISC was introduced to differentiate from the Complex Instruction Set Computer (CISC) architectures. The RISC was introduced in the late 1970s based on several design principles to create a new architecture, originally to facilitate the use of optimized compilers for the generation of machine code. The instruction set should contain only simple instructions, being decodable by the CPU within one clock cycle. Furthermore, the use of register files and pipelining allows for execution at high frequencies, also factoring out slow memory accesses. For an early introduction and review of RISC processors, the reader is referred to the article of Patterson [16].

Main representatives of RISC processors nowadays are the ARM processor family [17], the MIPS architecture [18] and the PowerPCs. Manufacturers of RISC processors mainly include Freescale Semiconductors, IBM, AMCC and MIPS Technologies.

2.2.3 Digital Signal Processor (DSP)

Another big and important group of embedded processors is formed by DSPs. Originally developed for one-dimensional signal processing tasks in the real-time computing domain for telecommunications; they are now more and more emerging into the image processing domain. To allow for fast itering and folding operations, one important feature of DSPs was an onboard multiplier, also providing a multiply-accumulate (MAC) instruction. This is still a common feature on nowadays DSP architectures. Recent DSPs also often include specialized instructions for more evolved digital signal processing operations, such as Viterbi en-/decoding for example. Being related closely to general-purpose computing systems in respect of their programming flexibility, the newer series DSPs are featuring Very Long Instruction Word (VLIW) and Single-Instruction, Multiple-Data (SIMD) technology, which means that multiple functional units can be handled concurrently. Software for DSPs is usually written in a high-level language like C or C++ which makes development of applications relatively straight-forward and efficient. Optimization of programs is done during compilation which means that the developer is only able to in influence optimization at a moderate level. While DSPs have been developed working mainly in the fixed point domain for almost two decades, recently DSPs equipped with a Floating Point Unit (FPU) have become more attractive. However, fixed-point calculation is still the predominant domain, as more evolved DSP architectures clearly come at a considerably higher price due to the increased hardware complexity. Needless to say, that the inclusion of FPUs in DSPs also comes at considerably higher energy consumption and increased chip area.

First prototypes of DSPs were proposed in the late 70s by Intel and AMI. Later the principles were refined and the result full-features DSPs were presented by AT&T and NEC in 1980. A big success was the introduction of the first DSPs from Texas Instruments which is still holding on. Today TI is the biggest manufacturer of DSPs beside other producers like Motorola and Analog Devices. The range of pricing for a DSP ranges from a few up to a few hundred dollars. Likewise the band of power consumption of DSPs ranges from 50mW up to 5W. The variety of DSPs is manifold and special type processors are available for almost each specific application, thus it is up to the developer make the right selection. Concerning a very popular group of

DSPs especially suitable for video processing, the TI TMS320C64x™ DSP series a block diagram of the basic processor design is depicted in Figure 2.1.

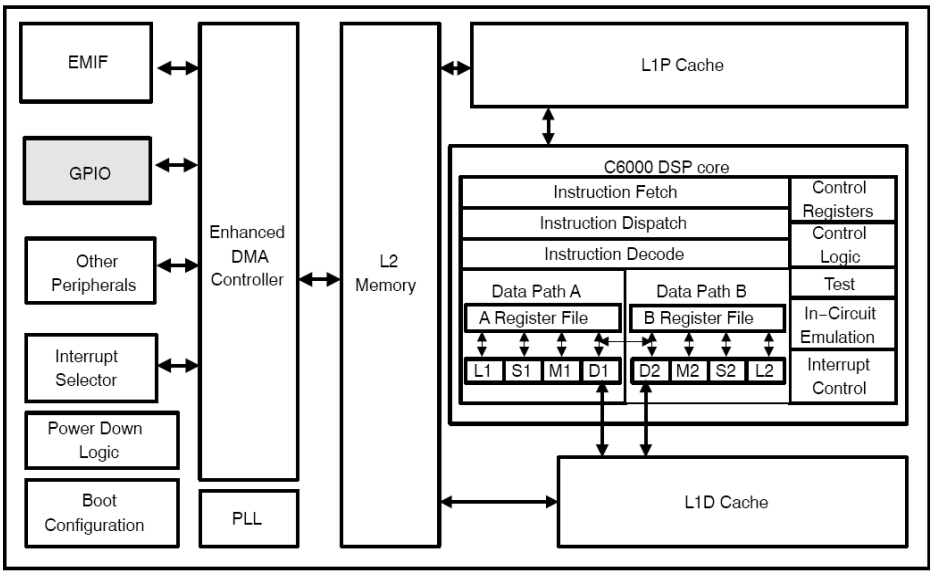


Figure 2.1 TI TMS320C64x™ DSP series a block diagram (adapted from [49])

2.2.4. Application Specific Integrated Circuit (ASIC)

A group of devices dedicated for high volume production are Application Specific Integrated Circuits. These devices only contain the components that are very specific and necessary for performing only one given task. Different design and manufacturing methodologies exist, ranging from full custom ASIC design to structured ASIC design. The major differences are in the usage of predefined macros, cell libraries and intellectual property (IP) cores. As in FPGA design these predefined modules can be used to speed up the development process trading against chip area and cost. While in custom ASIC development the granularity of development is to define the characteristics of metal layers in the semiconductor, in structured ASIC design a set of predefined characteristics and their semiconductor implementation are given in advance (which in turn reduces development time considerably). At the very high-end structured ASICs are also sometimes referred as SoCs if complete DSP cores and modules for interface functionality are included in the design.

The major benefits of ASICs are their asymptotically decreasing cost when manufactured in high numbers, a little increase in speed over FPGAs and a decrease in power consumption to a minimum. However, the biggest problem with ASICs is their design and production cycle and the non-recurring engineering costs, which can easily exceed 1 million dollars. Furthermore the static layout causes big problems as a redesign due to bug fixing is a costly exercise. However, as development tools get better also ASIC design and development becomes easier. As an example mobile phones are a representative application for ASICs as the worldwide sale in high volumes and the well-defined task to perform easily balances the concerns in manufacturing and design. Producers of ASICs include Altera, Fujitsu, Infineon or NEC among others.

2.2.5 System on Chip (SoC)

Microcontrollers and System-on-Chip platforms form the biggest group of embedded systems nowadays, being part of almost any electronically device in our environment. Microcontrollers are some type of microprocessor, additionally including memory resources for program and data storage, timers and external, typically serial, interfaces. Microcontrollers emphasize the aspects of cost-effectiveness, high integration, low power consumption and self-sufficiency. While microcontrollers are single physical packages aimed at performing small-sized tasks, the term System-on-Chip refers to the integration of an entire system - or all electronic circuits needed for performing a given application - into one single chip. Usually, System-on-Chip platforms are combinations of a core processor, a set of interfaces and a selection of external controllers in one physical package. Many SoCs consist of a General Purpose Processor (ARM), which can be a RISC processor like an ARM or a PowerPC, or can also be an x86-based processor like the Intel Celeron M. While the ARM is mainly included to perform operation system tasks, SoCs mostly contain one or more DSP units dedicated to the real signal processing tasks. These types of devices are also called Media Processors.

In the last few years, media processors are becoming more and more important as they are integrated for active video processing or streaming in many devices of

everyday use. Target applications are present especially in mobile phones, PDAs, handheld video players, set-top boxes and in automotive engineering. In the context of image processing, these devices feature special properties and capabilities, such as video en- and decoding support in hardware and limited controlling mechanisms for external devices such as cameras. Though, on these special platforms only little image processing capabilities are usually available as they are tailored to meet a special application in a single or a set of appliances. However, the principles of SoCs make them also appealing to the designers of Smart Cameras as all necessary features are provided by a SoC solution. Peripherals for system integration, interfaces for connecting video sources and an extendable architecture to include more DSP units for even higher signal processing power make this set of devices a good choice as a base for building a video processing system.

Several manufacturers of Microcontrollers and SoC platforms exist. The biggest ones are TI, Analog Devices and Intel as a vendor of the XScale driven devices. Further vendors are AMD which is producer of the Geode processor, Atmel, STMicroelectronics, Freescale and Cirrus Maverick among others.

2.2.6. Summary

The architectures described above, especially DSPs, FPGAs, ASICs and SoCs have their own field of application, their own roots and goals and obviously several benefits and disadvantages.

The choice of which technology to use for a self-made prototypical setup is manifold, and most times, the final decision is based on a detailed planning process applying Hardware-Software Co-Design principles. General rules of thumb to select the right technology for a dedicated task were proposed in the work of Kisaficanin [40]. The main aspects considered here, are
The available time for development (Time-To-Market),

- the expected and required funds,
- the targeted area of application, and
- The expected and desired volume of production.

For a relatively small volume of less than 1.000 units per year general-purpose computers and FPGAs are a good choice considering their price and the reduced need for hardware dependent development. At the high end of production of more than 100.000 units per year, the usage of a tightly tailored ASIC is reasonable as the initial development costs and the need for special developing are notably high, but the deployment in high volumes justifies adequate investments. In the mid-range DSPs and media processors are good selections as they form a good trade-off between programming flexibility and the amount of necessary specific development. Most smart cameras are based on a combination of several of these devices manufactured in SoC technology because a smart camera implicitly needs peripherals and interface but also signal processing power delivered by DSPs. Figure 2.2 shows a comparison of the most prominent smart camera implementation architectures and their properties.

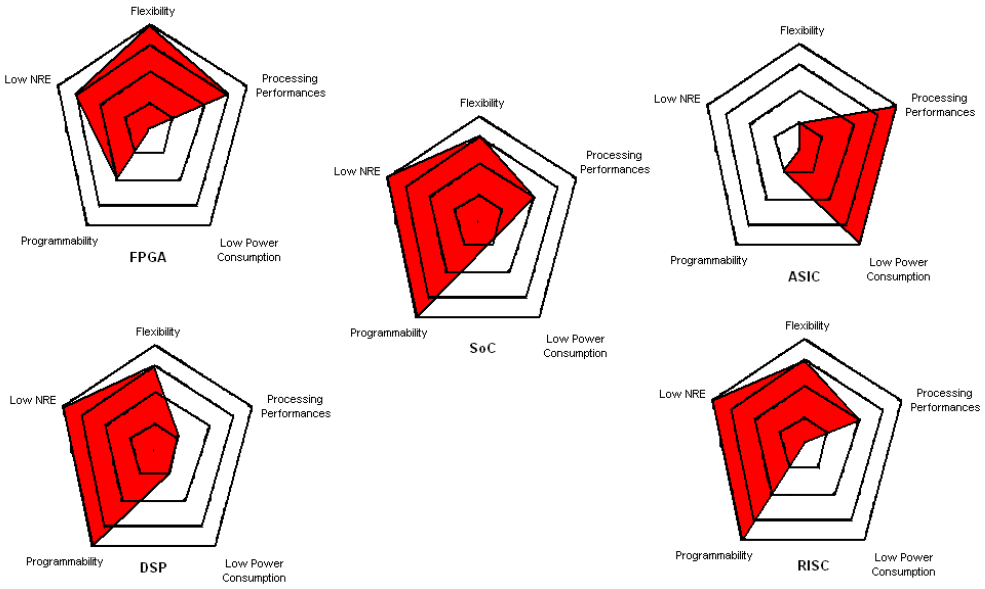


Figure 2.2 A Comparison of the most prominent smart camera implementation architectures and their properties (adapted from [19])

2.3. Examples of Smart Cameras

2.3.1 The CMUCam

In 2002, The Carnegie Mellon University Camera was originally presented as a low cost embedded vision platform. It was commercially available right from the beginning for nearly $100. The first version proposed in 2002 by Rowe, Rosenberg and Nourbakhsh [3] was based on three chips mainly, an Omnivision OV6620 CMOS camera sensor, an Ubicom SX28 microcontroller and a simple level shifter for serial communication.

In 2005, the second generation of the platform was presented. One enhancement compared to the original version was the use of an Ubicom SX52 microcontroller still running at 75 MHz but now offering 262 bytes of SRAM [4]. The power consumption at runtime was about 850mW, while the price for the platform was about $199.

The third and actual version of the system is available since the beginning of 2007 for approximately $239 [2]. Now another microcontroller is used, namely a NXP LPC2106 which is a 32-bit 60 MHz ARM7TDMI processor with 64k bytes of RAM. Another big advantage is the use of a Multimedia Card (MMC) interface which can be used to read and write files and a CIF resolution color image sensor (352x288). The overall system consumes between 300 and 500 mW of power, depending on the operation mode.

2.3.2 The WiCa

The Wireless Camera was developed at NXP Semiconductors and is a smart camera platform featuring two connectors which can be used freely to capture from one single camera or to form a stereo camera setup [5, 6]. The cameras deliver images in VGA resolution of 640x480 pixels. The SIMD processor is a Philips Xetal IC3D device whose core is a linear array of 320 single RISC processors for low-level image processing tasks. The ATMEL 8051 microcontroller is dedicated to multiple tasks. It mainly controls the IC3D processor, communicates with the RISC processor

array and takes care of program flow and video synchronization. The dual-port RAM available on the platform provides a total of 128k bytes of memory, which is separated into two banks with a block of 64k bytes each. Several algorithms have been proposed for face detection or human gesture analysis where the main focus is to exploit algorithm level parallelism [5, 12].

2.3.3 The Cyclops

In respect of large sensor networks and energy-aware smart sensors, Rahimi et al. proposed a highly power-efficient smart vision platform called Cyclops [7]. The camera module is a 352x288 resolution ADCM-1700 CMOS camera from Agilent Technology and can be configured to deliver 8-bit grayscale, 16-bit color or 24-bit color images. The microcontroller unit is a ATMEL ATmega128L running at 7.37 MHz, which offers 4k bytes of internal SRAM memory and is extended to 64k bytes total memory on the external SRAM block. The processor is mainly used to time and coordinate external and internal events and interrupts. The Xilinx XC2C256 CoolRunner CPLD serves as a frame grabber which can fulfill the high demands on fast data transfer and address generation. The platform is designed to work in larger sensor networks, thus it is highly power-aware and energy consumption is in the range of a few mW.

2.3.4 The SmartCam

The SmartCam is a low-power, high-performance embedded vision system. It consists of a set of individual components [8, 9]. The prototype is based on an Intel IXDP425 development board equipped with a 533 MHz XScale network processor, which makes several communication systems such as Ethernet, USB, RS232, WLAN, and GSM possible to be exploited. The board features 256M bytes of RAM and four PCI slots, an on-chip Ethernet connection and multiple serial ports amongst others. For the main processing task, each PCI slot can host an ATEME Network Video Development Kit (NVDK) board which consists of 264MB of memory and TI TMS320C6416 DSPs running at 1 GHz. The plausibility of the concept was

demonstrated on a vehicle detection and tracking application for tunnel safety [9, 13, and 14].

2.3.5 The MeshEyeTM

The MeshEyeTM platform was proposed as a single node for larger distributed smart camera networks [11]. The main platform can host up to 8 low-resolution imagers, however, the prototype contains only two 30x30 pixel optical mouse sensors and one 640x480 VGA resolution color sensor. The main processing core is an Atmel AT91SAM7S microcontroller which is a ARM7TDMI 32-bit RISC processor running at 55 MHz similar to the CMUCam3 platform the system contains a MMC/SD card interface allowing for easy memory expansion. Another important feature is the use of a TI CC2420 2.4 GHz IEEE 802.15.4/ZigBee-ready RF transceiver to connect the mote to other notes in a larger network.

2.3.6 Summary

The smart cameras described above, especially The CMUCam, The WiCa, The Cyclops, The SmartCam and The MeshEyeTM differ in hardware whereas their goals are similar. Table 2.2 shows a comparison of smart cameras. All of the smart camera implementation has a wireless node for network connection and they have powerful processing units that generally depend on 32-bit architecture that enables faster data processing. In some architecture a second processor is used for additional processing and control. Since most processors have small internal memories, additional external RAM and Flash memories are used for frame buffering and permanent data storage. Resolution of image sensors changes between one CIF and VGA. Some smart cameras use two image sensors to provide binocular vision. Wireless connection is achieved by an Ethernet or RF radio link based on IEEE 802.15.4. Almost all of the smart cameras implementation has an image processing application to add a value to captured frames from image sensor.

Finding the right and meaningful hardware and configuration to implement a smart camera application on is a matter of ongoing research. However, one can summarize that, in general, smart cameras are platforms that consist of

Table 2.2. A Comparison of Smart Cameras

	Proposal Year	Processing Unit	Running Speed	Memory	Image Sensor Resolution	Algorithm	Communication
The CMUCam	2002	Ubicom SX28	75 MHz	136 SRAM	143 * 80	Simple color blob detection (143*80 pixel, fps = 16.7)	IEEE 802.15.4 compliant (Telos mote)
	2005	Ubicom SX52	75 MHz	262 SRAM	143 * 80	Simple color blob detection and Color Statistic Calculation (143*80 pixel, fps = 16.7)	
	2007	NXPLPC2106 ARM7	60 MHz	64k RAM	352 * 288	Viola-Jones faces detection (60 * 60 pixel, fps=1)	
The WiCa	2006	SIMD Philips Xetal IC3D and ATMEL 8051	55 MHz	2*64k RAM	640 * 480	-	RF ZigBee
The Cyclops		XC2C256 CoolRunner CPLD and ATmega128L	7.37 MHz	4k internal, 64k external SRAM	352*288	Object Detection (128 * 128 pixel, fps=4) and Hand gesture Recognition (128 * 128 pixel, fps=2)	IEEE 802.15.4 compliant (MICA2 Mote)
The SmartCam		XScale Network Processor and TI TMS320C6416 DSP	533 MHz (XScale) and 1 GHz (DSP)	256 MB (XScale) and 264 MB (DSP)	640x480	Vehicle Detection and Tracking Application for Tunnel safety	Ethernet, Wireless GPS/GPRS Radio
The MeshEyeTM		ATMEL AT91SAM7S ARM7	55 MHz	64 KB SRAM and 256 KB Flash; external MMC/SD Flash	640 * 480 and 30 * 30	Object Detection and Tracking	RF ZigBee

- one high resolution (generally VGA) or multiple high and low resolution image sensors , which are growing steadily in their resolution,
- a 32-bit architecture microcontroller (generally ARM), which is performing low-level operation system tasks and a high performance computational processor, which is a DSP in the majority of cases
- extra RAM and Flash memory resources for frame buffering and permanent data storage

- Some type of communication module, which is mostly wired or wireless Ethernet.

After a summary of smart cameras in the literature we can list the below performance parameters to evaluate a smart camera application.

- Resolution of image sensor
- Processing power for implementation of high level image processing algorithms
- Low Power Consumption
- Low-bandwidth Network connection for streaming video
- User friendly control interface through network
- Low cost
- High Flexibility

A RISC architecture processor is the most appropriate processor to realize the Low-bandwidth Network connection, user control interface over network and high processing power performance parameters of a smart camera. ARM core can be good choice for RISC architecture processor because of its low power and price. Also a SIMD processor is necessary for implementation of high level image processing algorithms on high resolution image sensor outputs. Because of flexibility option a DSP processor can be good solution for implementation of a right and meaningful smart camera application. Therefore selection of ARM + DSP SoC processor that has image capturing and displaying features can be a compact and effective solution to implement a low-cost smart camera application.

2.4. Basic blocks of the implemented system

Figure 2.3 shows the flow chart of the implementation steps of a low-cost smart camera application on an ARM + DSP Soc Platform which is performed in this thesis. As can be seen from the figure, the first steps are defining the features of smart camera application. In this work a VGA resolution, YUYV pixel format A4Tech USB camera is selected as input sensor. For displaying captured frames

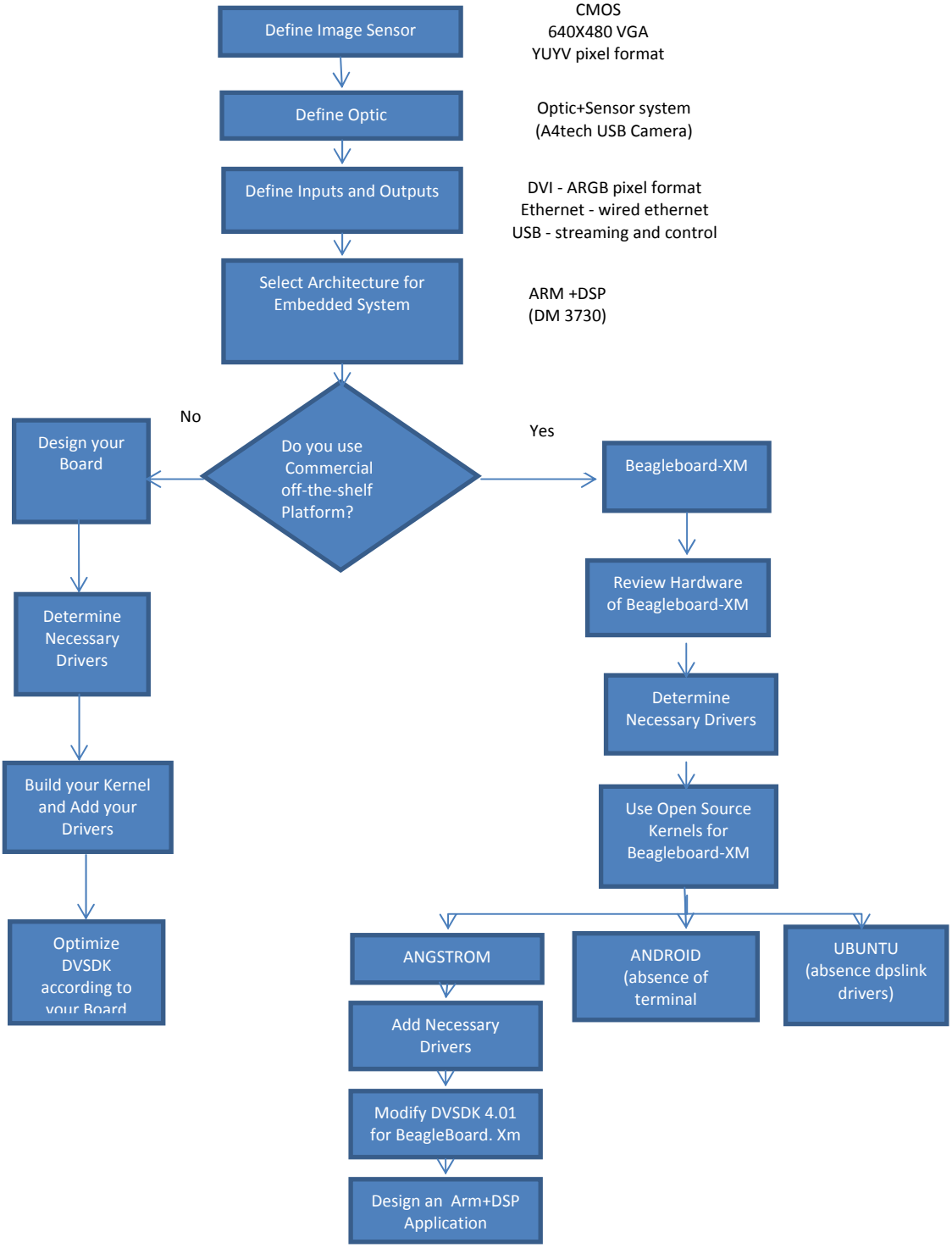


Figure 2.3 The Implementation Steps of a Low-Cost Smart Camera Application on an ARM + DSP Soc Platform

ARGB pixel format DVI output is selected. For streaming video and controlling the camera, a wired or wireless network connection is desired. For implementation of these features ARM + DSP architecture is selected. After selection of architecture, a prototype board can be developed for implementation. However, developing a new platform and applying necessary drivers on designed platform requires a NRE and time for implementation. So some low-cost COTS systems can be used for implementation. In this work Beagleboard-XM platform is selected for implementation, because of its powerful ARM + DSP SoC processor, low price and networking, capturing and displaying capabilities. The core of this platform is a new technology processor that is used in some cell phone applications and its new generations are in the roadmap of TI. Therefore implementation of smart camera on this platform can also be ported to the new generation boards of TI. However, this board is a COTS system and it is not supported officially by TI DVSDK. Therefore, to start up the platform, we must review the hardware of platform and determine the necessary drivers for the implementation of smart camera application. And then we must find an open source embedded OS that support necessary drivers and port it to the ARM core of processor. After that step we must modify the DVSDK that is developed for OMAP3530 EVM board according to our platform. After the modification, we must develop an application that runs on both ARM and DSP cores by using modified DVSDK.

CHAPTER 3

HARDWARE OF THE SELECTED PLATFORM

In this thesis, to design a low cost smart camera with a sufficient processing power, DM3730 processor, which is a SoC processor with 1 GHz ARM® Cortex™-A8 and 800 MHz TMS320C64x+™ DSP core, is selected from the digital media processor family of TI. To setup a low cost smart camera system with DM3730 processor the hardware of Beagleboard-xM platform is selected as the hardware of our system.

In this chapter, very brief information is given about the hardware of the selected platform, namely the Beagleboard-xM board and its core processor DM3730 architecture. This chapter describes the peripheral units on board and gives brief information about the architecture of the main cores and co-processors included in DM3730 processor. Development and implementation of algorithms that will be run on DM3730 target platform is so much related with the peripheral units of the board and the architecture of the processor. This chapter consists of two main sections of which the first one describes the Beagleboard-xM board, and the second one describes the DM3730 Architecture.

3.1. Overview of the Beagleboard-xM Platform

Beagleboard-xM is a single-board computer system based on TI's DM3730 processor. It is able to achieve laptop-like functionality thanks to its performance and to the expansion interfaces and peripherals available on the board. Although it has a high performance processing cores, it is at the same time a low-power and low-cost embedded computer system. The BeagleBoard-xM is not intended to be a development environment for a final product. It is designed as the basis for an

experimental and test platform [27]. The board consists of TI DM3730 processor, memory with 512MB of low-power DDR RAM, DVI-D and S-Video outputs, stereo audio inputs and outputs, high capacity SD card slot, JTAG input, digital camera input port, four-port USB hub and a 10/100 Ethernet port, while maintaining a tiny 3.25" × 3.25" footprint. At the time of writing, the cost of a BeagleBoard-xM is US$ 149. The Beagleboard used during this project was the version BeagleBoard-xM Rev B. Figure 3.1 shows the key features of this board.

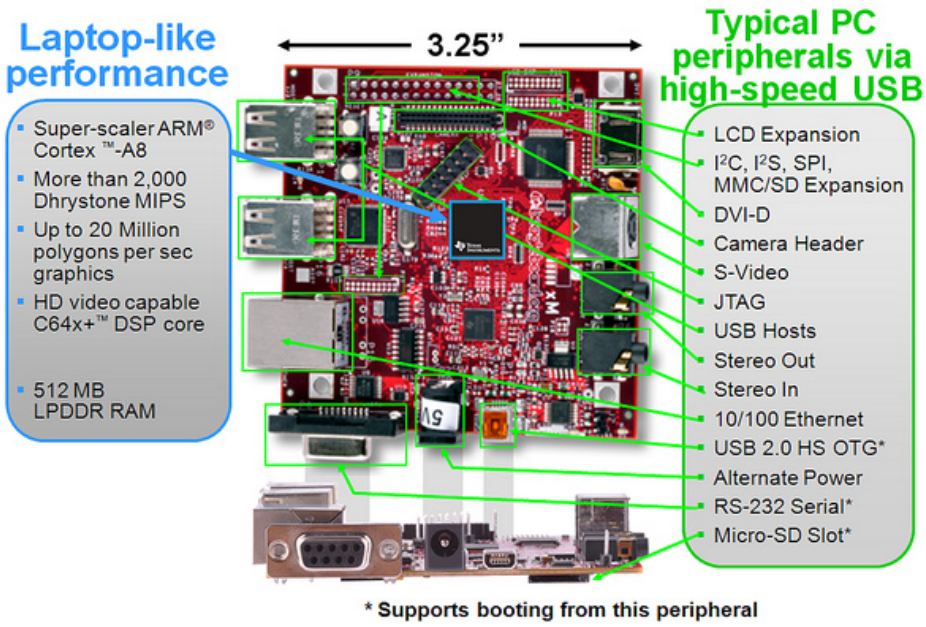


Figure 3.1 Key features of the Beagleboard-xM Board [53]

The core of the BeagleBoard-xM Rev B is the DM3730 processor packaged in a Package-on-Package (POP). In the POP packaging techniques, the memory chips are mounted on top of the processor package. In the Micron POP there is 4 Gb MDDR SDRAM x32 (512MB @ 200MHz). This device is the only on-board memory available. Nevertheless, since Beagleboard-xM has standard interfaces for connecting external storage devices. Additionally, it is possible to extend the system memory by means of SD or MMC cards or by a USB flash or hard drive. However, accessing these external memories will be quite slow. TI’s TPS65950 chip is used for

power management. The TPS65950 is a Power Management Multi-Channel IC (PMIC) solution. In a single IC a multichannel power-management device and an audio coder/decoder are integrated. This chip is in charge of controlling the power for both peripherals and the DM3730 processor. A 14-pin JTAG interface is also provided to permit software debugging and programming of the on-chip FLASH memory (i.e., to install a system image or boot loader). Support for RS232 via UART3 is provided by a DB9 connector. Through this interface it is possible to access the Beagleboard-xM using a DB9 flat serial cable.

3.2. DM3730 Processor

DM3730 is a high-performance, digital media processor based on OMAP™ 3 architecture provided by Texas Instruments for 2008. This media processor is designed to provide video, image and graphics processing [28]. This processor is offered for streaming video, 2D/3D mobile gaming, video conferencing, high-resolution still image, Video capture in 2.5G wireless terminals, 3G wireless terminals, and rich multimedia-featured handsets, and high-performance personal digital assistants (PDAs) [28]. Table 3.1 gives the distribution of the ARM core processors of TI according to application area.

Table 3.1. TI's ARM Core Processor Distribution

	General Purpose ARM Only	**General Purpose ARM+DSP**	**Video Oriented ARM/ ARM+DSP**
Family	**Sitara (AM)**	**Integra (C6L, C6A8)**	**Davinci (DM)**
ARM926	AM1705 AM1707 AM1806 AM1808	OMAP-L137 OMAP-L138 C6L13X	DM355 DM365 DM644X DM6467
Cortex A8	OMAP3503 OMAP3515 AM3505 AM3515 AM3703 AM3715 AM3982 AM3984	C6A8187 C6A8168	OMAP3525 OMAP3530 DM3725 **DM3730**

In [28], general subsystems of the device are summarized as:

• Microprocessor unit (MPU) subsystem
• IVA2.2 subsystem
• SGX subsystem
•Camera image signal processor (ISP)
• Display subsystem
• System Direct Memory Access (SDMA)
• Interprocessor Communication (IPC)

In this thesis, most of the subsystems of the device are used to implement a smart camera application on DM3730 processor and to increase the performance of it. This processor is a double core processor, namely 1 GHz ARM Cortex™-A8 core with NEON™ SIMD coprocessor as the general purpose main processor unit of the system and 800 MHz advanced very-long-instruction-word (VLIW) – TMS320C64x+™ DSP core [28]. This processor is a low power IC that is manufactured by TI (Texas Instruments) especially for hand-held devices, gaming consoles, video and image processing, and communication devices. In figure 3.2, generalized block scheme of DM3730 processor is given [28]. As can be seen in figure 3.2 all of the subsystems connect to a common interconnect network by 32 bit or 64 bit read/write registers. This network enables the data transfer and communication between subsystems. There is a 32 KB Read Only Memory (ROM) of processor. This ROM programmed at factory to load the startup files of kernel that will be installed on ARM core of the processor from different peripherals like NAND memory, Multi Media Card (MMC), USB etc. The processor has also a 64KB On-Chip Static Random Access Memory (SRAM). This shared SRAM is used as cache memory by ARM core and by other subsystems of the processor. Use of SRAM instead of Dynamic Random Access Memory (DRAM) increases the platform's read/write speed during its operation. Because SRAM stores a bit of data by using the state of a flip-flop while DRAM uses transistor-capacitor pairs for storage. Flip-flops are generally faster and require less power than transistor capacitor pairs to store a bit of data. However production of flip-flops is more expensive.

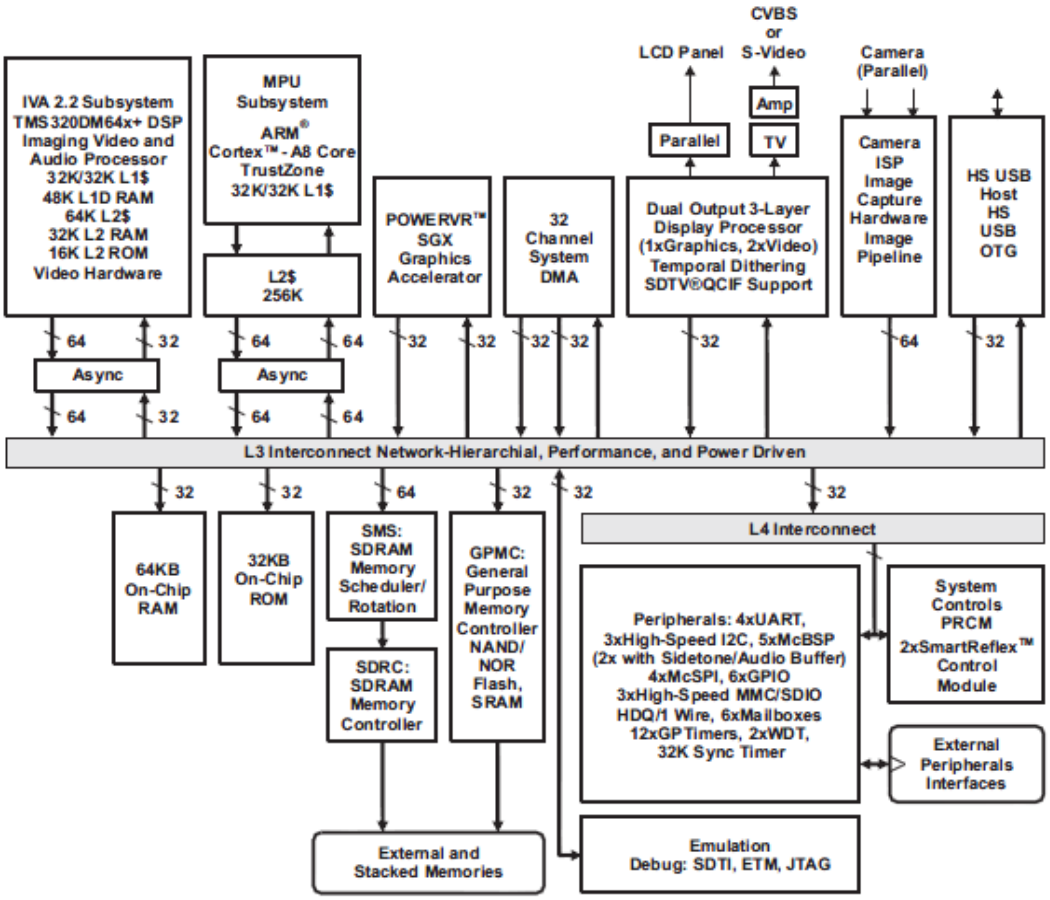


Figure 3.2 Functional block diagram of DM3730 media processor of the Beagleboard-xM Board

3.2.1 Microprocessor Unit (MPU) Subsystem

The MPU subsystem integrates the following:

- ARM® Cortex™-A8 core
- ARM Version 7™ ISA: Standard ARM instruction set + Thumb®-2, Janelle® RCT Java accelerator, and media extensions
- NEON™ SIMD coprocessor (VFP lite + media streaming instructions)
- Cache memories
    - Level 1: 32KB instruction and 32KB data 4-way set associative cache, 64 bytes/line

- Level 2: L2 cache and cache controller are embedded within the ARM Cortex-A8 CPU - 256KB, 8 ways associative, 64 bytes/line, parity and error-correction code (ECC) supported
- Memory management unit (MMU) and translation look-aside buffers (TLBs)
- Interrupt controller (MPU INTC) of 96 synchronous interrupt lines
- Debug, trace, and emulation features: ICE-Crusher, ETM, ETB modules.

ARM Cortex A8

The Cortex-A8 processor is a microprocessor designed by ARM Holdings based on the ARMv7-A, a 32-bit Reduced Instruction Set Computer (RISC). Figure 3.3 shows the performance, functionality and capability distribution of the ARM CPU processor cores of TI.

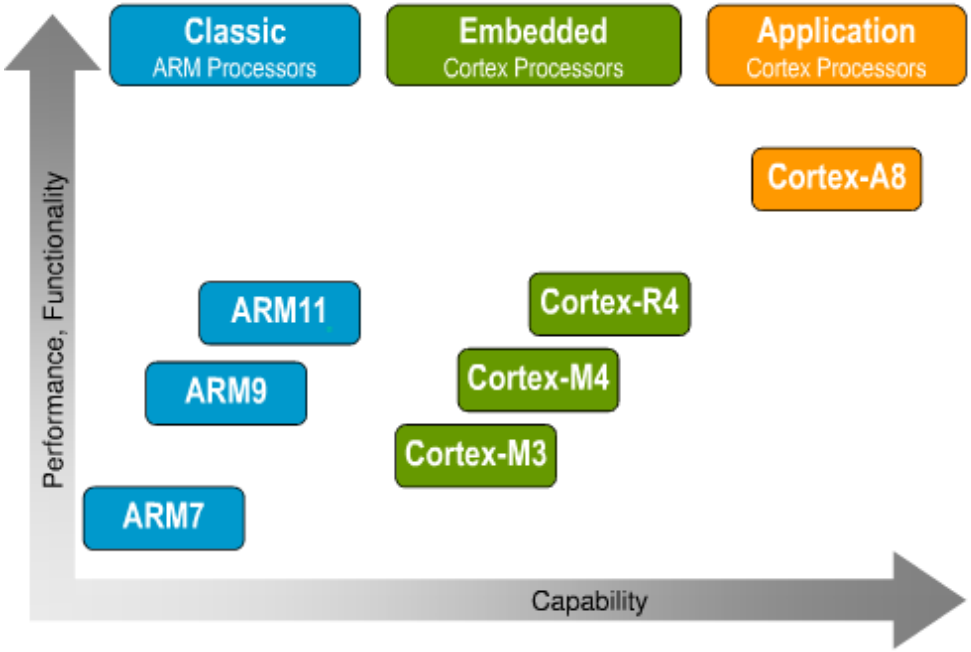


Figure 3.3 The TI's Linux supporting ARM CPU processor cores

The ARM9 (375-450Mhz) is the low cost and most widely used processor core in the market while the Cortex-A8 (600Mhz-1.5Ghz) is a low-power, high-performance single core microprocessor designed for portable devices having the following main features[29]:

• frequency from 600 MHz up to 1.5 GHz;
• Dhrystone performance is 2.0 DMIPS / MHz; Dhrystone refers to the performance as measured by means of the Dhrystone benchmark and DMIPS refers to Dhrystone Million Instruction Per Second. This benchmark was created in 1984 by Dr. Reinhold P. Weicker and it tests integer computation performance of a processor without any floating-point operations. It became popular since it is free of charge, while the most popular benchmarks belonging to the SPEC suite are quite expensive. However, it has several notable limitations as it does not consider many important factors such as the RISC nature of the processor, multitasking, memory hierarchy, and advanced processor designs (as found in superscalar and VLIW computers)
• a superscalar processor with two different pipelines. The first pipeline is in charge of the execution of integer ARM instructions. The second pipeline is a NEON pipeline for the execution of advanced SIMD and Vector Floating Point (VFP) instruction set;
• dynamic branch prediction with branch target address cache, global history buffer, and 8-entry return stack;
• Memory Management Unit (MMU) and two 32 entries Translation Look-aside Buffers (TLBs) for data and instruction (respectively);
• static and dynamic power management;
• L1 instruction and data cache of 16KB or 32KB (configurable size). The L1 cache is integrated on-chip so that it can be accessed in a single clock cycle;
• L2 cache up to 1 MB configurable size with parity and Error Correction Code (ECC) techniques implemented. The L2 cache is banked so that only the bank in question is activated for increased power saving.

Three technologies implemented in the Cortex-A8 are noteworthy for our project. The first one is the Thumb-2 instruction set, an extension of the earlier Thumb instruction set. When the processor is in the Thumb instruction set state, it is able to

execute variable-length instructions. In this state the instruction length is not fixed at 32 bits, but can be either 16 bits or 32 bits temporarily breaking the RISC model. The main advantage is to reduce the instruction code size. This aspect can be very important when dealing with embedded devices with a limited amount of main memory. The short instructions (16 bits) utilize implicit operands or limitations of the more general instruction set. In fact only a limited set of operations can be expressed through these 16 bits instructions. Thumb-2 is an enhancement of the Thumb technique as it introduces the possibility to interleave 16 bit instructions with 32 bit instructions while still in the Thumb instruction set mode.

The second technology is the Vector Floating Point (VFP) architecture. This consists of a coprocessor extension of the ARM architecture capable of executing floating point operation with half, single, and double precision. It is fully compliant with the IEEE 754 floating point format.

Third is the NEON technology [19], a 128 bit SIMD architecture extension. Thanks to this, the Cortex-A8 is able to execute advanced SIMD instructions. SIMD is a class of parallel execution that exploits parallel operations on data. NEON is considered short-vector architecture, this means that registers are considered as vectors of elements of the same type of data and the same operation is performed in parallel in different lanes. The data types available in this SIMD instruction set are signed and unsigned 8 bits, 16 bits, 32 bits, 64 bits and single precision floating point. This technology provides a significant acceleration in the performances of multimedia and signal processing algorithms such as video encode/decode, 2D/3D graphics, gaming, audio and speech processing, and image processing. The motivation for this is that in such applications it is very common that an operation is to be performed on an array of data, this is naturally highly parallel, hence it is well suited to a SIMD instruction set.

3.2.2. IVA2.2 Subsystem

The device includes the high-performance Texas Instruments image video and audio accelerator (IVA2.2), based on the TMS320DMC64X+ VLIW digital signal processor (DSP) core. The internal architecture is an assembly of the following components:

- High-performance TI DSP (TMS320DMC64X+) integrated in a megamodule, including local L1/L2 cache and memory controllers
- L1 RAM and L2 RAM and ROM
- Video hardware accelerator module, including local sequencer
- Dedicated enhanced data memory access (EDMA) engine to download/upload data from/to memories and peripherals external to the subchip
- Dedicated memory management unit (MMU) for accessing level 3 (L3) interconnect address space
- Local interconnect network
- Dedicated modules SYSC and WUGEN in charge of power management, clock generation, and connection to the power, reset, and clock manager (PRCM) module

VLIW TMS320DMC64X+ DSP

The TMS320C64x+ DSP is the latest and most widely used fixed point DSP from TI's C6000 DSP family CPU. The TI's other most widely used DSP for latest releases is C674 fixed and floating point DSP. Figure 3.4 shows the block diagram of C64x module.

The TMS320C64x+ DSP is a VLIW architecture that executes up to eight 32-bit instructions per cycle ([31]). This is possible because in the CPU architecture 8 functional units are present. These functional units are divided into:

• 6 ALUs (single 32 bit, double 16 bit, or quad 8 bit arithmetic operations per clock cycle);
• 2 multipliers (two 16x16 bit multiplies or four 8x8 bits multiplies per clock cycle).

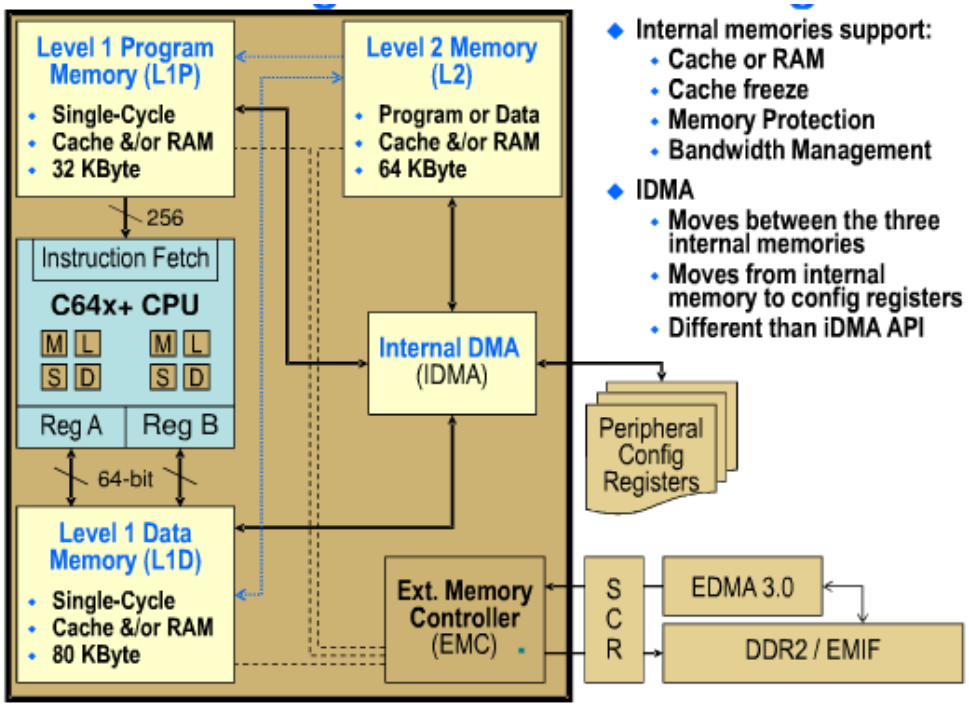


Figure 3.4 C64x module block diagram

This DSP processor includes sixty-four 32-bit general purpose registers. The TMS320C64x+ benefits from its VLIW architecture. The main advantage is due to the grouping of instructions. This reduces the number of instructions that are produced for a given amount of code (hence less memory is needed), thus the number of fetches from the instruction memory is reduced (resulting in less power consumption), and the execution time is reduced by exploiting the instruction VLIW architecture is a static way for exploiting the instruction level parallelism of a program. The compiler packs a group of instructions that can be executed in parallel into longer instructions at compile time. This means that when the CPU executes one VLIW, several single instructions are executed in parallel at each clock cycle. Due to the static nature of this technique, it is not possible to exploit optimally all of the potential instruction level parallelism.

The C64x+ is a fixed-point DSP. This implies that floating point operations are not executed in hardware, but rather are emulated by software. Nevertheless software performance can be improved by using TI's IQmath Library for C64x+ (details in

[41]). This library is a collection of highly optimized mathematical functions (written as C/C++ routines) aimed for porting floating-point algorithms to fixed-point code that can be executed by the C64x+ hardware. Another useful tool for improving performance of the software running on the DSP, is the TI C64x+ DSPLIB [42]. DSPLIB is a collection of highly optimized C-callable routines that are written in assembly code. Most of these routines are used for signal processing, especially in computationally expensive real-time applications. The functions in the DSPLIB are organized into seven different categories:

- Adaptive filtering
- Correlation
- Fast Fourier Transform (FFT)
- Filtering and convolution
- Math
- Matrix
- Miscellaneous.

3.2.3 Camera Image Signal Processor

The camera Image Signal Processor is a key component for imaging and video applications such as video preview, video record, and still-image capture with or without digital zooming.

The camera ISP provides the system interface and the processing capability to connect RAW image-sensor modules to the device.

The camera ISP implements 12 bit parallel interface. Their purpose is to act as a physical connection between the outside pins for connecting external sensors and the internal receivers. By configuring the outside physical layer and feeding the receivers, the camera ISP supports up to two simultaneous pixel flows from external sensors. Only one of the data flow can use the Video processing hardware while the other must go to memory. It can support up to 150 MHz Pclk when operating in 8 bit mode and 75 MHz for 12 bit interface.

3.2.4 Display Interface Subsystem

The display interface subsystem provides the logic to display a video frame from the memory frame buffer (either SDRAM or SRAM) on a liquid-crystal display (LCD) panel or a TV set. The display subsystem integrates the following elements:

- Display controller (DISPC) module
- Remote frame buffer interface (RFBI) module
- NTSC/PAL video encoder

The display controller and the DSI protocol engine are connected to the L3 and L4 interconnect; the RFBI and the TV out encoder modules are connected to the L4 interconnect.

3.2.5 2D/3D Graphics Accelerator (SGX)

The 2D/3D graphics accelerator (SGX) subsystem accelerates 2-dimensional (2D) and 3-dimensional (3D) graphics applications. The SGX subsystem is based on the POWERVR® SGX core from Imagination Technologies. SGX is a new generation of programmable PowerVR graphic cores. Targeted applications include feature phones, PDA, and hand-held games.

The SGX graphics accelerator efficiently processes a number of various multimedia data types concurrently:

- Pixel data
- Vertex data
- Video data
- General-purpose processing

This is achieved using a multithreaded architecture using two levels of scheduling and data partitioning enabling zero overhead task switching.

The SGX subsystem is connected by a 64-bit master and a 32-bit slave interface to the L3 interconnects.

3.2.6 System Direct Memory Access (SDMA)

The System Direct Memory Access (SDMA), also called DMA4, performs high-performance data transfers between memories and peripheral devices without microprocessor unit (MPU) or digital signal processor (DSP) support during transfer.

3.2.7 Interprocessor Communication (IPC)

Communication between the on-chip processors of the device uses a queued mailbox-interrupt mechanism. The queued mailbox-interrupt mechanism allows the software to establish a communication channel between two processors through a set of registers and associated interrupt signals by sending and receiving messages (mailboxes).
The mailbox module includes these features:

- Two mailbox message queues for microprocessor unit (MPU) and imaging video and audio accelerator (IVA2.2) communications.
- Flexible assignment of receiver and sender for each mailbox through interrupt configuration
- 32-bit message width
- Four-message FIFO depth for each message queue
- Message reception and queue-not-full notification using interrupts
- Support of 16-/32-bit addressing scheme
- Power management support
- Automatic idle mode for power savings

CHAPTER 4

SOFTWARE OF SYSTEM

A variety of operating systems can be executed on OMAP processors whereas there is only one choice for DSP core. However the OS ported to ARM core must support the necessary drivers to communicate with the DSP core. The ARM core is responsible from most of the platform functions including the control and coordination of the DSP. DSP is used for only real-time computation and I/O. It leaves the other tasks to the ARM core. The operating systems that can be executed by the ARM core are Linux®, Microsoft's Windows, Symbian OS™, Mobile™, and Android™. In this thesis, the Linux Angström distribution is used as the ARM core OS and DSP/BIOS Real-Time OS is used for the DSP core. Figure 4.1 shows the distribution of OS implemented on ARM and DSP cores.

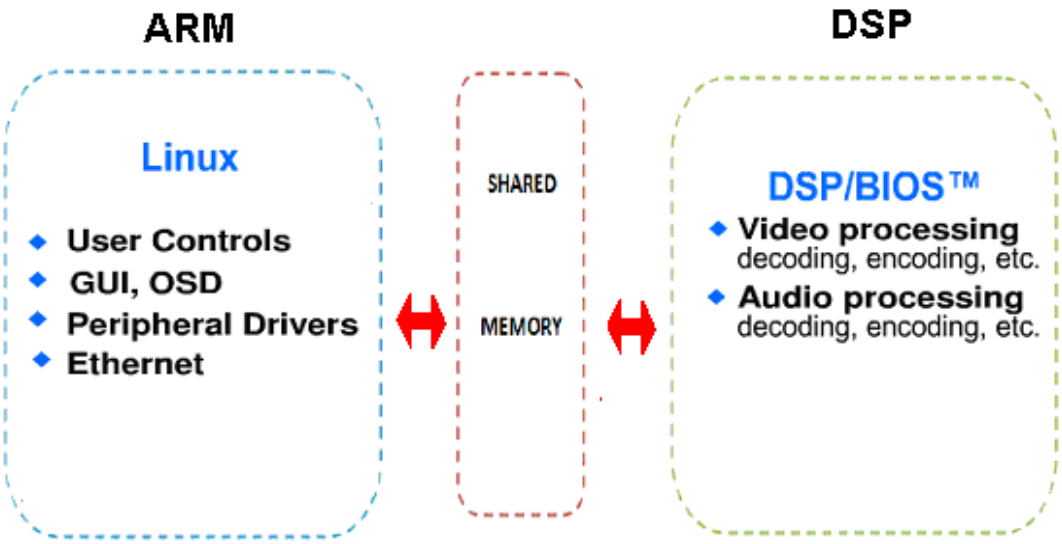


Figure 4.1 OS implemented on ARM and DSP cores of processor

In this chapter, software of the system implemented on the ARM and DSP cores of the DM3730 processor is introduced. Setup and configuration of the necessary files that ARM core uses at start up are described. This chapter consists of three main sections. First section describes the OS that implemented on ARM core. In this section setup procedures of OS for ARM core are introduced. Second section describes the OS implemented on DSP core. In this section setup procedures of OS for DSP core are introduced. Third and last section describes the necessary drivers for smart camera application.

4.1. Setup of OS for ARM Core

Angström is a Linux distribution intended for the embedded devices. It is claimed to be versatile and scalable. It can be installed on systems having at least 4 MB memory. The Angström distribution is based upon the union of the Open Embedded, OpenZaurus, and OpenSimpad projects. The OpenEmbedded Project is a framework to create Linux distribution for embedded systems. An important tool in the OpenEmbedded projects is `bitbake` . This is a tool used to build packages for the embedded distribution by means of cross-compiling. A cross-compiler produces executable code for a platform and a system that is different from the platform hosting the compiler. This technique is fundamental for building of software packages for those systems where compilation is not feasible due to limited system resources, processor speed, and ported compilers and OS. In this thesis Narcissus [12], online image builder for Angström distribution, is used to create the system files of Angström OS compatible with ARM core of the Beagleboard-xM board. To implement an OS to ARM core we need three basic files and a kernel file system namely MLO, u-boot.bin, uImage files and file system. The first three of them must be saved in an 80 MB fat16 or fat32 file system and the kernel file system must be saved in 1 GB or higher ext2 or ext3 file system according to size of file system of OS. These files can be saved on the internal NAND flash of the system. However, in this thesis we saved these files in an 8 GB SD card since Beagleboard-xM does not have a NAND flash on board. This situation decreases the start up speed of the system. Figure 4.2 shows how to port the Angstrom Linux kernel to ARM core of board. Beagleboard-xM board comes with a minimal Angström OS that do not have

most of the necessary packages for video capturing and displaying, IPC, wireless Ethernet and file sharing over internet. In this work, the original system files come with beagleboard-XM platform are replaced by the system files of Angström OS created by Narcissus online image builder tool to provide these necessary packages. In other words, MLO and u-boot.bin files that come with the board are used without any change.

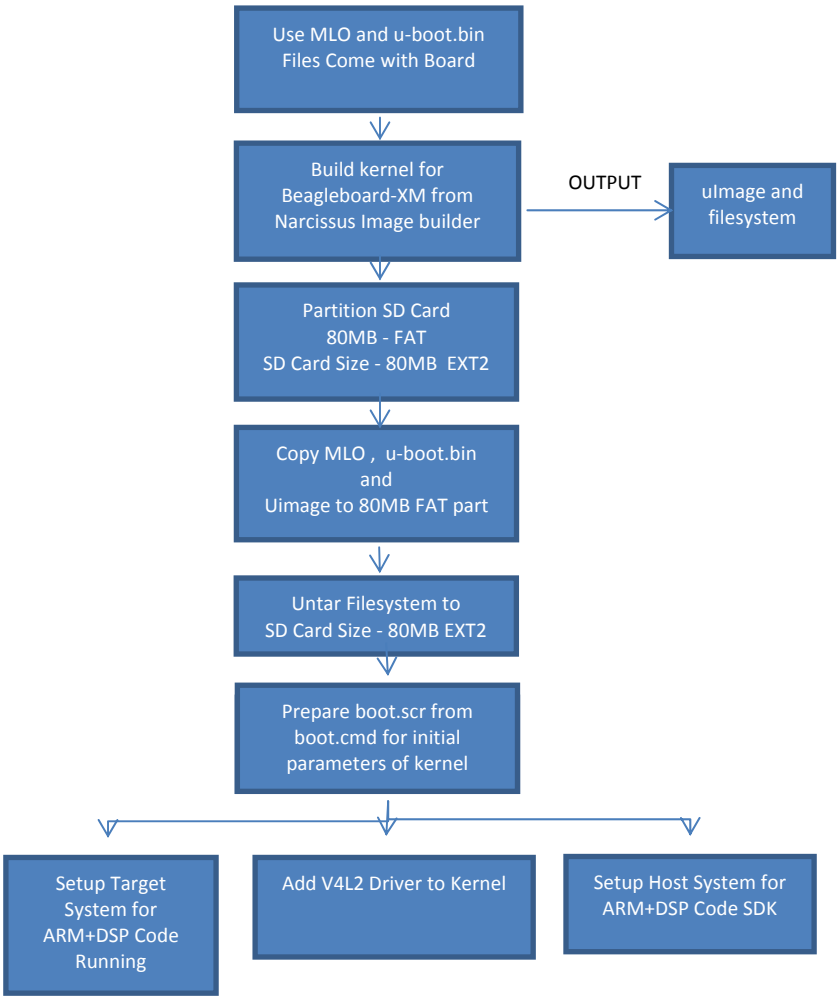


Figure 4.2 Porting Angstrom OS to ARM Core

uImage file and kernel file system of OS are created by using the outputs of the Narcissus online image builder. Narcissus online image builder can create different

combinations of images for different platforms. So some basic settings must be done to get an image that includes necessary packages for Beagleboard-xM platform. Setting details of Narcissus image builder is given in Appendix B. MLO file is created from the x-loader.bin file and it is in the first sector of SD card. MLO is the first file booted by processor. ROM of the DM3730 processor is programmed at the factory to boot the MLO file. It is an initial program loader for embedded devices based on OMAP processor. In this thesis, MLO is set up to load u-boot.bin file from SD card. u-boot.bin file is a board and processor specific file. It determines the settings of peripherals like serial port, Ethernet, video display, display format, RAM mapping for kernel etc and then it boots the user specified files with configured parameters. Therefore settings in the u-boot.bin file are so important to configure the board for desired conditions. A user guide to control the u-boot.bin file parameters is given in Appendix C.

Indeed u-boot.bin file is a minimal kernel with a console for the target board. After startup, setting of u-boot.bin file can be changed from the console by using serial port of the board. All of the changes for desired configuration can be saved to the NAND memory of device. If the platform does not have an internal NAND memory like Beagleboard-xM board then all of the settings must be done for each startup. To do this, a boot script that includes the desired configuration parameters must be written and saved as boot.cmd file. boot.scr file is created from boot.cmd file by using u-boot mkimage tool. This tool can be loaded Angstrom kernel by using opkg install mkimage command. A sample mkimage command to create a boot.scr file from boot.cmd file given below.

```
mkimage -A arm -O Linux -T script -C none -a 0 -e 0 -n "Angstrom" -d
./boot.cmd ./boot.scr
```

u-boot.bin file firstly boots the boot.scr file if boot.scr file is included in the fat file system of SD card. A sample boot.cmd file is given in table 4.1. As seen from table 4.1 boot.scr file that is created from boot.cmd file boots the user specified file and in this thesis it is set as uImage file of the OS. uImage file is a compressed kernel file of the OS. uImage makes the necessary configurations on the platform for the rootfs file system created by Narcissus online image builder. It is a compressed image of the

file system of OS. The last step after porting the MLO, u-boot.bin, boot.scr and uImage files to 80MB FAT partition of SD card is porting the kernel file system to EXT2/3 partition of SD card. This procedure can be last a bit long according to size

Table 4.1. Sample boot.cmd Files Commands

	COMMAND	AIM
1	setenv bootdelay 1	Set start-up delay
2	setenv loadaddr 0x82000000	Set loadaddr parameter to determine the loading start address of uImage kernel
3	setenv dvimode 'hd720'	Set dvimode parameter as 1280x720 resolution
4	setenv vram '12M'	Set vram parameter to determine how much of the RAM memory will be used for display graphic buffers
5	setenv mem 'mem=55M@0x80000000 mem=384M@0x88000000'	Set mem parameter to determine how much of the RAM memory will be used by OS. Unset memory shared by ARM and DSP for DSPLink and cmem part.
6	setenv defaultdisplay 'dvi'	Set defaultdisplay parameter as dvi output
7	setenv loadbootscript 'fatload mmc 0 ${loadaddr} boot.scr'	Set loadbootscript command to boot the boot.scr file
8	setenv bootscript 'echo Running bootscript from mmc ...; source ${loadaddr}'	Set bootscript command
9	setenv loaduimage 'fatload mmc 0 ${loadaddr} uImage'	Set loaduimage command to load the uImage file to the RAM memory
10	setenv mmcroot '/dev/mmcblk0p2 rw'	Set mmcroot parameter to open the rootfs file system in the second partition of SD card
11	setenv mmcrootfstype 'ext3 rootwait'	Set mmcrootfstype parameter to determine the rootfs file system partition type as ext3
12	setenv mmcargs 'setenv bootargs console=${console} vram=${vram} omapfb.mode=dvi:${dvimode} omapdss.def_disp=${defaultdisplay} root=${mmcroot} rootfstype=${mmcrootfstype} ${mem}'	Set mmcargs command with predefined parameters for mmcboot
13	setenv mmcboot 'echo Booting from mmc ...; run mmcargs; bootm ${loadaddr}'	Set mmcboot command to boot uImage file from the RAM with user defined parameters.
14	setenv bootcmd 'if mmc init; then if run loadbootscript; then run bootscript; else if run loaduimage; then run mmcboot; fi;fi;'	Set bootcmd command to boot from SD card. If boot.scr file included in SD card, system boots boot.scr file. If boot.scr file not included in SD card systems boot uImage file
15	run loaduimage	Execute the loaduimage command
16	run mmcboot	Execute the mmcboot command

of file system. After this last step SD card is ready to start up the Angstrom Linux OS on ARM core of DM3730 processor. Now we can put the SD card on Beagleboard-XM platform a power up it. After power up, the booting code in the ROM of the platform loads MLO file into boots the MLO file. MLO file boots the u-boot.bin file. U-boot.bin file makes the peripheral configuration of platform and run the boot.scr script. Boot.scr script ports the user defined kernel parameters into uImage uncompressed kernel file and boot uImage file. At the last step of uImage boot procedure, the init process of file system loaded on the EXT2/3 part starts. Init is responsible for starting system processes as defined in the `/etc/inittab` file. Init process starts the applications defined by user and kernel at start up. Also at shutdown procedure, init controls the sequence and processes for shutdown. The init process is never shut down. It is a user process and not a kernel system process although it does run as root. Below table shows Angstrom init process levels.

Table 4.2. Angstrom Init Process Levels

Process ID	Description
RC0.d	The Scheduler
RC1.d	The init process
RC2.d	Kflushd
RC3.d	Kupdate
RC4.d	Kpiod
RC5.d	Kswapd
RC6.d	Mdrecoveryd

To initialize an application at start-up some configuration must be done in init.d and RCx.d files in the rootfs file system. At the end of these steps OS of ARM core works according to configuration done by user. The details of configuration procedures are given in system implementation chapter of thesis.

4.2. Setup of OS for DSP Core

The Texas Instruments DSP/BIOS is a real-time multi-tasking kernel that was designed to run on DSP platforms, specifically the TMS320C6000, TMS320C5000, and TMS320C28x families. It does not require any license fee and it is available both standalone and integrated in the Code Composer Studio Integrated Development Environment (IDE) tool. DSP/BIOS offer many functions in order to support complex applications within the constraints and deadlines typical of real-time applications. It aims to achieve a minimal memory footprint by using configurable modules that can be excluded by the kernel if not used.

DSP/BIOS offer fundamental Inter Processor Communication (IPC) mechanisms for synchronization and communication among threads. DSP/BIOS offer other two IPC mechanisms: mailboxes (MBX) and message queues (MSGQ). A mailbox is a synchronous way to exchange messages among tasks on the same processor (the DSP in our case). The exchange is synchronous implying that both the sender and the receiver must be ready to send/receive the message. In this sense, a mailbox is both synchronization and a communication tool. In DSP/BIOS RTOS, exchanged messages must be fixed-size. The programmer can set the fixed size of these messages taking into account the limited memory that is available. If mailboxes are expected to exchange messages among threads located on multiple cores, then a message queue transport module is needed. Such a module is not available in the kernel, but it is provided by the DSPLink. Message queues are used for asynchronous communications: the message is stored in a memory location and is retrieved by the receiver as soon as it is ready. There is no need for both actors to be ready for the communication. The message queue can handle variable-length messages between tasks.

DSP/BIOS provide a set of services for interrupt management aimed to maximize flexibility, while reducing data memory requirements. For power management, DSP/BIOS focus on minimizing power consumption while still meeting performance constraints. To achieve this, a range of power management features is provided. Some of them are:

• idling the CPU when the idle cycle is running (i.e., suspending CPU operation when there is no computation to be done);
• providing Application Programming Interfaces (APIs) for voltage and frequency scaling;
• providing standby and hibernation APIs (to reduce power consumption for longer periods of time);
• automatic idling of the inactive peripherals; and
• coordination of complex systems power management by means of tracing and notify mechanisms.

In this thesis, DVSDK 4.01 package infrastructure is used to load DSP/BIOS to the DSP core of processor. Version of DSP/BIOS used in thesis project is 5.41 and it is compiled and linked for DSP core by using the make files inside DVSDK 4.01 package. DVSDK 4.01 package is a development package with source codes written for subsystems of TI DM3730 SoC processor like DSP, SGX graphic accelerator, DMA, ARM and NEON subsystems. It is developed for DM3730 EVM board by TI engineers and it can be downloaded freely from TI website. DVSDK uses ARAGO OS and this OS does not include necessary driver packages to start up the Beagleboard-xM board. However, it is possible to create a working development package for Beagleboard-xM board by combining Angström OS created by using Narcissus, DVSDK 4.01 package files with some configuration file changes and some patches from open sources. The details of the procedures to create a working development package for Beagleboard-xM board are given in system implementation chapter of thesis. Figure 4.3 shows the porting the DSP/BIOS to DSP core. DSP/BIOS are compiled from the subfolders of DVSDK. Therefore first step to port the DSP/BIOS to DSP core is modifying the DVSDK according to Beagleboard-XM board. Modifying steps is described in part 4.4.3.1. DSP/BIOS is placed on codec server and is loaded on DSP by ARM core. Therefore the second step is adding necessary drivers to Angstrom kernel to enable the communication of ARM core with DSP core over shared RAM memory. These drivers are Contiguous Memory Management (CMEM), Local Power Manager (LPM), System DMA (SDMA) and DSPLink drivers. Duty of these drivers is given in part 4.3.2. After adding necessary drivers to kernel the next step is compiling DSP server from C6ACCEL and

CODECS folder of DVSDK. This can be done by running “make c6accell” and “make codecs” commands on console under DVSDK folder. Both of these commands create a DSP server that ends with “.x64P”. This file includes all the files related with DSP like DSP/BIOS, DSP Applications, DSP EDMA driver etc. C6ACCEL includes the applications of MATHLIB, IMGLIB and DSPLIB libraries. However it does not include the source files of these libraries. A user DSP application can be added to these libraries and can be called from ARM core. Codecs include the C6ACCEL outputs and add these outputs TI’s encoders and decoders.

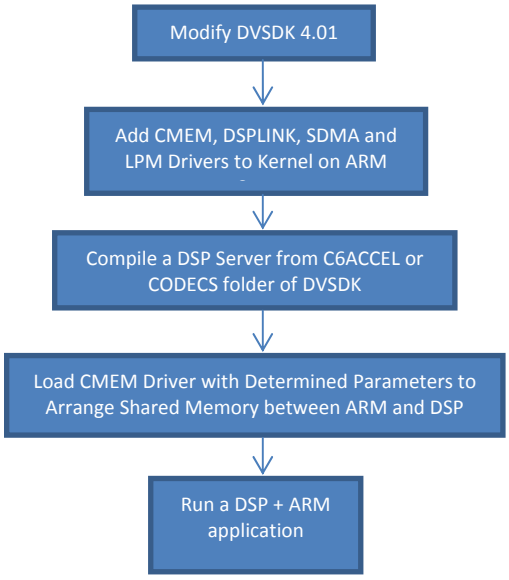


Figure 4.3 Porting DSP/BIOS to DSP Core

The next step after compiling a DSP server is installing CMEM driver to kernel with the parameters that are defined in “.tcf” file of DSP server. A sample “server.tcf” file is given in Appendix H. This file describes to DSP core how to use shared ram memory. So ARM core must also partition the shared RAM according to these parameters by CMEM driver. Partitioning step is described in part 4.3.2. As a last step we can run an ARM + DSP application and the ARM core application loads the DSP server packages including also DSP/BIOS on DSP core.

4.3. Integration of Necessary Drivers

Angstrom OS implemented on the ARM core of processor is an embedded system with a basic package list. Some additional packages are also necessary to design a smart camera on this embedded system. Firstly, video capture and video display drivers must be added to OS to capture frames from USB webcam and to display the captured frames on DVI or svideo output of the board. Secondly the Interprocessor communication drivers must be added to OS to send the captured frames by ARM core to DSP core for image processing and send the processed frames back to ARM for displaying and streaming. Thirdly wired or wireless Ethernet drivers must be added to OS to stream the captured and processed frames through the internet.

4.3.1 Video Capture and Video Display Drivers

The first important driver to design a smart camera on an embedded system is video capture driver. There are different ways to capture frames from a camera by using Beagleboard-xM board. One of them is capturing frames from the digital YUV output of CMOS sensors by using camera input port of the board. This is a cost effective solution to design smart camera system. However, in this thesis a YUV pixel format USB webcam is used to design a simple smart camera and USB inputs of the board are used for camera input. To capture frames from USB webcam, Video for Linux Two (V4L2) drivers are used. V4L2 is a standard Linux video driver used in many Linux systems. It supports the video input and output ports of the OS. In this thesis project only video input drivers are used. Figure 4.2 shows the queue structure of V4L2 driver. Outgoing queue of the driver is used for input operations and incoming queue of driver is used for output operations. V4L2 driver capture frames from the devices connected to the video ports of the OS. These devices can be seen under the /dev file of file system. Captured frame buffers are saved under the V4L2 driver memory in a queue and these buffers can be own by an application. In this thesis work OpenCv capture commands and DMA hardware of the DM3730 processor are used to own captured frame buffers from the V4L2 frame buffer queue. To capture frames from V4L2 driver with DMA hardware some modifications are

made on the capture files that are present inside the DVSDK/DMAI/ti/sdo/packages folder. The details about these modifications are given in part 4.4.3.1.

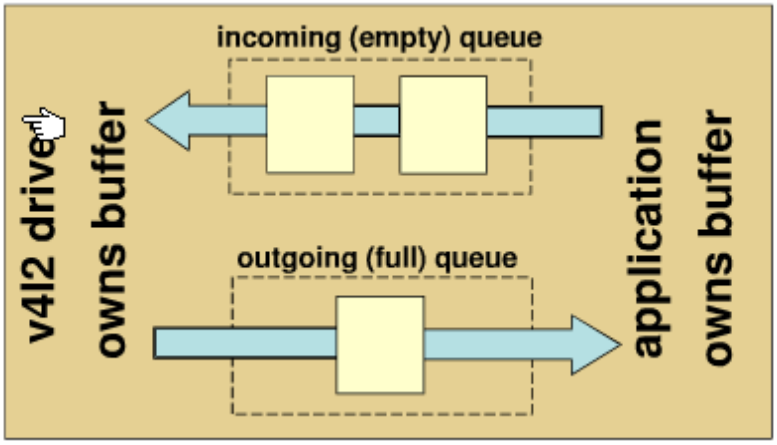


Figure 4.4 V4L2 Driver Queue Structure

V4L2 driver packages are included in the package list of Angström distribution and these drivers can be installed from console of OS by using “opkg install” package installation commands of Angstrom. Furthermore, different webcam drivers not included in V4L2 driver libraries can be search in the package list of Angström distribution by using package search commands and can be installed to kernel by using package install commands . After successful installation of necessary drivers, frames from webcam can be captured from the /dev/video port of the OS.

The second important driver to design a smart camera with an analog output on an embedded system is the video display driver. DM3730 processor supports two video display driver and one overlay driver. Display drivers are Video for Linux Two (V4L2) and OMAP Frame Buffer Device (OMAPFBdev) display drivers and overlay driver is OMAP Display Sub-System (OMAPDSS2) overlay driver. Figure 4.5 shows the OMAP Display Sub-System. As can be seen from figure OMAPDSS2 achieves the pipeline connections between display drivers and display devices and manage the overlay operations. Furthermore, V4L2 display drivers is used for video

streaming operations whereas OMAPFBdev drivers are used for graphic operations. However V4L2 display driver ,namely "omap_vout.ko" driver can not be installed to kernel taken from Narcissus image builder. Therefore OMAPFBdev video display driver and OMAPDSS2 overlay driver are used to display the captured and processed images at the outputs of the board. OMAPFBdev driver is a standard

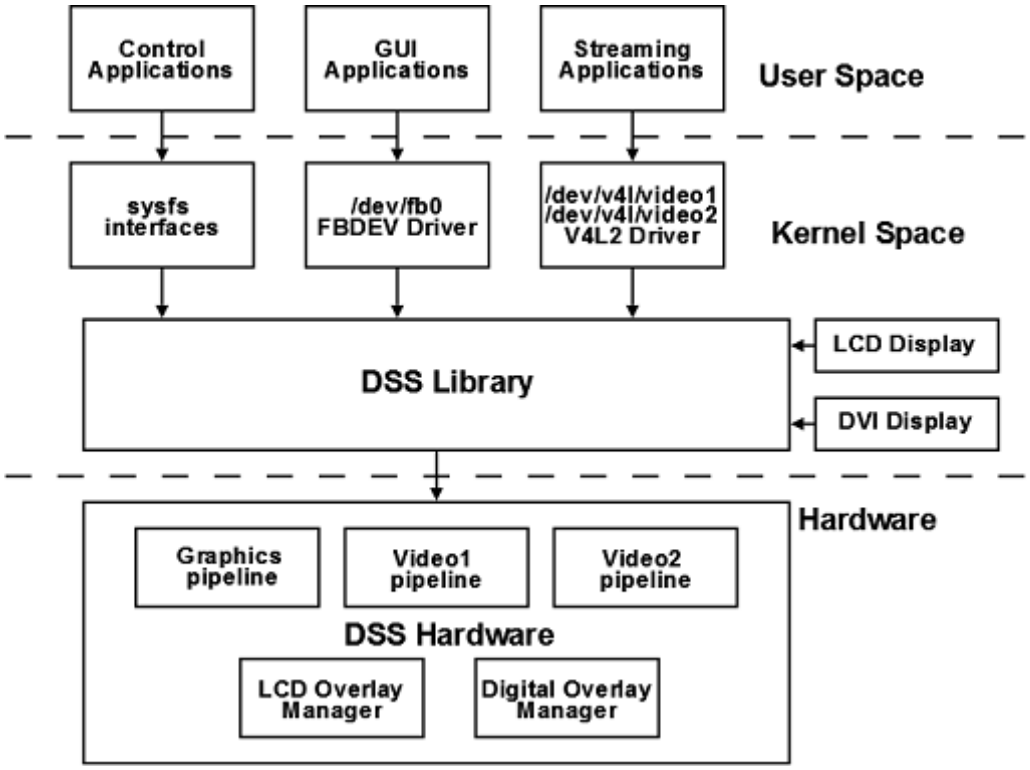


Figure 4.5 OMAP Display Sub-systems (OMAPDSS2)

Linux video driver used in many Linux systems. It maps the frame buffer of a display device into user space. Frame buffers can be seen under the /dev file as /dev/fb/x. Memory usage and the number of frame buffers can be configured by setting u-boot parameters at start up. In this thesis work three frame buffers are used with 8MB, 4MB and 4MB sizes. The Beagleboard-xM platform has two different video outputs, namely DVI and svideo outputs. In this thesis DVI output of the camera is used. Mapping the display frame buffers to different overlays on DVI

output is achieved by using configuration files of OMAPDSS2 overlay driver. Overlays can be mapped to different display frame buffers of display drivers. Figure 4.6 shows a sample OMAP (DSS) overlay implementation. In this example we have four overlays. Background is on the bottom level overlay and graphics are on the second level overlay. Third level overlay is mapped to video1 frame buffers and the top level overlay is mapped to video2 frame buffers. In this thesis kernel GNOME window manager buffers are mapped to fb0 frame buffer of OMAPFBdev driver and

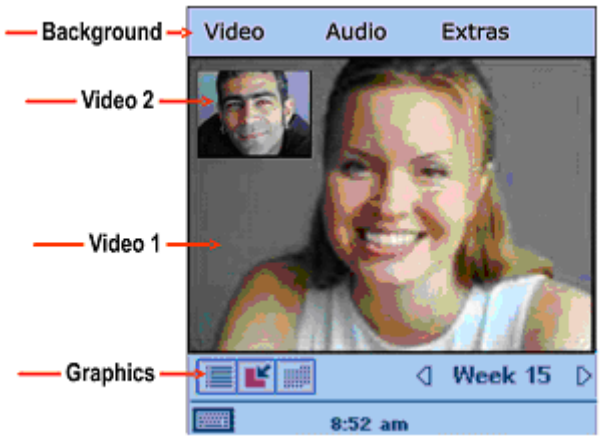


Figure 4.6 OMAP (DSS) Overlay Examples

fb0 driver is mapped to overlay0 of OMAPDSS2 driver and overlay0 is mapped to display0 (DVI output) of the OMAPDSS2 driver. The captured and processed images buffers by the DMA hardware of the DM3730 processor are mapped to fb1 frame buffer of OMAPFBdev driver and fb1 driver is mapped to overlay1 of OMAPDSS2 driver and overlay1 is mapped to display0 (DVI output) of the OMAPDSS2 driver. By using OMAPDSS2 driver different overlays with different sizes and transparency can be used on display. This enables to user design very attractive display outputs. Also, a user interface to control the camera settings can be achieved easily by designing a transparent OSD on different overlays of the OMAPDSS driver.

4.3.2 Inter-processor Communication drivers

The third important driver of the design step is Inter-processor Communication (IPC) drivers that enable the communication between ARM and DSP cores. IPC driver is the DSP/BIOS Link (DSPLink) driver. Beside the DSPLink driver, Contiguous Memory Management (CMEM), System DMA (SDMA), Local Power Manager (LPM) and Enhance DMA (EDMA) drivers are also necessary drivers for IPC mechanism.

DSP/BIOS Link is fundamental software for the IPC between the ARM and the DSP [32, page 9]. Today many DSP-based applications use ARM to control one or a set of DSPs. The most common operations between the two cores are:

• exchanging control and data information;
• booting the DSP by ARM; and
• control and coordination of algorithms and tasks running on the DSP by ARM.

DSPLink is software designed to facilitate such interactions. It offers programmers a set of APIs that abstract the characteristics of the physical communication layer. The programmers' attention can focus on the development of the application algorithms, rather than on the communication mechanisms. DSPLink is designed to work with DSP/BIOS OS on the DSP side, while no specific OS is required on the ARM side. Since the DSPLink package contains all the source code, it is possible to port it to a variety of operating systems (i.e., all of those supported by the ARM).

On both sides, link drivers hide the physical link layer. On the ARM side, the processor manager exposes the DSPLink APIs to the client, while the OS adaptation layer makes the DSPLink components independent from the specifics of the OS.

DSPLink offers the following services to the end-points clients [39, page 9]:
• basic processor control (via the PROC module);
• sharing and synchronization of a memory pool among the cores (POOL);
• notification of user events (NOTIFY);

• mutually exclusive access to shared resources (MPCS);
• linked list data streaming (MPLIST);
• data transfer over channels (CHNL);
• exchanging of messages (MSGQ);
• circular buffer data streaming (RINGIO) and
• zero-copy messaging.

The list above represents the complete set of services offered by DSPLink. However, depending on the platform and OS used, the system may support only a subset of these services. The last entry in the list refers to the zero-copy transfer mode (ZCPY). Using this technique the data are exchanged among the cores by means of shared memory, avoiding any need to physically copy data to and from the respective memory spaces. In this way data transfer implies only the allocation of buffers in the shared memory region and addresses translation. The ZCPY system is composed of three sub-components:

• Shared memory allocator: the data to be exchanged are stored in buffers allocated in the shared memory;

• Address translator: the buffers allocated in the shared memory need to be accessed by any core as user-space buffers. Since the DSP sub-system does not have a MMU, this component must perform address translations from the DSP physical address space to ARM virtual space and vice versa. Furthermore, the translation from ARM user and kernel spaces are performed by this component; and

• Shared Memory Inter Processor Signaling: this component is in charge of managing control structures and to inform the processors about changes in the shared buffers.

DSPLink provides several advantages for programmers working with an embedded multicore system:
• Portability: an application that uses DSPLink is easily portable to architecture with no or few changes since architectural low level details are hidden;

• Flexibility: applications are more flexible since programmers can use the most appropriate high or low level communication protocols depending on their necessities;

• Scalability: only needed modules need to be compiled and included in the executable file, saving resources; and

• Physical layer abstraction: the details of the physical link are hidden from the programmer, so that it is easier to focus on application development.

The Contiguous Memory Management driver provides the ability for user-mode applications to allocate and free blocks of physically contiguous memory. This is done to manage multimedia data buffers which will be worked on by DSP multimedia algorithms. The reason DSP multimedia algorithms require physically contiguous memory is that DSP core not contain an MMU and DMA that is used for Linux-based algorithms don't access memory through an MMU. Linux is different from an RTOS like DSP/BIOS in that it manages all resources in the system for the application. The application requests access to a resource and Linux grants it depending on UNIX permissions and availability. This means that all the memory you give to Linux will be "owned" by Linux and is out of your direct control. Furthermore, you do not know where in physical memory this requested memory is allocated and whether it is physically contiguous or not (the MMU makes the memory look virtually contiguous to the process). This is normally a great feature, but it becomes a problem when you want to share a buffer between the ARM and the DSP. This because the DSP needs physically contiguous memory to work with. This is the reason why the CMEM kernel module was created, i.e. to provide physically contiguous buffers to be shared between the ARM and the DSP. This is also useful for buffers which are to be accessed using the DMA.

The System DMA driver provides the ability for user-mode applications to request SDMA channels and operate on a channel using direct, memory-mapped access to the channel's DMA registers. It also provides "blocking" support for waiting for the completion of a transfer on a particular channel.

Enhanced DMA is incorporated on the Davinci family of C64+ devices. The EDMA driver provides the same operations for DSP core.

In this thesis work Texas Instruments DSPLink, CMEM, SDMA and LPM drivers are compiled from the DVSDK 4.01 package and changed with Angström kernel module drivers. EDMA driver that is used by DSP/BIOS is included in codec server compiled from DVSDK 4.01 package.

4.3.4 Ethernet Drivers

The other important driver to design an IP output smart camera is wired or wireless Ethernet drivers. These drivers enable the system to stream the captured and processed frames through the internet and to file share between user and camera. Wired Ethernet driver on board is provided by Angstrom OS as a standard. For wireless Ethernet specification, a wireless USB Ethernet must be added to the system with a supplied driver. In this thesis work Belkin N150 USB Ethernet with 150Mps speed is used for wireless connection [50]. Below equation shows the necessary connection speed to stream a YUYV format VGA resolution image at 30 fps.

```
Bytes per Pixel = 2 (YUYV format)
Connection Speed = Bytes per Pixel x 8 x resolution x fps
                 = 140.6 Mbps
```

Streaming data rate can be decreased by encoding. So connection data rates of Belkin N150 USB Ethernet (150Mbps) and USB 2.0 (400Mbps) will be sufficient for video streaming over USB wireless Ethernet. Belkin N150 USB Ethernet driver is used from Ubuntu kernel and wireless connection is achieved with Ubuntu OS. However, its driver is not found in the open source package list of the Angström distribution. Therefore wireless Ethernet connection is not used in thesis work.

4.4 System Implementation

As we explained before Beagleboard-XM platform has a SoC DM3730 processor that includes two cores, namely ARM and DSP and several hardware accelerators. In this thesis, the objective is using DSP core of the processor in parallel with ARM core for some image processing algorithms to see the contribution of DSP on system performance. The most effective way of that using DMA hardware accelerator of the processor for data transfers between ARM, DSP and shared RAM memory. In this part of thesis we explain how to develop and port algorithms onto Beagleboard-XM platform to use ARM and DSP cores separately and together with the aid of NEON co-processor and DMA hardware accelerator. In order to quantify the system's performance and to be able to perform an empirical analysis, the development environment and test algorithms are explained in this part of thesis. The function of the test algorithms is to evaluate the video capturing, displaying and processing capabilities of board. Figure 4.7 shows the necessary setup for the implementation of test algorithms.

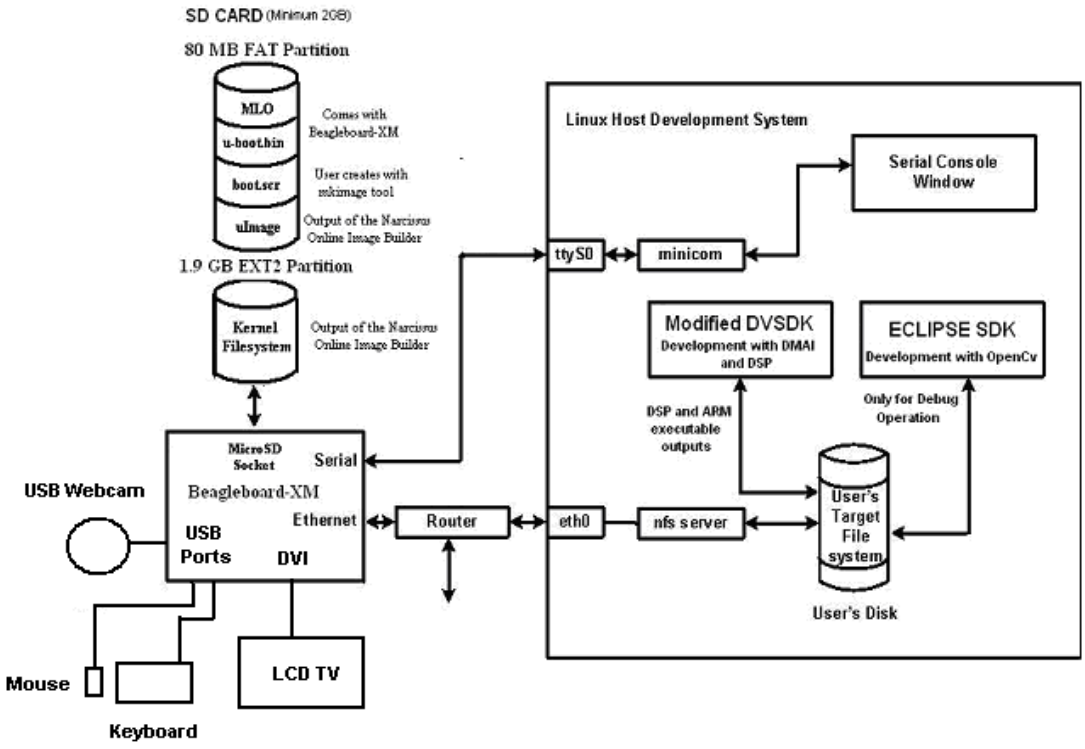


Figure 4.7 Necessary Setup for Implementation of Test Algorithms

Up to know we have described the how to port the Angstrom OS to ARM core by partitioning SD card and installing necessary files on it. Furthermore, we have explained how to install the necessary drivers for implementation of a smart camera on Angstrom OS. Now we can start-up and use the Angstrom OS on beagleboard-XM and we capture frames from USB webcam and display them on a LCD TV by using Angstrom kernel and added drivers. At that point we also need a mouse and keyboard to control the Angstrom OS. For ARM core applications it is possible to develop image processing algorithms on board by using a C compiler. However, in this work Eclipse SDK with added opencv libraries is used at the development stage of ARM core application. For development of DSP core application, DVSDK 4.01 is modified according to Beagleboard-XM and used for development of DMA and DSP based applications. A serial connection is made between Host PC and Beagleboard-XM to control the Beagleboard-XM platform from HOST PC. This serial connection is also used to display and change the u-boot parameters of Beagleboard-XM platform at start up. Furthermore, a Network File System (NFS) is established between Host PC and Beagleboard-XM to share the outputs of the Eclipse and DVSDK software development environments with target board Beagleboard-XM.

4.4.1. Configuration of Host PC and Target Board

In this thesis work, to develop and compile the algorithms for experiments, VMware has been used for create a virtual machine. VM is created with 512 MB ram memory on 2.80 GHz Intel CPU. As DVSDK 4.01 setup needs the Ubuntu Release 10.04 OS, this release is used as OS of virtual machine. Minicom serial communication application is used to communicate with target board by using the console of OS. Eclipse SDK was installed on host PC and CDT GNU Tool chain was added to Eclipse SDK as C++ development environment. And then GCC 4.4.3 package was installed host PC as a C/C++ compiler and OpenCv library packages were also installed on host PC to use for video capture and video display operations in algorithms. Furthermore a modified DVSDK 4.01 software development package was installed on host PC to develop and compile the algorithms that will be run on both ARM and DSP cores of DM3730 processor. Finally necessary NFS packages

were installed on host PC to set the host PC as NFS server to share the outputs of SDK's with target board.

4.4.2 Serial Communication setup

Serial communication is made by using a USB serial converter between serial communication port of Beagleboard-xM and USB port of host PC. Default serial communication port of Beagleboard-xM is set to ttyS0 port at 115200 baud rate. These parameters can be changed by adding desired parameters to `console=${console}` parameter of boot.cmd file given in table 4.1. In host PC, minicom application is used for communication. Minicom package is installed from the open source packages of Ubuntu community. Serial communication port of host PC is set to 115200 baud rate, ttyUSB0 port by console commands of minicom.

4.4.3 Configuration of Network File Sharing

File sharing is a necessity to transport the developed algorithms to target board easily. A flash memory can also be used to transport the developed algorithm files from host PC to target board. However this will take a lot of time and flash memory will be plugged in and plugged out several times to host PC and target board. Network file sharing (NFS) packages can be installed from Ubuntu and Angstrom distributions open source community. The necessary packages for NFS are "Portmapper" and "Dropbear-Ssh server". After installation of these packages, some configuration must be done on exportfs and fstab files under the /etc file. To set up host PC as server, exportfs file must be configured to share the files with target board. The IP number of the target board as client and location of the shared folders must be enter in exportfs file to make the necessary server configuration. To set up the target board as client, fstab file must be configured to mount the share files of host PC. The mount location of shared files and IP number of server must be enter in fstab file to make the necessary client configuration. The sample and exportfs file given in Appendix E

4.4.3 Algorithm Development Setup

In this thesis project, Eclipse Galileo development environment is used to developed algorithms that will be evaluated on ARM core and NEON co-processor. A C++ compiler and OpenCv libraries are attached to the Eclipse development environment. OpenCv libraries are used for capturing and displaying operations of algorithms. The outputs of Eclipse SDK can not be used on target board directly because it uses an i486 architecture support C/C++ compiler and this compiler does not support ARM core of the DM3730 processor. Therefore, the algorithms that are developed on host PC with Eclipse SDK must be recompiled with a cross compiler supporting ARM architecture. In this thesis instead of using a cross compiler, the C/C++ compiler of target board is used and developed sources files on host PC are recompiled on target board. To do this a GCC 4.4.3 C++ compiler and OpenCv library packages were also installed on target board to recompile sources files of Host PC on board. Figure 4.8 shows the implementation of an application on ARM core and table 4.3 gives the necessary components for implementation on ARM core.

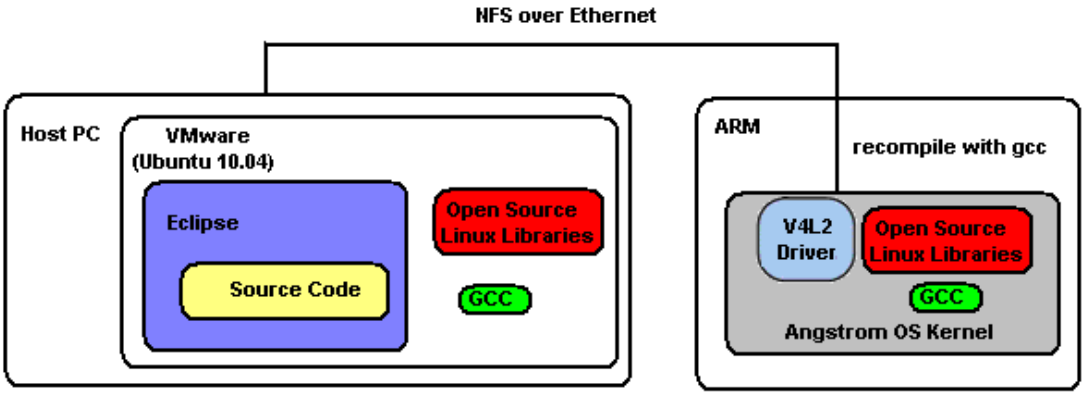


Figure 4.8 Implementation of an Application on ARM Core

Also to make easier the compile procedure of algorithms for ARM core and NEON co-processor “Makefile” files are created. Steps of compilation for algorithms in Eclipse are taken as reference to create a “Makefile” file. NEON co-processor is also

used for performance evaluation. The only thing must be done to use the NEON co-processor in applications is adding the below NEON co-processor parameters to the compiler.

Table 4.3. Necessary Components for Implementation on Arm Core

Component	Purpose in this application	Location
Eclipse	Development Environment	Host PC/VM/
Source Code	Source code of Application code	Host PC/VM/Eclipse
GCC	C compiler on target board and Host PC	Angstrom Filesystem/../gcc
Open Source Linux libraries	Provides libraries such as cv, cvaux, cxcore, highgui, ml etc	Angstrom Filesystem /usr/include/OpenCv etc
V4L2 Driver	Provides video capture drivers for the kernel	Angstrom Filesystem/../v4l2
Linux kernel Builder	The Linux kernel with device drivers	Outputs of Narcissus Online Image

- -march=armv7-a
- -mtune=cortex-a8
- -mfpu=neon -ftree-vectorize
- -mfloat-abi=softfp

A sample make file to compile the developed algorithms for ARM core or NEON co-processor is given in Appendix C. Make file is a script that includes the compile steps of a project that have more than one c source file. C source files of a project must be compiled by the compiler of OS in true order according to dependencies to each other. Make file links the output files of the source files to each other after compiling all source files and create a working output file. Make command packages must be installed into the kernel to use make file script. In this work, make command packages are installed to Angström kernel and NFS file sharing protocol is used to run the make file script that is created and saved on host PC. By using this method ARM core C compiler and libraries are used to compile the source codes of the algorithms developed on host PC. This method also enables one to see compiler errors that depend on target platform library. Make files commands are used by calling them from the console of OS. The second argument besides the make

command determines the which part is used in make file. For example “make neon” command given in Appendix C compiles the source code for NEON co-processor while “make arm” compiles the source code for only ARM core. Furthermore, using the source codes that are developed under Eclipse environment for ARM core enables us to compare the performances of ARM core of target board and Intel core of Host PC for same applications.

In this work, A modified DVSDK 4.01 software development environment is used to developed algorithms that will be evaluated on ARM core , DSP core and DMA hardware accelerator of DM3730 processor. The modification of DVSDK 4.01 is given in part 4.4.3.1. The outputs of DVSDK can be used directly on target board because of cross C/C++ compiler and C6run DSP compiler included in the DVSDK package. Figure 4.9 shows the implementation of an application on ARM+ DSP core and table 4.4 gives necessary components for implementation on ARM + DSP core.

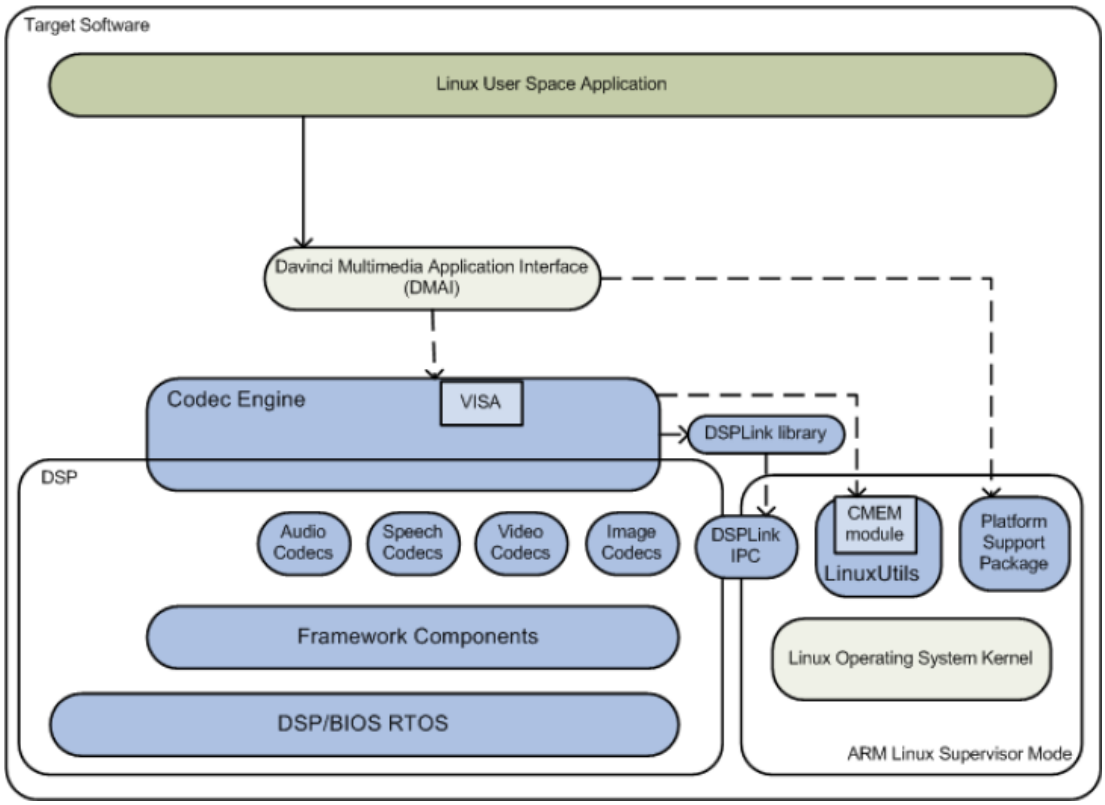


Figure 4.9 Implementation of an Application on ARM + DSP Core

In this implementation the most important component is Davinci Multimedia Application Interface (DMAI). DMAI uses the Direct Memory Access (DMA) hardware accelerator of the DM3730 processor. DMAI includes the necessary source files to configure and manage the Linux device drivers. It is a very quick data bridge between shared RAM and DSP Codec Engine and Linux kernel drivers on ARM core. DMAI is a thin utility layer on top of the OS (Linux or DSP/BIOS) and the Codec Engine (CE) to assist in quickly writing portable applications on a ARM + DSP SoC platform. The benefits of using DMAI include:

Table 4.4. Necessary Components for Implementation on ARM + DSP Core

Component	**Purpose in this application**	**Location in the DVSDK**
Platform Support Package	Provides device drivers for the kernel and documentation and examples to support them .	DVSDK/psp
Codec Engine	Cross platform framework for the Applications invoking multimedia codecs and other algorithms.	codec_engine_xx_xx_xx_xx
Framework Components	Cross platform framework for Servicing resources to algorithms.	framework_components_xx_xx_xx_xx
LinuxUtils	Linux specific utilities for Framework Components assisting with resource allocation of DMA channels (EDMA module), physically contiguous Memory (CMEM module) and allows the codecs to receive completion interrupts of various coprocessor resources (IRQ module) .	linuxutils_xx_xx_xx_xx
Davinci Multimedia Application Interface	Multimedia application utility layer	dmai_xx_xx_xx_xx
Multimedia Codecs	Compression and decompression of multimedia data	codecs_<platform>_xx_xx_xx_xx
RTSC (XDC)	Tool used to configure Codec Engine, Framework Components and multimedia Codecs for your application.	xdctools_xx_xx_xx_xx
XDAIS	TI Algorithm Interface Standard used for algorithm standardization which is used by various other components including Codec Engine	xdais_x_xx_xx_xx
DSPLINK	GPP to DSP processor communication link for passing messages and data in	dsplink_x_xx_xx_xx

- DMAI enables the XDAIS Algorithm Standard compatibility of the implemented application for DMA resources.

- DMAI wraps the Linux device drivers in a user defined application, shielding you from the rapid progress of the Linux kernel, increasing your portability.

DMAI does not wrap the OS or CE, but as the below picture shows the application can choose when to use DMAI and when to use the OS or Codec Engine directly:

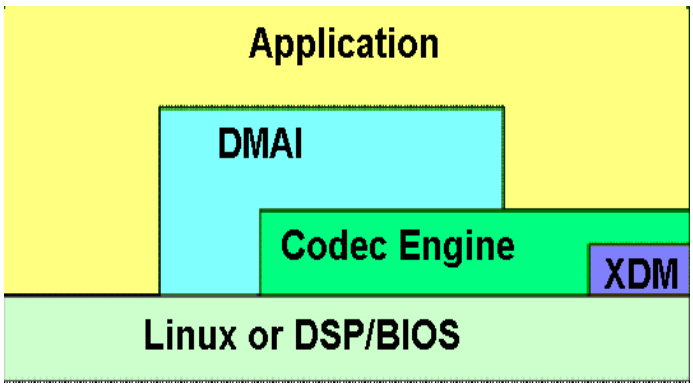


Figure 4.10 The Place of DMAI

DMAI is a functional design, meaning that the modules often describe a certain operation (frame capture using Capture, frame display using Display, color conversion using Ccv etc.), but the module implementation may change between devices and operating systems depending on which peripheral device drivers and other local Application Programming Interfaces (APIs) are available. In other words, DMAI does not abstract the peripherals themselves, it abstracts the actual operations e.g. frame capturing. DMAI then implements a frame capture using the peripherals and resources at its disposal on a particular platform. DMAI is a collection of modules, and the application can pick and choose which modules to use. Since DMAI comes with source code, it can also be used as a reference on how to accomplish certain tasks using e.g. a certain device driver. The different DMAI modules communicate using a Buffer abstraction which carries not only the actual (video, speech, audio etc.) data but also meta data describing the Buffer which is used by the Codec Engine and Linux device drivers to perform operations on the data.

DMAI is used by the DVSDK demos and makes additional demo development quick. In this work edge detection demo infrastructure of DVSDK 4.01 is used as reference application code to develop algorithms on both ARM and DSP cores of processor. DMAI module of DVSDK 4.01 is designed according to Linux drivers of DM3730 EVM board peripherals. DM3730 EVM board uses CVBS, svideo and component video inputs to capture frames and USB webcam input is not supported by DMAI. Furthermore, kernel of DM3730 EVM includes the V4L2 display drivers and can display YUV format frames on display. Because of different capture driver properties of USB camera and different pixel format of OMAPFBdev display driver, DVSDK demos can not be run on Beagleboard-XM platform. To run these demo applications on Beagleboard-XM DMAI capture and display functions must configured according to drivers properties of USB webcam and pixel format of OMAP FBDEV display driver.

4.4.3.1 Modification of DVSDK 4.01 for Beagleboard-XM

DVSDK 4.01 package is a development environment with a lot of open sources codes. These open source codes are good examples to start to develop algorithms by using different coprocessors and subsystems of the DM3730 processor. However, DVSDK 4.01 package officially supports only DM3730 EVM board and the configuration files of the development environment are set for the peripherals of the DM3730 EVM board. Therefore, these configuration files must be set according to Beagleboard-xM peripherals to use DVSDK 4.01 package together with Beagleboard-xM platform. After this setting, DVSDK sample examples can be used to develop specific algorithms that run on both ARM and DSP cores of the processor. The DVSDK 4.01 package contains many software components. Figure 4.11 shows the subfolders of the DVSDK 4.01 package. These folder includes the source files of software components. In this thesis the source files of necessary software components are investigated and configured according to Beagleboard-XM platform. The duty of necessary software components are given in table 4.4.

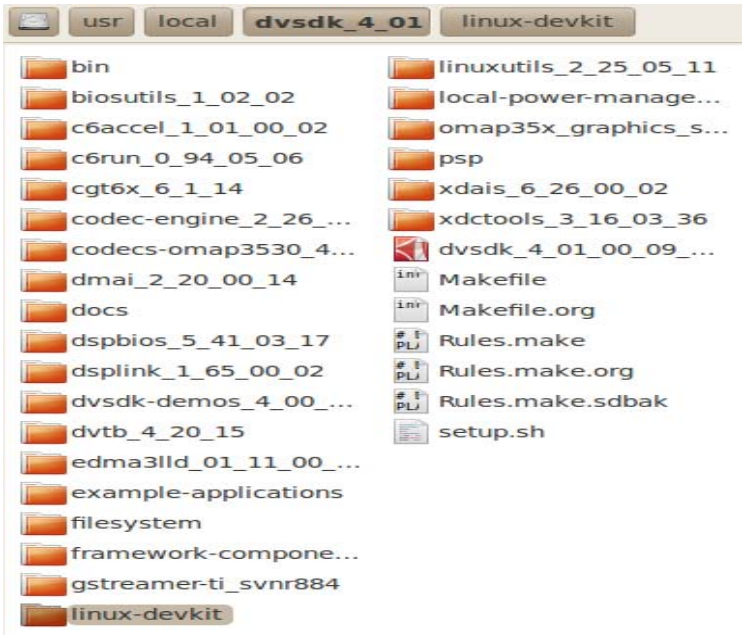


Figure 4.11 Subfolders of DVSDK 4.01

Most of the software components of the DVSDK 4.01 is related with the structure of the DM3730 SoC processor and does not change so mush according to peripherals of platforms based on DM3730 processor. Here the most important and platform specific software component is DMAI software component. We will describe how to change this component in more detail in the following sections of this part.

Figure 4.12 shows the localization of software components inside the DVSDK 4.01 for an application implementation. Software components that are developed by Texas Instruments are shown as blue where some that are developed by the open source community are shown as grey. In this thesis all the software components except 3D openGL, Qt/embedded, GStreamer-ti-plugin and multimedia codecs (video, audio, speech and image) are used for ARM + DSP application implementation on Beagleboard-XM platform. However GStreamer-ti-plugin and multimedia codecs components can be used to improve the encoding decoding capabilities of implemented smart camera application on board. Qt/Embedded component can be used to design attractive and user friendly camera user interfaces.

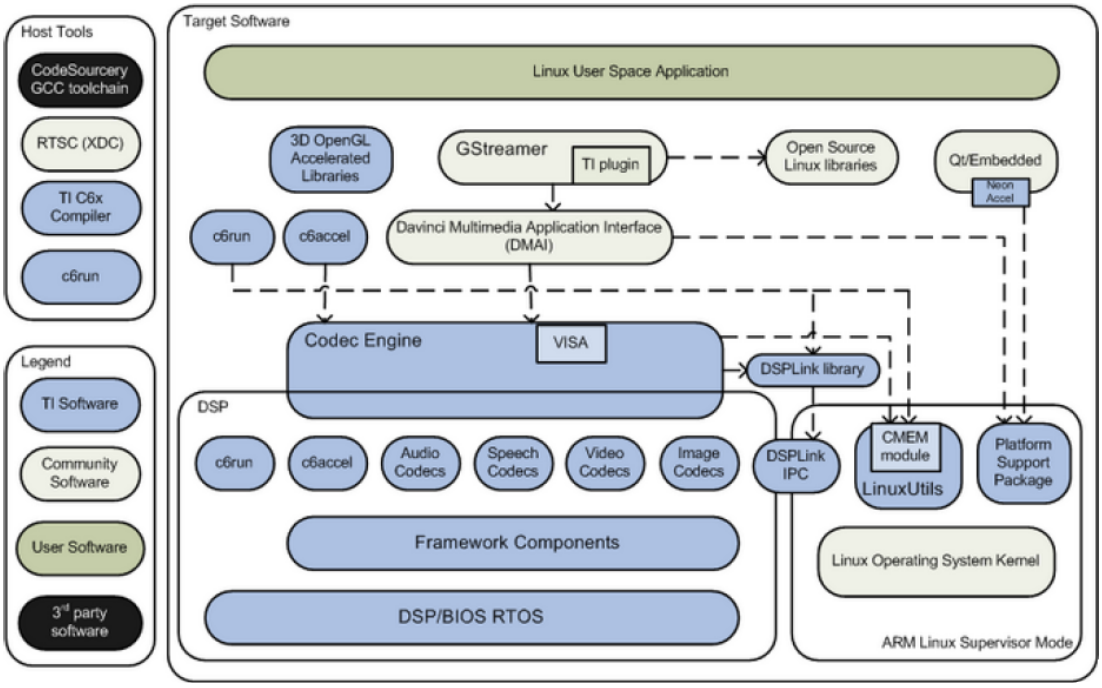


Figure 4.12 Localization of Components Inside the DVSDK 4.01 for an Application Implementation

User can connect these user interfaces over the network to control the parameters of smart camera or update the software of smart camera. 3D openGL accelerated libraries may be used to display the 3D view of objects that are monitored by stereo cameras connected to platform.

Necessary files to learn about how to install and use the DVSDK 4.01 is included in "docs" folder. Also each folder of software component includes a user manual about software components. "bin" folder includes the scripts to setup the serial and network connections between target and Host PC and to prepare the SD card with ARAGO OS of TI with the outputs of software component and necessary drivers. However using these scripts is not possible for Beagleboard-XM platform because of different configuration of it from DM3730 EVM. So configuration parameters of files must be modified according to Beagleboard-XM platform. Configuration parameters generally included in rules file of Makefile that compile the software components. Most of the configuration files of DVSDK 4.01 can be modified by some patch that

can be found in open source community. Appendix-F gives the steps to port DVSDK 4.01 onto Beagleboard-xM rev B/C by using patches defined by sourceforge.net. By following these steps we can start to use the DVSDK 4.01 for Beagleboard-XM. However none of the multimedia application that require a camera input does not work again. Because Beagleboard-XM platform does not include a camera. In this thesis A4tech USB webcam is selected and added to Beagleboard-XM board as application specific device and this configuration change is not defined in modification patch. So this modification also must be done by us.

To start up the multimedia applications on Beagleboard-XM platform we must struggle with the necessary files of DMAI component that control the V4L2 capture drivers and OMAPFBdev display drivers. It is possible to capture frames from V4L2 drivers by using DMAI hardware of the DM3730 processor. However, to make the connections between V4L2 driver and DMAI we must configure the some request parameters in capture and videobuf files. Figure 4.13, 4.14 and 4.15 shows the place of capture and videobuf files of DMAI.

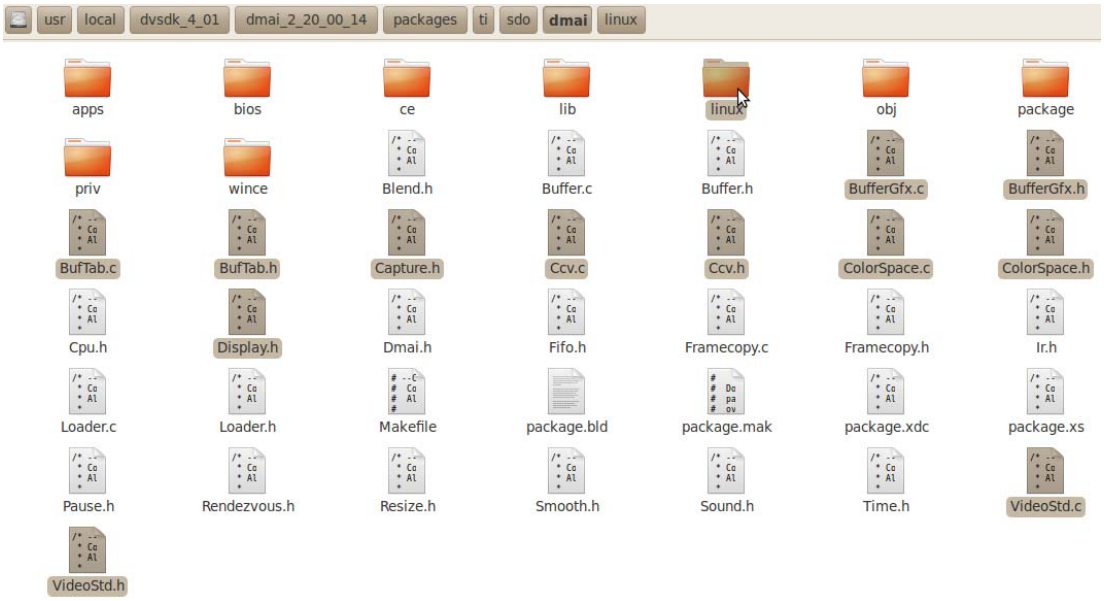


Figure 4.13 Files under of DMAI folder

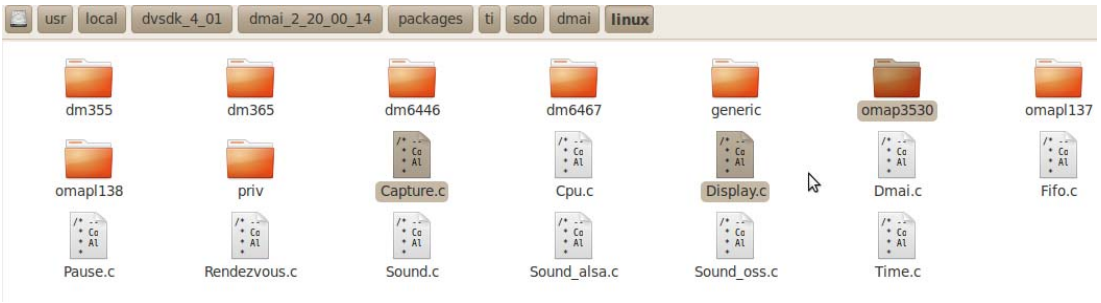


Figure 4.14 Files under of DMAI/LINUX folder

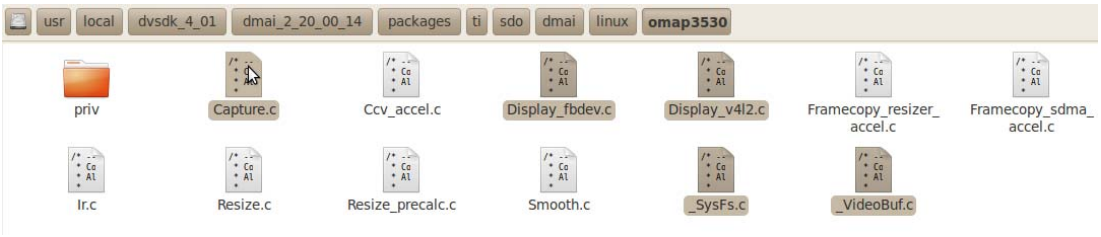


Figure 4.15 Files under of DMAI/LINUX/OMAP3530 folder

Capture.h file is shown in figure 4.13 defines the functions related with capture drivers. So it does not change for our application because we do not need to define a new function. The files in figure 4.13 are general functions and do not depends on the peripherals of platform. So in this files there is no need a change for capture and display operation. These files include some special operations like frame copying ,resizing, color space converting etc. Ccv.c files includes some the color space conversion operations. In our application we need to convert the YUYV pixel format captured frames to ARGB pixel format DVI display output frames. Because of absent color conversion from YUYV to ARGB we add a YUYV to ARGB color conversion function to Ccv.c file in figure 4.13 to achieve the color conversion in ARM core.

Capture.c file is shown in figure 4.14 defines the capture input configurations of different platforms. In modified DVSDK 4.01 our platform is defined as OMAP3530. So default input and output parameters are modified according to USB

webcam used in this work. Default modified capture parameters are given below. Here because of undefined USB input, COMPOSITE input is defined as input. Capture device is mapped to /dev/video0. Because of VGA resolution of camera video standard defined as VGA. Because of undefined YUYV color space, color space is defined as interlaced YUV format UYVY.

```
const Capture_Attrs Capture_Attrs_OMAP3530_DEFAULT = {
   3,
   Capture_Input_COMPOSITE,
   -1,
   -1,
   -1,
   -1,
   "/dev/video0",
   FALSE,
   VideoStd_VGA,
   -1,
   ColorSpace_UYVY,
   NULL,
   FALSE,
};
```

Display.c file is shown in figure 4.14 defines the display video and OSD output configurations of different platforms. Default modified display parameters are given below. Here, display standard is defined as FBDEV and display output is defined as DVI. Display device is mapped to /dev/fb1 and OSD is mapped to /dev/fb1 frame buffers. Because of VGA resolution of camera, video standard of display defined as VGA. Therefore we see the camera output at full screen. Because of undefined ARGB color space, color space is defined as RGB565.

```
const Display_Attrs Display_Attrs_O3530_VID_DEFAULT = {
   1,
   Display_Std_FBDEV,
   VideoStd_VGA,
   Display_Output_DVI,
   "/dev/fb1",
   0,
```

```
    ColorSpace_RGB565,
    -1,
    -1,
    FALSE,
    0,
    0,
    FALSE
};

const Display_Attrs Display_Attrs_O3530_OSD_DEFAULT = {
    1,
    Display_Std_FBDEV,
    VideoStd_VGA,
    Display_Output_DVI,
    "/dev/fb2",
    0,
    ColorSpace_RGB565,
    -1,
    -1,
    FALSE,
    0,
    0,
    FALSE
};
```

_SysFs.c file is shown in figure 4.15 defines the configuration of OMAP Display Sub-system. In the EVM platform DVI output is mapped to display2 output. However In the Beagleboard-XM platform DVI output is mapped to display0. So in this file we convert the display2 parameters to display0 parameters. Furthermore from overlay configuration part we set the output of applications to overlay1. At that point we finished the modifications for display driver.

The most important parameter modification for capture driver is made on capture.c and _VideoBuf.c files are shown in figure 4.15. There are necessary “Input Output Control Logic” (IOCTL) requests in that files. To configure the capture driver with DMAI this requests must be modified according to USB webcam. To define the

format of image data exchanged between driver and application, DMAI must configure V4L2 driver with VIDIOC_S_FMT request.

Int ioctl(fd, VIDIOC_S_FMT, &fmt)

fd = File descriptor returned by open().

This ioctl is used to used to negotiate the format of data (typically image format) exchanged between driver and application. Here we must set the pixel data format of used camera as YUYV So before calling this ioctl we must set the fmt parameters as below.

fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;

After this step we must disable the requests that are not supported by USB webcams. This request are VIDIOC_ENUMSTD, VIDIOC_QUERYSTD and VIDIOC_G_STD requests. VIDIOC_ENUMSTD request enumerates supported video standards. VIDIOC_QUERYSTD request sense the video standard received by the current input. VIDIOC_G_STD request query or select the video standard of the current input. USB webcams return this request with an error. So we must disable these requests and set the video standard manually as given below.

attrs->videoStd = VideoStd_VGA

The next step is configuration of the capture buffer information. There is two way of it. First one is allocating buffer from device memory map by using virtual addresses of device memory and the second way is using user allocated buffers. To define to memory buffers that will be used by application, DMAI must configure V4L2 driver with the VIDIOC_REQBUFS request.

Int ioctl(int fd, int VIDIOC_REQBUFS, struct v4l2_requestbuffers *argp);

fd = File descriptor returned by open().

This ioctl is used to initiate memory mapped or user pointer I/O. Memory mapped buffers are located in device memory and must be allocated with this ioctl before they can be mapped into the application's address space. User buffers are allocated by applications themselves, and this ioctl is merely used to switch the driver into user pointer I/O mode. To allocate device buffers applications initialize three fields of a v4l2_requestbuffers structure. They set the type field to the respective stream or buffer type, the count field to the desired number of buffers, and memory must be set to V4L2_MEMORY_MMAP. When the ioctl is called with a pointer to this structure the driver attempts to allocate the requested number of buffers and stores the actual number allocated in the count field. It can be smaller than the number requested, even zero, when the driver runs out of free memory. A larger number is possible when the driver requires more buffers to function correctly. When memory mapping I/O is not supported the ioctl returns an EINVAL error code. Applications can call VIDIOC_REQBUFS again to change the number of buffers, however this cannot succeed when any buffers are still mapped. A count value of zero frees all buffers, after aborting or finishing any DMA in progress, an implicit VIDIOC_STREAMOFF. To negotiate user pointer I/O, applications initialize only the type field and set memory to V4L2_MEMORY_USERPTR. When the ioctl is called with a pointer to this structure the driver prepares for user pointer I/O, when this I/O method is not supported the ioctl returns an EINVAL error code.

In this thesis work the driver of the used USB webcam do not support the user pointer I/O. So device memory map is used to get the data bytes of captured frames. 3 capture buffer requested from driver and virtual addresses of these buffers are used by setting a pointer to the starting address of these buffers.

After making necessary configuration of capture and display files of DMAI folder we can get the data of capture buffers and we can put the desired data inside the display buffers. And now to make a video loopback connection between capture and display buffers we must convert the capture pixel format to display pixel format. In

this work, YUYV capture format and RGBA display format is used. Figure 4.16 is show the YUYV pixel format and figure 4.17 is show the RGBA pixel format

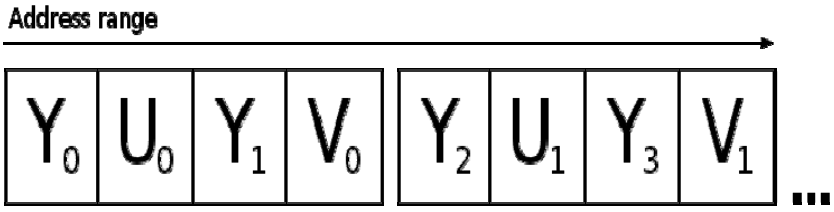


Figure 4.16 YUYV Pixel Format

YUYV pixel format is a subspace of YUV color space. YUV color space encodes a color image and result with a reduced bandwidth for chrominance components. This enables that transmission errors or compression artifacts are more efficiently masked by the human perception than using a "direct" RGB-representation. In YUYV format each two pixel is defined by four bytes. The first and third bytes (Y) are the Luma (the brightness) components of pixels. The second and fourth bytes (UV) are two chrominance (color) components of pixels.

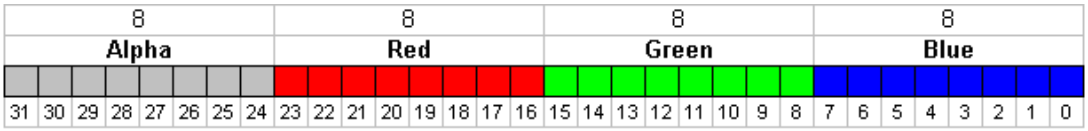


Figure 4.17 RGBA Pixel Format

RGBA is the abbreviation of Red Green Blue Alpha. It is a color space of RGB color model, with extra information alpha. The alpha channel is the opacity channel. Alpha pixel value determines the transparency of the display. For Alpha value “0”, it is fully transparent , whereas Alpha value “255” gives a fully opaque pixel.

Below expressions shows the necessary mathematical equation to convert YUYV pixel format to RGBA pixel format. In the conversion process 4 bytes, 2 pixel data of YUYV pixel format (Y0 U Y1 V) is read and converted to 8 bytes, 2pixel data of RGBA pixel format (A0 R0 G0 B0 A1 R1 G1 B1). Here Alpha values A0 and A1 are not depend on the input bytes and defined by user. Here Alpha value is selected as “255” for a fully opaque pixel.

$$\begin{bmatrix} R0 \\ G0 \\ B0 \end{bmatrix} = \begin{bmatrix} 0x2543 & 0 & 0x3313 \\ 0x2543 & -0x0C8A & -0x1A04 \\ 0x2543 & 0x408D & 0 \end{bmatrix} * \begin{bmatrix} Y0 - 16 \\ U - 128 \\ V - 128 \end{bmatrix} \quad (1)$$

$$A0 = 0xFF \quad (2)$$

$$\begin{bmatrix} R1 \\ G1 \\ B1 \end{bmatrix} = \begin{bmatrix} 0x2543 & 0 & 0x3313 \\ 0x2543 & -0x0C8A & -0x1A04 \\ 0x2543 & 0x408D & 0 \end{bmatrix} * \begin{bmatrix} Y1 - 16 \\ U - 128 \\ V - 128 \end{bmatrix} \quad (3)$$

$$A1 = 0xFF \quad (4)$$

After getting the YUYV pixel format captured data we must make the necessary conversion to display it in RGBA format. We can make the conversion by using ARM core or DSP core of the DM3730 processor. However to use the DSP core of the system, captured data must be written on shared memory continuously, DSP CMEM driver must be loaded with correct parameters and conversion algorithm must be added codec server (cs.x64) file. These steps will be explained in part 5.4. ARM core can use the pointers of device driver directly to make the necessary conversion.

After conversion we must arrange the display driver. For DM3730 processor there are two display driver namely OMAPFBdev (omap_fb.ko) and V4L2 (omap_vout.ko) display drivers. V4L2 driver is used generally for video streaming

whereas OMAPFBdev driver is used for graphics. For our application using V4L2 driver is much more appropriate because of its streaming capability and supporting YUYV pixel format. By using this display driver it is possible to make a pipeline between video capture and display buffers without any conversion. However kernel compiled by Narcissus image builder do not have omap_vout.ko driver and it can not be added by using menuconfig command of kernel. Because Narcissus image builder does not produce the sources files of kernel at the output file system. So OMAPFBdev driver is used for display operations. Details of OMAPFBdev driver parameters is given in [56] . In our kernel we have three framebuffer (fb0, fb1, fb2) map to three separate overlays (overlay1 and overlay2 and overlay3). We have two output devices namely TV and DVI. At default configuration the fb0 framebuffer that is used by kernel GNOME window manager is mapped to overlay0 and overlay0 is mapped DVI output of platform. So to see the output of our application on the DVI output over the GNOME window of kernel we must use framebuffer fb1 or fb2 and we must map these framebuffers to overlay1 or overlay2. And then the selected overlay must be mapped to DVI output. In this thesis work we use fb1, overlay1 and DVI output of OMAPFBdev driver and we map these parameters to each other. The other important driver parameters must be set is the resolution of display. It is set to 640x480 VGA format to map the captured frame all of the screen. All of the configuration of this parameters must be set from _SysFs.c file as explained before.

Here the other important thing disabling the overlay1 while exit from application to return the GNOME window manager of kernel. Because if all of the overlays enabled with opaque transparency, overlay2 covers on the overlay1 and overlay1 covers on the overlay0.

4.4.3.2 Configuration of Shared RAM Memory

The Beagleboard-XM is equipped with 512 megabytes of DDR2 memory. A user defined part of DDR2 memory is shared between the ARM and the DSP cores and both cores can access the shared part of the DDR2. The ARM however views this memory as virtual addresses through an MMU (Memory Management Unit) while the DSP uses the physical addresses directly. The virtual addresses is used by Linux to provide memory protection between processes, making sure a process only

accesses memory which it has access to. Since the DSP has no MMU it can not be restricted to certain memory addresses and it can over the ARM (Linux) side code and data. This issue result with a fatal error. To prevent this situation CMEM driver is used to arrange the shared memory used between ARM and DSP cores. By the help of this driver ARM and DSP cores work on physical address of shared memory. The physical memory addresses are the same for the ARM and the DSP on DM3730 and range from 0x80000000 to 0x90000000.

In this part of the thesis, after giving some background on various elements of shared memory, we discuss how to how to partition that memory. We partition the shared memory to six elements, namely Linux partition, Contiguous Memory Allocator, DDRALGHEAP, DDR, DSPLINKMEM and RESET_VECTOR.

The Linux partition is used for various internal I/O buffers and application caching features, so the bigger this partition is, the better.

CMEM partition is used to share buffers between ARM processes and the DSP. It takes a physical memory region you specify at CMEM driver load time and carves it up in to pools of contiguous buffers according to your specifications. The buffers are typically not cached on the ARM side whereas they are cached at the DSP side by the help of Codec Engine.

The DDRALGHEAP partition contains the heap from which the active codecs allocate all their dynamic memory. This section can be quite large, especially if video codecs are used.

The DDR partition contains DSP side code and static data for all the codecs plus the system (i.e. DSP/BIOS and Codec Engine).

DSPLINKMEM partition is used by the DSPLink IPC software from TI. Codec Engine uses this software module for communicating between the ARM and the DSP as well as loading the DSP with code and controlling it.

RESET_VECTOR partition contains the DSP reset vector, i.e. the vector table which the DSP side ISTP register is pointing to when the DSP is pulled out of reset by DSP

Link. The reset vector code moves the vector table else where by changing the ISTP, but this is where it is located at boot. This section needs to start at an even 1MB and needs to be 128 bytes in size.

Sum of the size of these partitions gives the total RAM memory of the platform. In our case total memory is 512MB.

To put it in an equation,

```
512MB = Linux Memory + CMEM memory + DDRALGHEAP + DDR + DSPLINKMEM + RESET_VECTOR
```

from which follows that

```
Linux Memory = 512MB – (CMEM memory + DDRALGHEAP + DDR + DSPLINKMEM + RESET_VECTOR)
```

Linux memory partition allocate maximum memory to increase the performance of embedded OS on ARM core. So necessary minimum sizes for the other partitions must be calculated and set to get the optimum performance from system. In this thesis shared memory is not optimized for optimum system performance. However more detailed information can be found from [60] to optimize the shared memory.

The physical memory addresses and sizes for shared memory partitions that are used in this work is given in table 4.5. This partition table shows the first bank of 512MB memory the other banks is used by Linux memory. So Linux memory allocates 438MB from 512MB total memory. The Linux memory partition is determined with the boot.scr parameters at start up. The CMEM partition size is determined while installing the CMEM driver into kernel. The other partitions are determined from server map files (*.tcf files) that is placed in sources files of DSP server. Configuration of server map files used for Running Average Gaussian algorithm is given Appendix H. This map file is placed under ../DVSDK/CODECS-OMAP3530/PACKAGES/TI/SDO/SERVER/CS folder.

Table 4.5. DM3730 Memory Map (128 MB Total Memory)

Address Range	Size	Description
0x80000000 - 0x836FFFFF	55 MB	Linux
0x83700000 - 0x858FFFFF	34 MB	CMEM
0x85900000 - 0x878FFFFF	32 MB	DDRALGHEAP
0x87900000 - 0x87EFFFFF	6 MB	DDR2 (BIOS, Codecs, Applications)
0x87F00000 - 0x87F00FFF	4 KB	DSPLINK (RESET)
0x87F01000 - 0x87FFFFFF	1 MB- 4KB	DSPLINK (MEM)

The figure 4.18 shows the first bank 128MB RAM Memory Map of DM3730 used for Running Average Gaussian algorithm.

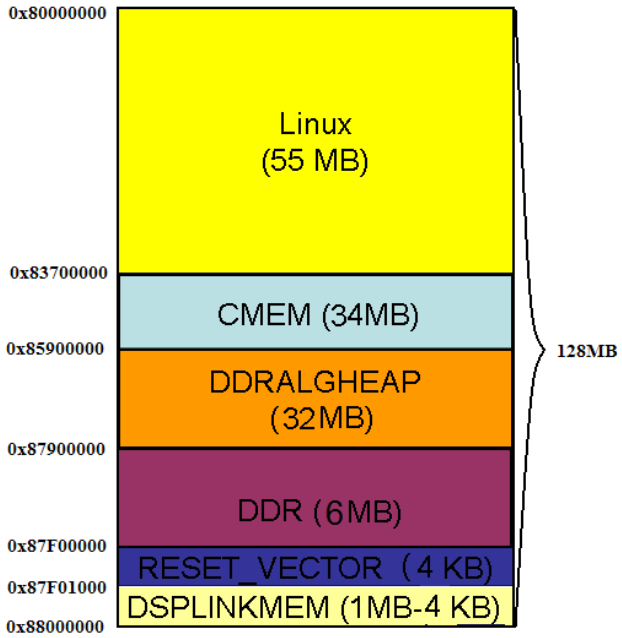


Figure 4.18 DM3730 Memory Map (128 MB Total Memory)

4.4.3.3 Implementation of Smart Camera Applications on DSP Core

Moving object detection, single and multiple object tracking, event recognition and face detection are the necessary and widely used algorithms in automated visual surveillance. Development of image processing algorithms for smart camera design generally focuses on the combination of these algorithms according to application area. Background/foreground detection algorithm is the most important and challenging part of all of these algorithms. Therefore implementation of a successful background/foreground detection algorithm on a smart camera platform is a necessity for the implementation of a variety of smart camera algorithms. In this thesis adaptive and non-adaptive background/foreground detection algorithms are used. As non-adaptive method, pixel difference and thresholding method is used. As adaptive method Running Gaussian Average (RGA) and Gaussian Mixture Model (GMM) background/foreground detection algorithms are selected for implementation.

Pixel difference and thresholding background/foreground detection algorithm is the simplest non-adaptive background/foreground detection method. It is very sensitive to selected threshold and it is not good at different challenging conditions like illumination changes and shadows in scene. In this method difference between current frame pixel and previous frame pixel is calculated and this difference is compared with a predefined threshold value. If difference is greater than threshold value, this pixel belongs to foreground. Otherwise it is belong to background. Figure 4.19 shows the background foreground detection process based on pixel difference and thresholding method.

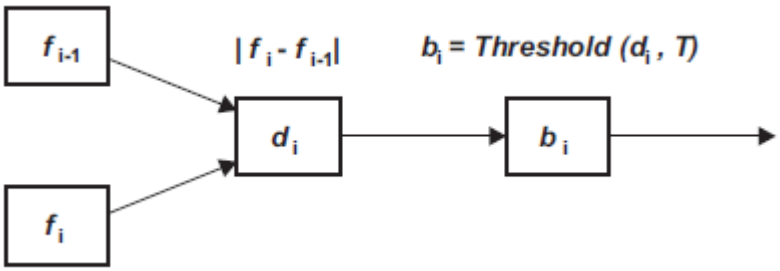


Figure 4.19 Background Foreground Detection Based on Pixel difference and thresholding

Descriptions of the parameters shown in figure 4.19 are given below.

· fi : A pixel in a current frame

· fi-1 : A pixel in a previous frame

· di : Absolute difference of pixels |fI - fI-1|

· bi : Background/Foregroundmask – background = 0 and foreground = 0xFF.

· T : Threshold value.

RGA background/foreground detection algorithm is the a moderate adaptive background/foreground detection method. This method is based on a statistical background model and can adapt to dynamic light changing and environmental noise. In this method a pixel of video frame is model as a random variable that follows a Gaussian distribution that as determined by its mean and standard deviation. Mean and standard deviation of a pixel is calculated continuously from the frames of live video data. The pixels whose values are in the standard deviations of the corresponding Gaussian distribution belong to the background. Otherwise, they belong to the foreground. Figure 4.20 shows the background foreground detection process based on Running Average Gaussian.

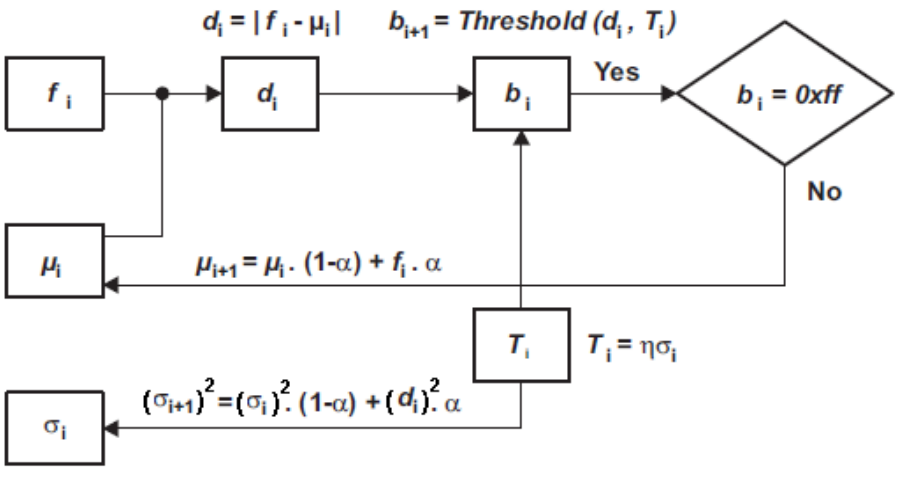


Figure 4.20 Background Foreground Detection Based on Running Average Gaussian

Descriptions of the parameters shown in figure 4.20 are given below.

· fi : A pixel in a current frame

· μi : Mean of a pixel-wise background Gaussian distribution

· σi : Standard deviation of a pixel-wise background Gaussian distribution.

· di : Absolute difference between fi and μi.

· Ti : A pixel-wise threshold.

· a : Learning rate of the background.

· η : Threshold gain.

GMM background/foreground detection algorithm is the most successful background/foreground detection algorithm because of its success on different challenging conditions like illumination changes and shadows in scene. However GMM algorithm is a complex algorithm that needs a lot of mathematical operations for each pixel on the scene. Because of this, DSP core of the processor is much more appropriate to implement this algorithm. However, implementation of complex algorithm on DSP core is not the scope of this thesis. So GMM algorithm is implemented on only the ARM core to see the performance of the algorithm. GMM background/foreground detection algorithm is a widely used approach for background modeling to detect moving objects as foreground objects from static cameras. This background subtraction model is a parametric approach designed by Zivkovic and van der Heijden [2, 3]. In this approach, each pixel is compared with each Gaussian and is classified according to its corresponding Gaussian. A mixture of Gaussians for the base distribution for each pixel's color values are calculated and maintained. The mean and covariance of each pixel in the mixture is updated in every new frame to reflect the change of the pixel values. This model calculates the Mahalanobis distance of the pixels from the RGB value to the pixel's means by using calculated mean and covariance of each pixel in the mixture. The pixels those Mahalanobis distance are larger than, for instance, three times of the standard deviation are considered as the foreground pixels. Mahalanobis distance is based on the correlations between pixel's variables by which different pixels can be identified and analyzed. In other words Mahalanobis distance of the pixels gauges the similarity of new frame's pixels to older frame's pixels. The details and mathematics of the algorithm can be found in [2, 3].

GMM background/foreground detection algorithm decreases the processing time for each frame while achieving good segmentation. However, this algorithm is still not fast enough for real time applications, especially when the algorithm has to deal with high-resolution video segments.

In this thesis work the C6Accel package is used to create the DSP side algorithms that is compatible with xDAIS standards. By the help of C6Accel, implemented algorithms can be invoked from the ARM side using simple API calls. C6Accel can be used in a plug and play like any other codec used for encoding and decoding audio and video streams. C6Accel is built in the codec engine compliant IUniversal framework and can be used on various DSP only and ARM + DSP devices. The purpose of C6Accel is to provide the ARM user with the compute power of the DSP on computational intense tasks like running Color Space Conversion, Filtering or Image/Signal Processing algorithm. The library of DSP kernels wrapped in C6Accel are optimized for performance on the DSP core and would allow the ARM user to use the DSP as an accelerator for their application. By using these routines, the ARM developer can develop a more compelling application by achieve execution speeds considerably faster than equivalent C code written on ARM. In addition, by providing ready-to-use DSP kernels, C6Accel can significantly shorten the ARM application development time.

The benefits of using C6Accel include:

- Ready to use kernels: Library of Optimized DSP kernels wrapped in a single package. Reduces learning curve and time to market.

- Easy to interface: ARM side API library abstracts complexities while invoking DSP functionality from ARM application

- Easy Portability: Fully compatible with most TI C6x devices

- Efficient multiple call execution: Capability to chain kernel calls using single call to codec engineEasy

- Evaluation of DSP performance: DSP kernel Benchmarks (cycle and code size) provided in C6Accel aid in evaluating performance that can be leveraged from the DSP and make informed decisions while developing applications

- Parallel processing: Asynchronous calling mode enables parallel processing on DSP and ARM

- Simple Template to add functionality on DSP: SoC developers can explore maximum flexibility by using C6Accel algorithm as a template to add custom compute intense functionality on the DSP that can be accessed from the ARM.

In this work implemented algorithm for DSP side is added to IMGLIB library of C6Accel. To add a DSP algorithm to the C6Accel infrastructure, a user defined ID for the function of algorithm and input output structure of function must be added the library and API call sources files placed inside the c6Accel folder. The steps to add the Running Average Gaussian function to C6Accel component is given below. More detailed information is given in [58].

Step 1 : Implement the desired Algorithm for DSP.

Running Average Gaussian algorithm is implemented in that step. Three input buffer, one output buffer and an input image size parameter is used for algorithm. Equation for Running Average Gaussian algorithm is given below.

M: Mean buffer
V: Variance buffer
F: Captured Frame buffer
O: Output buffer
Alfa : Adaptation rate
THR: Threshold

$$M(t) = Alfa * F + (1-Alfa) * M(t-1) \quad (5)$$

$$V(t) = \text{sqrt}(\text{Alfa} * (F-M[t-1]) * (F-M(t-1)) + (1-\text{Alfa}) * v(t-1) * v(t-1)) \quad (6)$$

$$THR = F-M(t) / v(t); \quad (7)$$

$$\text{IF } THR < 2*v(t) \text{ then } O = 0xFF \quad (8)$$

$$\text{ELSE } O = 0x00 \quad (9)$$

Here we capture a YUYV pixel format and 640x480 pixel resolution image frames from camera. This means we have two bytes for each pixel. So the size of our input buffer is equal to:

$$\text{Input Buffer Size} = 640 \times 480 \times 2 = 614400 \text{ bytes} \quad (10)$$

At the calculation of mean and variance only luma values of pixel is used. This means for each pixel we have one byte data. So the size of our mean and variance buffer is equal to:

$$\text{Mean Buffer Size} = \text{Variance Buffer Size} = 640 \times 480 = 307200 \text{ bytes} \quad (11)$$

Input image size parameter holds the data of multiplication of resolution.

$$\text{Size of Input image Size parameter} = \text{size of } (640 \times 480) = 15 \text{ bits is enough} \quad (12)$$

For each input pixel we produce an ARGB pixel format bytes for output display. This means we have four bytes for each pixel. So the size of our output buffer is equal to:

$$\text{Output Buffer Size} = 640 \times 480 \times 4 = 1228800 \text{ bytes} \quad (13)$$

Therefore we need one 1228800 bytes , one 614400 bytes and two 307200 bytes buffers and one 15 bit integer for our algorithm implementation. These size will be used to determine the necessary size for CMEM partition and to determine the size of pools parameters that is at CMEM driver installation. The source code of algorithm is given in Appendix G as RGA_DSP.c.

Step 2 : Add algorithm ID to C6Accel.h and iC6Accel_ti.h files. The place of files and definition of ID is given in Appendix G as C6Accel.h and iC6Accel_ti.h.

Step 3 : Add the call API of algorithm library from ARM side to c6accelw.c file. An implementation for that step is given in Appendix G as c6accelw.c

Step 4 : Add the call API of algorithm from codec engine side to C6accel_ti_imglibFunctionCall.c file. An implementation for that step is given in Appendix G as C6accel_ti_imglibFunctionCall.c

Step 5: Configure the shared memory map from server.tcf file that is placed under ../DVSDK/CODECS-OMAP3530/PACKAGES/TI/SDO/SERVER/CS folder

Step 5: Configure the shared memory map from server.tcf file that is placed under ../DVSDK/CODECS-OMAP3530/PACKAGES/TI/SDO/SERVER/CS folder

Step 6: Compile the codec server from DVSDK folder with below make commands

$../DVSDK/ make c6accel_clean codecs_clean

$../DVSDK/ make c6accel codecs

$../DVSDK/ make c6accel_install codecs_install

Step 7: Get the *.x64P DSP codec server file from the codec install directory of compiler. Install directory of compiler is defined in Rules.make file under DVSDK folder. In this work codec install directory is set as below.

```
# The installation directory of the SDK.
DVSDK_INSTALL_DIR=/home/ti-dvsdk_dm3730-evm_4_01_00_09
# Where the codecs are installed.
CODEC_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/codecs-omap3530_4_01_00_00
```

Step 8: Implement the desired Algorithm for ARM and call DSP function.

Implementation of ARM side application is achieved by using the demo applications under the DVSDK.

Step 9: Compile the arm application from DVSDK folder with below make commands

$../DVSDK/ make demos_clean

$../DVSDK/ make demos

$../DVSDK/ make demos_install

Here if a new folder for a project is created under ../DVSDK/DVSDK-DEMOS/OMAP3530 folder, the name of new application folder must be added make file under ../DVSDK/DVSDK-DEMOS

Step 10: Get the executable output of ARM application from the demo install directory of compiler. Install directory of demo is defined in Rules.make file under DVSDK folder. In this work demo install directory is set as below.

```
# Where the SDK demos are installed
DEMO_INSTALL_DIR=$(DVSDK_INSTALL_DIR)/dvsdk-demos_4_00_00_21
```

Step 11: Prepare a script file to install the DSPLink, LPM and CMEM drivers with determined pool sizes.

Step 12: Copy the *.x64P DSP server, executable ARM application and driver installation script same place. Run the driver script and confirm that driver is load from console.

Step 13: Run executable ARM application from console.

At that step ARM application run on ARM core and call DSP application from DSP server. Data sharing between these cores is achieved on the CMEM shared memory defined by CMEM driver installation parameters.

CHAPTER 5

IMPLEMENTATION AND PERFORMANCE EVALUATION

Our performance evaluation is done by using Beagleboard-XM platform and a host PC. To evaluate the performance of the platform, a comparison is made with a host PC having a 2.80 GHz Intel core processor and 512 MB DDR2 memory. The specifications of the system components are listed in table 5.1.

Table 5.1. Specifications of components of the experiments environment

	HOST PC	Target Board
OS	Ubuntu Release 10.04 (lucid)	Angstrom Release 2011.03 (Dureza)
KERNEL	2.6.32	2.6.32
MEMORY	512 MB DDR2	404.6 MB LPDDR
PROCESSOR	INTEL(R) CORE(TM) 2 DUO* CPU T9600 2.80 GHz	ARM Cortex-A8 1 GHz DSP TMS320C64X+ 800 MHz
COMPILER	gcc version 4.4.3 Target: i486-linux-gnu	g++ version 4.3.3-r23.1.6 Target: arm-linux-gnu
ARM SDK	Eclipse SDK Version 3.5.2 CDT GNU Tool Chain 6.0.0	Use Outputs of Host SDK
ARM+DSP SDK	Modified DVSDK 4.01	Use Outputs of Host SDK

In this chapter, we will focus on the performance of the DM3730 processor for video capture, video decode and image processing. We will measure the performance contributions of NEON co-processor, DMA hardware accelerator and DSP core on the applications that are run by ARM core .

5.1 Video Loopback Process

In this section we will evaluate the video loopback performance of DM3730 processor. Video loopback can be thought as a simple camera model. In this model, a frame is captured from image sensor by the help of the video capture driver and captured frame is converted to a compatible format for display unit. Then format converted frames are sent to a display unit by using video display driver. The simplest implementation of this process is using the OpenCv libraries installed on embedded OS. By using these libraries we do not struggle with capture and display driver configurations. However, we can achieve this type of implementation only on ARM core or NEON co-processor to use OpenCv libraries in our application. Figure 5.1 shows the implementation of video loopback process on ARM core

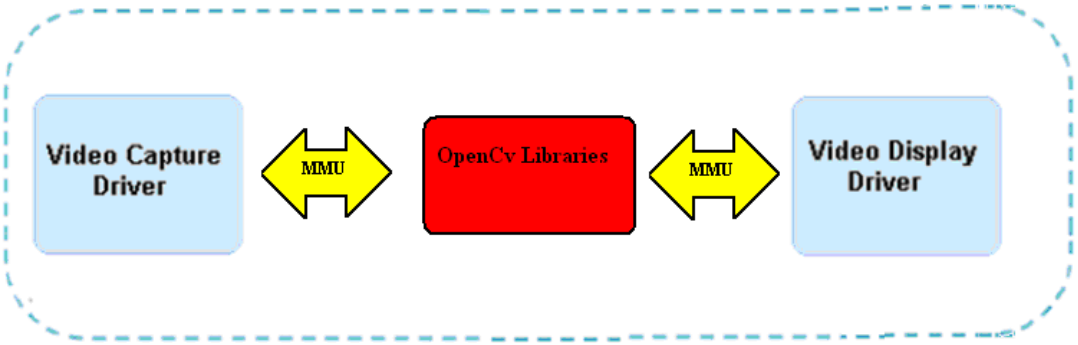


Figure 5.1 Block Diagram of Video Loopback Process on ARM

As can be seen from figure 5.1 OpenCv libraries capture frame data from video capture driver by using Memory Management Unit (MMU) of kernel and put this frame data to display driver with same structure. To implement this model on the

ARM core or NEON co-processor, we implement a code by using the OpenCv libraries. Details of the code are given in Appendix A-ii.

This code is written on the host PC. Code is implemented as compatible both the host PC and target board environment. 640x480 resolution YUYV pixel format A4Tech USB webcam is used as image sensor. Rated fps of the USB webcam is 30 fps and its speed changes with illumination of environment.

Video loopback process can be also implemented by using DMA hardware accelerator of DM3730 processor to speed up the process. Figure 5.2 shows the implementation of video loopback process with DMA hardware accelerator. However in this case we must struggle with capture and display driver configurations and we must convert the capture format to display format inside the application. In our work, frames are captured in YUYV pixel format from a USB webcam and they are displayed in ARGB pixel format on DVI output of platform. Figure 5.2 shows the implementation of video loopback process with DMA.

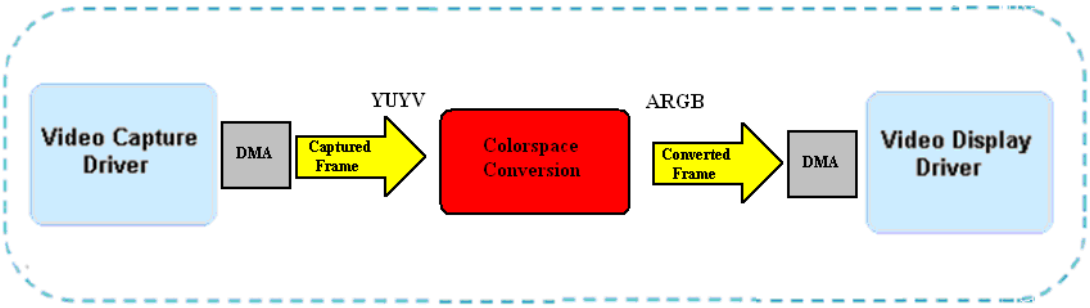


Figure 5.2 Block Diagram of Video Loopback Process with DMA

As can be seen from figure a frame is captured from capture driver with DMA hardware accelerator in YUYV format and captured frame is sent to a color space conversion application. Then the ARGB output of color space conversion application is sent video display driver with DMA hardware accelerator. In this case color space

conversion application can be implemented on ARM core or DSP core of the DM3730 processor. The performances of these cores effect the processing time of color space conversion application and that affects overall performance of application. Here because of the large size of the image data that will be processed by color space conversion application, the SIMD architecture of DSP can be good solution to speed up the system performance. However, we must put the captured image data on the shared RAM memory to process it with DSP. Furthermore, DSP writes again the outputs of the processed image data on the shared RAM memory. This means captured YUYV pixel format frames must be put on shared RAM memory and processed ARGB pixel format frames must be read by display driver from shared RAM memory. The figure 5.3 shows the implementation of video loopback process on ARM + DSP cores.

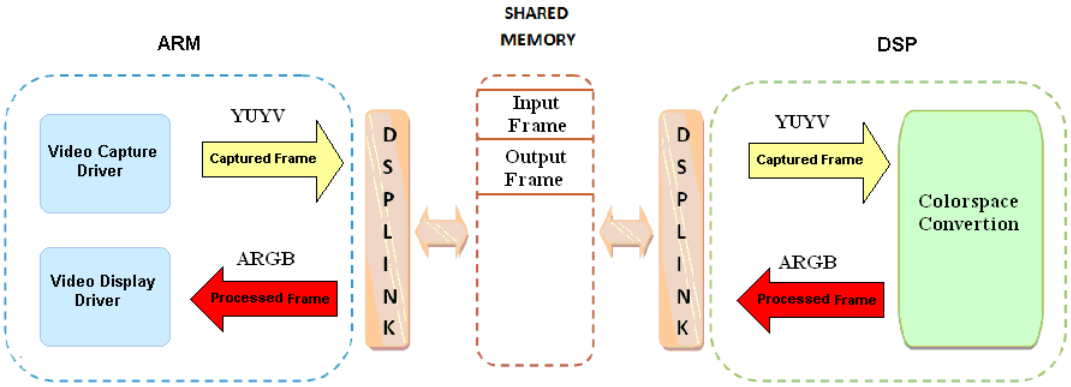


Figure 5.3 Block Diagram of Video Loopback Process on ARM + DSP

Table 5.2 compare the performances of different implementations on Beagleboard-XM platform for video loopback process. Furthermore, the performance of the host PC for video loopback application is also given as reference.

As you see from table 5.2 host PC can capture the 640x480 resolutions frames at 20.2 fps speed by using %17 of its CPU and %3.6 of memory. However target board can process same resolutions frames at 11 fps speed by using %73 of its CPU and

%7.7 of the memory with his ARM core. As can be seen from table using NEON co-processor for video loopback process does not change the performance of overall system. On the other hand, using DMA hardware accelerator for process, increase the

Table 5.2. Performance of Beagleboard-XM for Video Loopback Process

Experiment Setup	Host PC	Beagleboard-XM Platform			
		ARM	NEON	DMA + NEON	DMA + DSP
Resolution	640x480	640x480	640x480	640x480	640x480
CPU Load %	17	73	73	97	70
Memory Usage %	3,6	7.7	7.7	0.6	0.6
Processed FPS	20.2	11	11	22	10
FPS/CPU	0.119	0.015	0.015	0.023	0.014

system performance drastically. However, in that case CPU usage of system goes up to %97 and this is a undesired condition for operation of an embedded OS. Another interesting point is low memory usage in DMA case. The reason for that DMA uses the shared memory and embedded operation systems do not calculate memory usage from shared memory. As can be seen from table 5.2 DMA+DSP implementation does not increase the system performance according to DMA+NEON case. We will investigate that situation at the following sections.

Here the ratio of processed frames per second to CPU usage is thought as processing capability of processor with %1 CPU usage. This value is used to make a comparison between performances of the host PC and target board.

This experiment shows that processing capability of host PC is on the average 7.9 times of processing capability of ARM core of the target board although the CPU processor speed of host PC is 2,8 times of the ARM core processor speed of the target board. NEON co-processor does not affect the processing speed for video loopback operation. This situation shows that video loopback processing speed is not

related only with the CPU of the system. In this application the limit is memory reading and writing speed and it takes 91 msec capturing and displaying a frame for ARM case. In ARM case memory transfer operations and cpu usage operation of application is controlled by the embedded OS and this affects the performance of application. On the other hand using DMA hardware accelerator speeds up the memory transfer operations while it also increase the CPU usage of system. Using DMA increase overall system performance %50. In DMA+NEON case, capturing and displaying a frame takes 45 msec. 45 msec is almost the rated fps of USB camera. However, because of CPU usage this performance can not catch the performance of host PC. At that point using DSP processor beside the DMA hardware accelerator decreases the CPU usage of ARM processor. However, because of extra frame transfers operation between shared memory, ARM and DSP, fps value decreases to 10 fps in DSP case. Figure 5.4 shows the frame processing speed for different operations in DMA + DSP case.

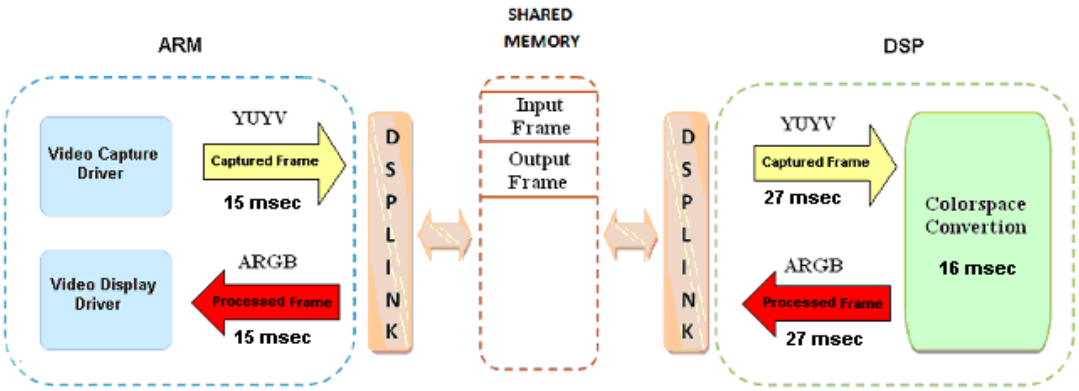


Figure 5.4 Speeds of Memory Transfer Operations and DPS Image Processing

As can be seen from figure 5.4 memory transfer operation takes 84 msec totally whereas DSP processing speed 16 msec. So in overall system to process a frame we need 100 msec and this equals to 10 fps.

Memory transfer operations between device drivers and DSP may be speeded up by using some architectures in DSP. The DMA engine can be used on the GPP processor to speed up memory transfers. Instead, on the DSP side is provided an Enhanced Direct Memory Access (EDMA) Controller for data transfers between the L2 cache and the device peripherals. This unit may speed up data transfer when peripheral devices are directly connected to the DSP. The figure 5.5 shows the DSP core process in an Application. As can be seen from figure data can be cache to L2 cache by DDR interface or may be cached EDMA.

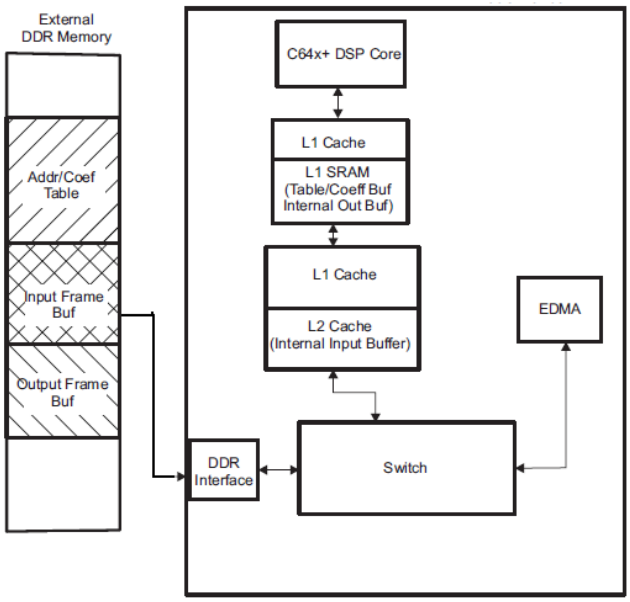


Figure 5.5 Block Diagram of DSP core Process in an Application

As a summary video loopback application by using DMAI hardware of DM3730 speeds up the video loopback application and we can process the rated fps of USB webcam. However ARM CPU load goes to %97 and no processing power for other applications remains. Therefore at that point addition of the background and foreground image separation process to the algorithm will decrease the processed fps because of full load of CPU. We will see the effect of additional Running Gaussian Average background and foreground image separation algorithm on the speed of

process at part 5.3. On the other hand using DSP besides the DMA do not increase the performance of overall system and fps of process decrease because of the extra memory transfer speed. Memory transfer speed from device drivers to DSP can be increased by optimizing memory path between DSP and device drivers. Also L2 cache and Enhanced DMA of DSP core may be used the speed up data transfer between device drivers to DSP.

5.2 Background/Foreground Detection Process

In this section, we evaluate the image processing performance of Beagleboard-XM platform. In this thesis to evaluate the image processing capability of the platform, three different background/foreground detection algorithms namely pixel difference, RGA and GMM background/foreground detection algorithms are used during the experiments.

Figure 5.6 shows the block diagram of the GMM background subtraction process implemented on ARM core of DM3730 processor. Details of the code are given in Appendix A-iv.

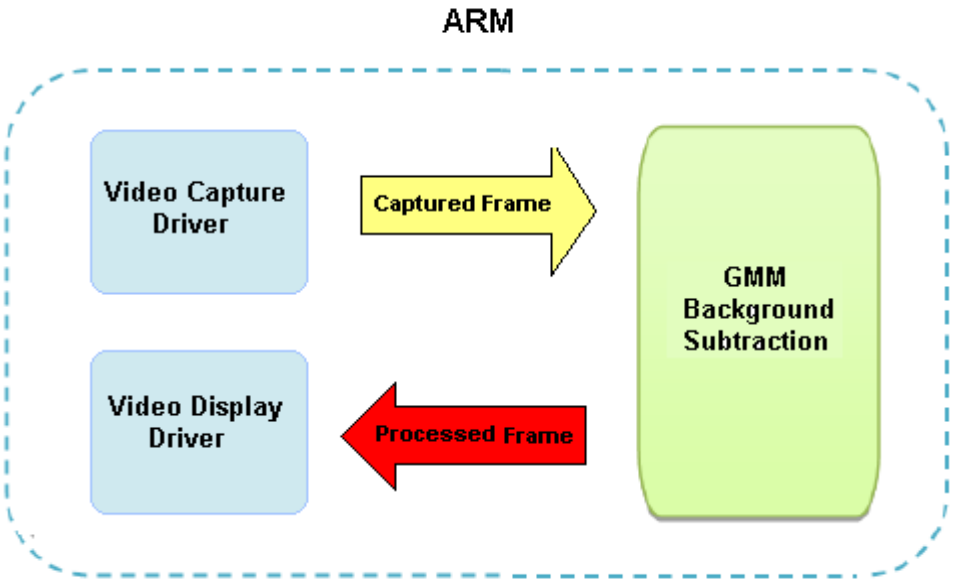


Figure 5.6 Block Diagram of GMM Background/Foreground Detection Process

Table 5.3 shows that ARM core of the target board can capture, process and display the 640x480 resolution frames by using GMM background subtraction algorithm at a speed of 0.8 fps by using %95 of its CPU. This is a low rate to design a smart camera that has real time image processing capability. On the other side host PC can capture, process and display 640x480 resolution frames by using GMM background subtraction algorithm at 15 fps by using %80 of its CPU. Furthermore the comparison between processing capability of host PC and target board shows that GMM background subtraction algorithm works 22 times faster than target board. This is a very big performance difference when we look at the result of the former experiments. The most probable reason for that is the usage of ram memory more than one times to read and write data at different levels of GMM background subtraction algorithm according to former experiments and saturation of ARM core CPU usage. Here we also compile the code to use the NEON co-processor of the ARM. The only thing to use neon co-processor is adding NEON parameters to gcc compiler.

Table 5.3 shows that NEON co-processor of the target board can capture, process and display the 640x480 resolution frames by using GMM background subtraction algorithm at a speed of 1.2 fps by using %95 of its CPU. This result shows that NEON core speed up the process at a %50 rate. Furthermore the comparison between processing capability of host PC and target board shows that GMM background subtraction algorithm works 15 times speeder than target board when we use NEON co-processor.

Table 5.3. Gaussian Mixture Model Background/Foreground Detection

Experiment Setup	Host PC	Beagleboard-XM Platform	
		ARM	NEON
Resolution	640x480	640x480	640x480
CPU Load %	85	95	95
Memory Usage %	8,8	14.1	14.1
Processed FPS	22.4	0.8	1.2
FPS/CPU	0.26	0,008	0,013

a)

b)

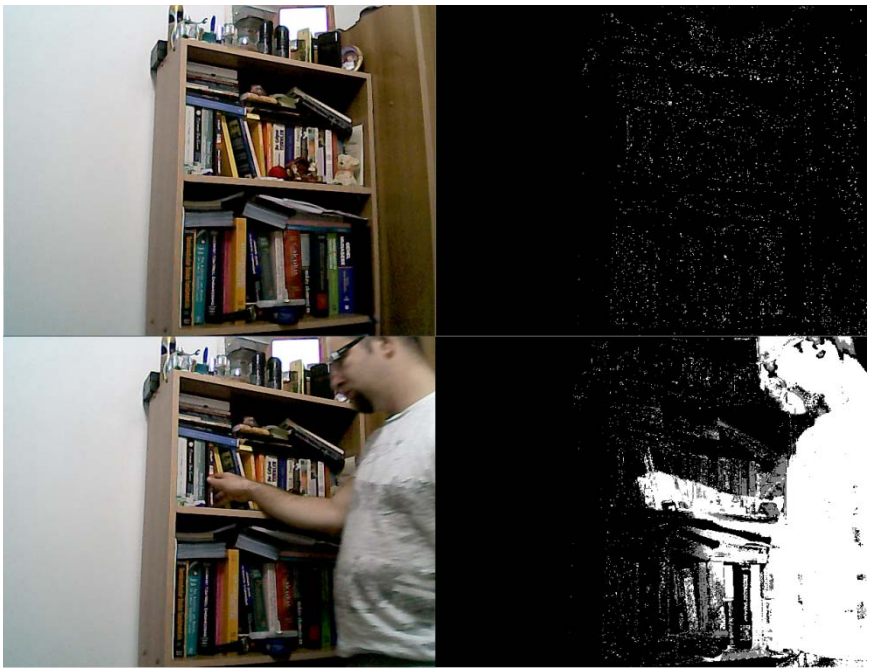


Figure 5.7 GMM Background Subtraction Result of a) Host PC b) Target Board

Figure 5.7-a shows the results of the GMM background subtraction algorithms implemented on host PC. The upper two frames show the original background image and its foreground mask. The effects of changing light conditions in the environment can be seen from the foreground image that is belonging to the only background model. The lower two frames show the moving objects that is newly entering the scene and its foreground mask. Most of the moving pixels can be seen from the foreground image and this shows the success of background subtraction algorithm on host PC. Figure 5.9-b shows the results of the GMM background subtraction algorithms implemented on target board for. The upper two frames show the original background image and its foreground mask. The lower two frames show the moving objects that newly enter the scene and its foreground mask. If we compare the results of GMM background subtraction algorithm implemented on host PC and target board, we can see that target board foreground image that is belong to the only background model do not differ so much from the host PC foreground image. However, target board foreground image that belongs to the moving object differs so much from the host PC foreground image and most of the moving pixels can not be seen from the target board foreground image. Although most of the details of moving objects are lost in the target board implementation, this result can be sufficient to implement some simple moving object detection algorithms on the target board. Additionally we can extract from this experiment that an increase in the resolution of image result with a decrease in the number of processed images while it result with an increase in the success of GMM background algorithm.

In this work, to see the contribution of DSP core on image processing, Pixel difference and RGA background/foreground detection algorithms are implemented on DSP core. Figure 5.8 show the block diagram of RGA background/foreground detection algorithm. As can be seen from figure 5.8 processing time of DSP is 500 msec and this is a very high value for real time image processing. There are two reasons for that situation. First reason is floating point operations included in algorithm and non-optimized structure of DSP code according to VLIW. Second reason is extra memory transfer operations for the new mean and variance operations.

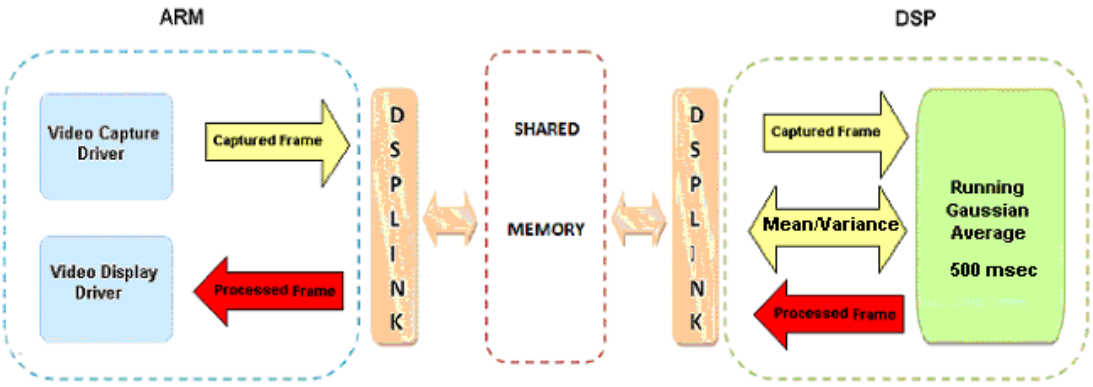


Figure 5.8 Block Diagram of RGA Background/Foreground Detection Process

In the RGA algorithm we are taking square root to calculate the variance and only this operation takes the 79 cycle of the DSP. Also floating operation on DSP decreases the performance of it. Here we also use the fastRTS_i library to calculate the floating point operations in DSP and this really increase the performance of algorithm. Processing time of DSP was 1750 msec without fastRTS_i library.

Table 5.4 shows the performance of Beagleboard-XM platform for RGA Background/Foreground Detection algorithm.

Table 5.4. Running Gaussian Average Model Background/Foreground Detection

Experiment Setup	Beagleboard-XM Platform	
	DMA + NEON	DMA + DSP
Resolution	640x480	640x480
CPU Load %	97	17
Memory Usage %	0.6	0.6
Processed FPS	3.3	1.9
FPS/CPU	0.034	0.112

As can be seen from table 5.4 DMA + NEON case processing speed is faster than DMA + NEON case. There are two reason for that situation. First one is floating point operation capability of NEON co-processor and the second one extra memory transfers operations in DMA+ DSP case.

Pixel difference background/foreground detection algorithm does not include floating point operations so we can apply this algorithm to see the only memory depended performance difference of DSP. The table 5.5 shows the performance of Beagleboard-XM platform for pixel difference Background/Foreground Detection algorithm.

Table 5.5. Pixel Difference Model Background/Foreground Detection

Experiment Setup	**Beagleboard-XM Platform**
	DMA + DSP
Resolution	640x480
CPU Load %	64
Memory Usage %	0.6
Processed FPS	7.1
FPS/CPU	0.112

As can be seen from table 5.5, using pixel difference algorithm decreases the processing time on DSP. However, CPU time of system is increases and overall performance of the system does not change (FPS/CPU is 0.112 as in table 5.4). This shows the effect of memory transfer operations on ARM + DSP applications. In our ARM + DSP application, DSP can not access the captured frame data directly. ARM core copy the image data from capture driver the to shared RAM memory. This result with a CPU usage in ARM core.

5.3 Video Playback Process on ARM Core

Smart camera application processes that must be implemented on platform can be more than one. These processes must be shared between ARM and DSP cores of system to design an efficient system. Because these cores can work asynchronous and they can run different processes at the same time. For example in an application, DSP core can process and encode the captured image in real time and ARM core can store them in a memory. At that time if the camera user wants to playback a video stream from memory, one of the cores of platform must decode encoded video streams at memory while the real time image processing duty of DSP continues. In that case this decoding process can be achieved by only ARM core. This section is performed on only ARM core to show the ability of platform for playback operation that can be used in a smart camera design. In this thesis video playback describes the decoding and retrieving frames from a file and displaying them on a display unit. Video playback can be used to play the recorded frames from an image sensor. Recording and playing important frames is one of the basic properties of smart camera design. Important frames are generally recorded by using encoders like MPEG2, MPEG4 and H.264. Now we will give brief information about MPEG and H.264 codec before describing the implemented video playback algorithm.

5.2.1 MPEG

MPEG is used as the abbreviation of the Moving Picture Experts Group that works on the development of the international standards for processing, compression, decompression and coded representation of moving pictures, audio and their combination [43]. Up to know this group have developed lots of MPEG standards namely MPEG1, MPEG2, MPEG4, MPEG7, MPEG21, MPEG-A, MPEG-B, MPEG-C, MPEG-D, MPEG-E, MPEG-M. However MPEG2 and MPEG4 standards have been most widely used MPEG standards among others. These media coding standards have been used for storage of video on CD and DVD and transportation of video over video satellite broadcast TV, HDTV [44].

Among others, MPEG2 is the predominant media coding standard for digital video equipments in the market. Present, this media coding standard is mostly used in

digital video broadcasting. By the development of the high definition video standards in digital video television broadcasting, coding efficiency (bits per image for constant quality) of the digital video compression algorithms has become a challenging issue to broadcast high resolution digital video. This situation result with an improvement in media coding standards. MPEG4 media coding standard completed in 2000 and this standard added more flexibility and functionality to MPEG2 standard. However, it did not improve the efficiency of MPEG2 media coding standard significantly. MPEG4 standard provides many features of MPEG1, MPEG2 and other related standards and it is still a developing standard. Especially the most interactive feature of this standard is its enabling one to use video functions almost like a Web page. MPEG4 standard bundle different types of audio and video contents into a file and break large scale files into sufficient small pieces to send over the network .

5.2.2 H.264

While MPEG team had been working on the MPEG4 standard, the International Telecommunications Union (ITU-T) started an independent Project to improve the compression performances of MPEG2 standard significantly. This standard committee designed a completely new media coding standard H.264 by using new ideas and innovative algorithms. The intention of the ITU-T committee was to provide good video quality at substantially lower bit rates than previous standards. Furthermore they wanted to use this standard for a wide variety of applications on a wide variety of networks and systems. At the end of the Project, H.264 Standard provided an advancement in picture quality and coding efficiency against MPEG4-2. The results show that that H.264 at least provides a 2X improvement over MPEG4.

Indeed, both MPEG4-2 and H.264 standards are image transformation algorithms that are based on forward/backward block motion compensated prediction with entropy coded transform coefficients. However, the methods selected for image transformation and entropy coding are differ at MPEG4-2 and H.264. These differences are summarized in Table 5.6. Video playback experiment is implemented to measure the video decoding capability of the platform. MPEG2, MPEG4 and H.264 standard encoded video files are decoded during the experiments.

Table 5.6. Comparison of H.264 Coding Algorithm to MPEG2/MPEG4 Algorithms (Adapted from [45])

Algorithm Characteristic	MPEG2/MPEG4	H.264
General	Motion compensated predictive, residual transformed, entropy coded	Same basic structure as MPEG
Intra Prediction	None	Multi-direction, Multi-pattern
Coded Image Types	I,B,P	I,B,P, SP
Transform	8x8 DCT	4x4 DCT-like Integer Transform
Motion Estimation Blocks	16x16	16x16, 8x8, 8x4, 4x4
Entropy Coding	Multiple VLC Tables	Arithmetic Coding and adaptive VLC Tables
Frame Distance for Prediction	+/- 1	Unlimited forward/backward
Fractional Motion Estimation	1/2 Pixel (MPEG2) 1/4 Pixel (MPEG4)	1/4 Pixel
Deblocking Filter	None	Dynamic edge filters

Figure 5.9 shows the block diagram of video playback process implemented on the ARM core of processor. Details of the code are given in Appendix A-iii .

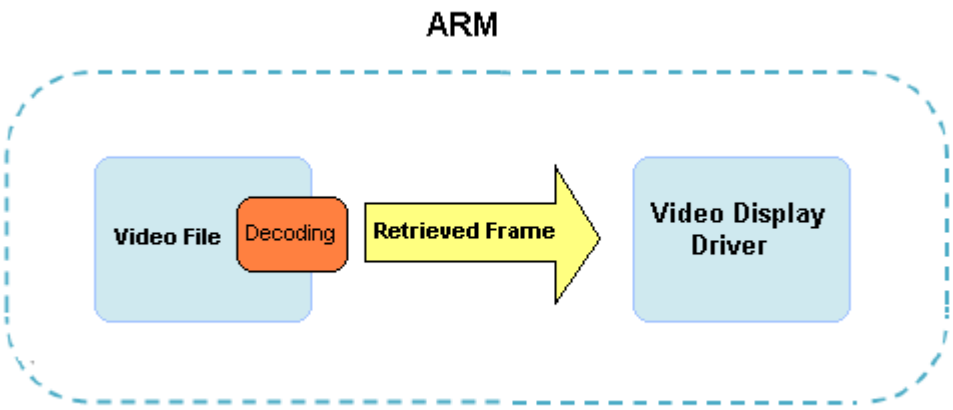


Figure 5.9 Block Diagram of Video Playback Process

As can be seen in Figure 5.9 we retrieve the frames from video file by decoding the video files with a compatible decoder. Decoder algorithms are implemented by installing necessary MPEG and H.264 packages like Ffmpeg, mpeg2dec to OS of host PC and target board. OpenCv functions are used to call the decoding algorithms that are implemented on installed packages and retrieve frames from files. Finally retrieved frames send to a display unit by using video display drivers. Table 5.7 shows the video playback performances of the host PC and target board for different resolution and different standard encoded video files. The result shows that processing capability of the host PC and target board does not change significantly for MPEG2 and MPEG4 standards. This is an expected result because of the similar compression efficiency of MPEG2 and MPEG4 standards. When we look at the processing capability for H.264 standard, we see a decrease in the number of processed pixel per second. This is also an expected result because a more complex algorithm is used in H.264 standard. However this complexity of H.264 algorithm provides an advance in compression efficiency while decreasing the processing power of the platform. To see the effect of input file resolution on the performance, two different resolution input file sets are used with 1280x720 and 720x480 resolution. The result shows that when resolution of input files decreases, as expected the number of processed frames increases and CPU load of processor decreases. However, the result also shows that the processing capability of host PC decreases significantly by decreasing input file resolution while processing capability of target board is not affected so much from input frame resolution. This situation can be explained by the increase in the memory usage for high resolution input file. However this is not sufficient to explain the decrease in the processing capability of host PC for low resolution images. The comparison of the processing capability of host PC with processing capability of target board shows that host PC video playback performance is about six times better than the target board for 1280x720 resolution input files and it is about four times better than the target board for 720x480 resolution images. This experiment shows that difference between performances of host PC and target board decrease for video playback application by decreasing resolution of input files. This experiment also shows that ARM core of the target board can playback a 720x480 resolution MPEG4 encoded video file at 10.2 fps by using % 29.6 of its CPU.

Up to now we have evaluated the performance of the algorithm developed by using the OpenCv, Ffmpeg and mpeg2dec libraries of the OS. And now we evaluate the video playback performance of the ARM core by using mplayer package of OS. Mplayer is an optimized video player developed for different OS. Figure 5.10 shows the block diagram of the video payback process implemented by using mplayer.

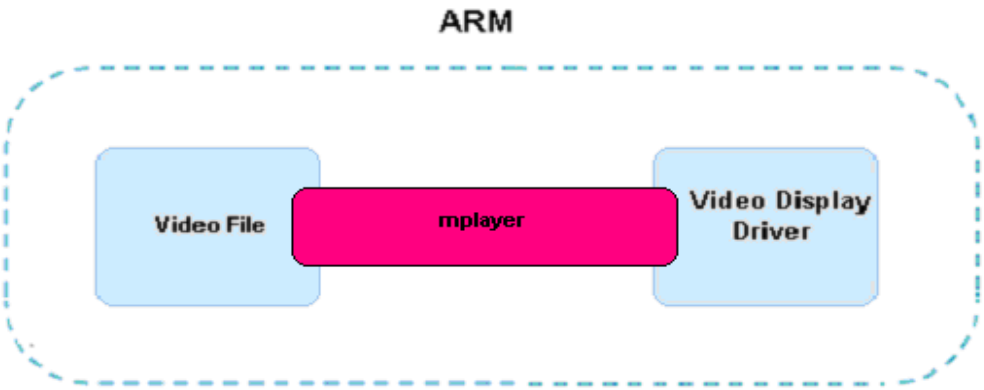


Figure 5.10 Block Diagram of Video Playback Process with mplayer

Table 5.10 shows the result of video playback experiment using mplayer. Mplayer normally plays the MPEG videos at 30 fps and it plays H.264 videos at 60 fps. However because of the processing power of the ARM core, mplayer of target board can not play H.264 videos at 60 fps while playing MPEG videos at 30 fps. From table 5.8 it can be seen that ARM core uses the %93 of its CPU to be able to playback the H.264 videos at 60 fps and its CPU is not sufficient to do that. Furthermore the comparison of the processing capabilities shows that host PC video playback performance is about four and half times better than target board for 1280x720 resolution input files and it is about two and half times better than target board for 720x480 resolution images. These results are better than the former experiment. So we can say that implementation of algorithm also effects the performance difference between host PC and target board besides the input file resolution. This experiment also shows that ARM core of the target board can playback a 720x480 resolution MPEG4 encoded video file at 30 fps by using % 29.4

of its CPU and this is three times better than the result of video playback experiment using OpenCv libraries.

Table 5.7. Video Playback

Experiment Setup	Host PC						Target Board					
Source Video Specifications	1280x720 (H.264)	1280x720 (MPEG4)	1280x720 (MPEG2)	720x480 (H.264)	720x480 (MPEG4)	720x480 (MPEG2)	1280x720 (H.264)	1280x720 (MPEG4)	1280x720 (MPEG2)	720x480 (H.264)	720x480 (MPEG4)	720x480 (MPEG2)
CPU Load %	58,3	50,9	52,2	49,7	45,7	44,8	53,7	51,0	36,8	31,3	29,6	29,2
Memory Usage %	6,1	5,1	4,4	3,8	3,6	3,4	6,3	5,1	4,2	3,7	3,2	2,9
Processed FPS	32,2	38,8	43,1	53,0	57,3	58,1	4,6	6,5	5,3	8,2	10,2	10,0
Resolution * (FPS/CPU)*100 (Mpixel/s)	48,6	66,9	72,7	35,1	41,3	42,7	7,6	11,2	12,7	8,6	11,3	11,3

Table 5.8. Video Playback with mplayer

Experiment Setup	Host PC						Target Board					
Source Video Specifications	1280x720 (H.264)	1280x720 (MPEG4)	1280x720 (MPEG2)	720x480 (H.264)	720x480 (MPEG4)	720x480 (MPEG2)	1280x720 (H.264)	1280x720 (MPEG4)	1280x720 (MPEG2)	720x480 (H.264)	720x480 (MPEG4)	720x480 (MPEG2)
CPU Load %	42,9	17,9	13,6	22,4	11,0	11,9	93,2	79,3	60,8	92,4	29,4	25,1
Memory Usage %	6,0	5,5	4,6	4,1	4,0	3,6	4,6	4,0	2,8	2,6	2,6	2,0
Processed FPS	60	30	30	60	30	30	20	30	30	54	30	30
Resolution* (FPS/CPU)*100 (Mpixel/s)	122,8	147,4	193,7	88,3	90,1	83,1	19.8	33,2	43,4	20.2	33,7	39,5

CHAPTER 6

SUMMARY and CONCLUSIONS

The aim of this thesis work is to implement a smart camera application on a COTS system. In this work, SoC architecture was selected for implementation because of its advantages. Beagleboard-XM platform that has a ARM + DSP SoC processor is chosen as COTS platform. During this thesis, the design steps of porting an embedded linux to ARM core of SoC processor to start-up the COTS platform and design steps of implementation an Algorithm that runs on both ARM + DSP cores in parallel are descried. Furthermore, with the experiments given in chapter 5, the real-time image processing performance of the Beagleboard-xM platform for the smart camera applications is evaluated.

For the overall system to work, first Beagleboard-xM board is started up by installing the necessary packages on the DM3730 processor and by making necessary configurations for a smart camera application. Arm core of the processor is started up by combining the files coming with the board with the output files of Narcissus image builder. DSP core of processor has started up by modifying TI DVSDK 4.01 package and porting it to the embedded OS implemented on ARM core of processor. Interprocessor communication between cores is provided by using DSPLink and configuring the shared RAM memory between ARM and DSP core.

Then the development environment is prepared. C/C++ compiler of target board OS is used to implement the developed algorithms on host PC's Eclipse platform. DVSDK 4.01 DSP compilers and example source files is used for the development and implementation of DSP core algorithms. DPS part of the application code is implemented by using the C6ACCEL component of DVSDK 4.01. Basic

applications of smart camera implementation are distributed over ARM and DSP cores of DM3730 processor.

Eventually, the performance of the Beagleboard-XM board for smart camera applications has been evaluated. Firstly to see the performance of the ARM core of DM3730 processor, basic smart camera applications is implemented on only ARM core. Video loopback performance of the ARM core is founded as 11 fps by using %73 of its CPU for 640x480 resolution image sensor. Video playback performance of the ARM core is found as 10.2 fps by using % 29.6 of its CPU for 720x480 resolution MPEG4 video file. This performance improved by using an optimized video player and performance of ARM core goes up 30 fps by using % 29.4 of its CPU for 720x480 resolution MPEG4 video file. This improvement shows that optimization of algorithms for ARM core enable at least three times better results. GMM background/foreground detection image processing capability of ARM core is measured as 0.8 fps by using %95 of its CPU for 640x480 resolution frames.

Furthermore, performance effects of NEON co-processor, DMA hardware accelerator and DSP core evaluated during experiments. By the help of NEON co-processor, image processing capability of ARM core increase at %50. DMA hardware accelerator increases the performance of memory transfer operations at %50. DSP core can not achieve an advantage for our application because the lack of direct access to capture and display drivers from DSP side. The data of captured frames are transferred to shared memory from capture driver by ARM core because of unsupported user allocation of capture buffers. Therefore, DSP core waits for ARM core to copy the necessary data to shared memory. This decreases the performance of ARM + DSP Application.

In this thesis, implementation steps to implement an embedded smart camera application on Beagleboard-XM platform are described in detailed. Beagleboard-XM is an inexpensive platform and can teach about open source embedded processing and DSP programming. Furthermore, its rich environment that includes ARM, NEON, DSP, OpenGL and QT can be used for implementation and evaluation of any algorithm. These capabilities of board can be used by hobbyists, academics, and

professionals who want to learn about Linux and DSP systems.

6.1 Future work and recommendations

The system performance can be improved in three directions.

The first direction is implementing the video background/foreground detection algorithm on DSP core of system by using the benefits of the VLIW architecture and single-instruction, multiple data (SIMD) instructions. State of art implementation of video background/foreground detection algorithms on TMS320C64/64x+ DSP is given with an application report of Texas Instrument in [55].

The second direction is to improve the performance of memory transfer operations by using user allocated buffers from device drivers and optimizing the data transfers with the help of L2 cache and Enhanced DMA architecture included in TMS320C64/64x+ DSP. An optimization process to decrease the time of data transfer is double buffering by splitting L2 cache two parts. This process is given with a TMS320C64x+ DSP Cache User's Guide in [61]. In this work a USB webcam is used because of the defined system specifications for smart camera and to add system flexibility for different cameras. However, Camera Image Sensor Processor (ISP) module of platform can be used to add the system raw output CMOS sensor cameras. This module enables the user allocation for captured images. So captured images can be taken directly in cache memory of DSP.

The third direction is to improve the performance of codes and reduce the development time by some useful libraries. In order to do that various royalty free software libraries that are provided by TI can be useful for future applications. Table 6.1 shows these libraries and number of functions inside these libraries.

Table 6.1. Number of Algorithms in Different Libraries Supported by TI

Algorithms	Now	2Q11	4Q11
Foundation Signal Processing Software Algorithms			
Digital Signal Processing	32	45	60
Image Processing	40	50	60
Floating Point Math	30	30	30
Fixed Point Math	32	32	32
Filter Package			128
Application Specific Software Algorithms			
Vision And Analytics Library (VLIB)		52	52
Open Source Computer Vision (OpenCV)		60	100
ProAudio: Audio Processing Library			20
Power and Energy			15
Total Supported Functions			
	134	**269**	**497**

These libraries can be grouped into three categories to serve different needs [46].

• Building Block Libraries: The libraries in this category are provided in completely in source. They can be used to understand the optimization of C6x devices. These libraries include reference C implementations and test bench for kernel. DSPLIB and IMGLIB are two key libraries that are included in this category for DM3730 processor.

• Specialized Application/Accelerator Libraries: This category provides ready to use algorithms with increased performance up to 10 times. This category also includes libraries that allow the user to develop the algorithms without specialized information about peripherals and accelerators of platform. VLIB is a key library that is included this category for DM3730 processor.

• Platform Libraries to Ease Development and Improve Quality: This category library provides floating point processing ability to the fixed point C6x devices. These libraries improve the accuracy and execution speed of algorithms. IQMath and fastRTS library are key libraries of these category for DM3730 processor.

During this project, all the experiments were executed on a BeagleBoard-xM Revision B. For future experiments to analyze the performance of TI's TMS320C6A8167 and TMS320C6A8168 processors can be also interesting. These processors both contain an ARM Cortex-A8 processor with up to a 1.5 GHz working frequency with NEON technology. The DSP is TI's TMS320C674x floating point VLIW DSP, fully compatible with the c64x+ DSP, used during this project.

REFERENCES

[1] L. Albani, Pietro Chiesa, Daniele Covi, Pedegani G., Alvise Sartori, and Monica Vatteroni. VISoc: A Smart Camera SoC. In Proceedings of the 28th European Conference on Solid-State Circuits (ESSCIRC), pages 367-370, September 2002.

[2] Anthony Rowe, Adam G. Goode, Dhiraj Goel, and Illah Nourbakhsh. CMUcam3: An Open Programmable Embedded Vision Sensor. Technical Report CMU-RI-TR-07-13, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2007.

[3] Anthony Rowe, Charles Rosenberg, and Illah Nourbakhsh. A Low Cost Embedded Color Vision System. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002.

[4] Anthony Rowe, Charles Rosenberg, and Illah Nourbakhsh. A Second Generation Low Cost Embedded Color Vision System. In Embedded Computer Vision Workshop (held in conjunction with CVPR), 2005.

[5] Vincent Jeanne, Francois-Xavier Jegaden, Richard Kleihorst, Alexander Danilin, and Ben Schueler. Real-time Face Detection on a Dual-Sensor Smart Camera using Smooth Edges Technique. In DSC 2006 (Workshop on Distributed Smart Cameras), 2006.

[6] Richard Kleihorst, Ben Schueler, Alexander Danilin, and Marc Heijligers. Smart Camera Mote with High Performance Vision System. In DSC 2006 (Workshop on Distributed Smart Cameras), 2006.

[7] Mohammad Rahimi, Rick Baer, Obimdinachi I. Iroezi, Juan C. Garcia, Jay Warrior, Deborah Estrin, and Mani Srivastava. Cyclops: In-situ Image Sensing and Interpretation in Wireless Sensor Networks. In SenSys '05: Proceedings of the 3rd

International Conference on Embedded Networked Sensor Systems, pages 192{204, New York, NY, USA, 2005. ACM.

[8] Michael Bramberger, Andreas Doblander, Milan Jovanovic, Arnold Maier, Bernhard Rinner, and Allan Tengg. Embedded Smart Cameras as Key Components in Reactive Sensor Systems. In COGnitive systems with Interactive Sensors (COGIS), 2006. 160 BIBLIOGRAPHY

[9] Michael Bramberger, Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Distributed Embedded Smart Cameras for Surveillance Applications. IEEE Computer, 392:68-75, February 2006.

[10] Clemens Arth, Christian Leistner, and Horst Bischof. TRICam - An Embedded Platform for Remote Traffic Surveillance. In Proceedings of the 2nd Workshop on Embedded Computer Vision , IEEE International Conference on Computer Vision and Pattern Recognition, page 125, Washington, DC, USA, 2006. IEEE Computer Society.

[11] Stephan Hengstler, Daniel Prashanth, Sufen Fong, and Hamid Aghajan. Mesheye: a Hybrid-Resolution Smart Camera Mote for Applications in Distributed Intelligent Surveillance. In Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN), pages 360-369, New York, NY, USA, 2007. ACM.

[12] Narcissus. Online image builder for Angström distribution. URL http://narcissus.angstrom-distribution.org. Last accessed: August 2011.

[13] Michael Bramberger, Roman P. Pugfelder, Arnold Maier, Bernhard Rinner, Bernhard Strobl, and Helmut Schwabach. A Smart Traffic Camera for Stationary Vehicle Detection. In Proceedings of the Workshop on Intelligent Solutions in Embedded Systems (WISES). Vienna University of Technology, June 2003.

[14] Markus Quaritsch, Markus Kreuzthaler, Bernhard Rinner, Horst Bischof, and Bernhard Strobl. Autonomous Multicamera Tracking on Embedded Smart Cameras. EURASIP Journal on Embedded Systems (Special Issue on Embedded Vision System), 2007:48-57, 2007.

[15] WayneWolf. High-Performance Embedded Computing: Architectures, Applications, and Methodologies. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[16] David A. Patterson. Reduced Instruction Set Computers. Communications of the ACM, 281:8-21, 1985.

[17] Andrew Sloss, Dominic Symes, and Chris Wright. ARM System Developer's Guide: Designing and Optimizing System Software. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[18] MIPS Technologies. http://www.mips.com. Last accessed: August 2011.

[19] Ahmed Nabil Belbachir. Smart Cameras. Breinigsville, PA, USA, December 2009

[20] Gotchev , A. Tikanmäki , A. Boev , K. Egiazarian , I. Pushkarov , N. Daskalov, Mobile 3DTV technology demonstrator based on OMAP 3430, Proceedings of the 16th international conference on Digital Signal Processing, p.759-764, July 05-07, 2009, Santorini, Greece

[21] A. Varfolomieiev, O. Antonyuk, O. Lysenko. Zheng, ” Camshift object tracking algorithm implementation on dm6437 EVM”, 4th European DSP Education and Research Conference, EDERC2010, p96-100, France, December 2010

[22] TI OMAP. URL http://focus.ti.com/general/docs/gencontent.tsp?contentId =46946. Last accessed: August 2011.

[23] R. Goering, .Platform-based Design: A Choice, not a Panacea,. EETIMES, September 2002; http://www.eetimes.com/story/OEG20020911S0061. Last accessed: August 2011.

[24] Texas Instruments, The Open Multimedia Applications Platform (OMAP). URL http://www.omap.com. Last accessed: August 2011.

[25] Aguilar-Ponce, R., Kumar, A., Tecpanecatl-Xihuitl, J. L., Bayoumi, M., & Radle, M. (2008). Automated Object Detection and Tracking for Intelligent Visual Surveillance Based on Sensor Network. In Sugumaran, V. (Ed.), Intelligent Information Technologies: Concepts, Methodologies, Tools, and Applications. (pp. 1263-1284). doi:10.4018/978-1-59904-941-0.ch071

[26] W. MacLean, "An evaluation of the suitability of FPGAs for embedded vision systems," in Proc. 2005 IEEE Comp. Soc. Conf. Comput. Vis. Pattern Recognit., Jun. 2005, pp. 131–131.

[27] BeagleBoard-xM System Reference Manual - Revision C.1.0, April 2010. Literature reference: BB_SRM_xM.

[28] Texas Instruments. DM3730, DM3725 Digital Media Processor, August 2010. Literature reference: SPRS685D.

[29]ARM. Cortex-A8processor. URL http://www.arm.com/products/processors/-cortex-a/cortex-a8.php. Last accessed: August 2011.

[30] ARM. NEON. URL http://www.arm.com/products/processors/technologies/-neon.php. Last access: August 2011.

[31] Texas Instruments. TMS320C64x/C64x+ DSP CPU and Instruction Set Reference Guide, July 2010. Literature Number: SPRU732J.

[32] Texas Instruments. TMS320 DSP/BIOS User's Guide, November 2002. Literature Number: SPRU423B.

[33] Texas Instruments. DSP/BIOS LINK User Guide - Version 1.65.00.02, March 2010. Reference: LNK 058 USR.

[34] Building Angström. URL http://www.angstrom-distribution.org/building-angstrom. Last access: August 2011.

[35]Debian. DebianWiki-ArmEabiPort. URL http://wiki.debian.org/ArmEabiPort-#Inanutshell. Last access: August 2011

[36] Andres Calderon and Nelson Castillo. Why ARM's EABI matters. March 2007. URL http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Why-ARMs-EABI-matters/. Last access: August 2011

[37] Jonathan Y. Stein. Digital Signal Processing: A Computer Science Perspective. John Wiley & Sons, 2000. ISBN: 0471295469.

[38] Texas Instruments. Cortex-A8 NEON architecture. URL http://processors.-wiki.ti.com/index.php/Cortex-A8_Neon_Architecture. Last access: August 2011

[39] ARM. Architecture and implementation of the ARM Cortex-A8 microprocessor. URL http://arch.eece.maine.edu/ece471/images/2/26/TigerWhite- paperFinal.pdf. Last access: August 2011

[40] ARM. ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition – Errata markup, October 2010. ARM DDI 0406B_errata_2010_Q3 (ID100710).

[41] Texas Instruments. TMS320C64x+ IQmath Library User's Guide, December 2008. Reference: SPRUGG9.

[42] TMS320C64x+ DSP Little-Endian DSP Library Programmer's Reference, March 2006. Literature Number: SPRUEB8B.

[43] Moving Picture Experts Group (MPEG). URL. http://www.scholarpedia.org/-article/Moving_Picture_Experts_Group_(MPEG). Last access: August 2011

[44] The Online Video Marketing Guide. URL. http://www.reelseo.com/encoding-formats-mpeg4-vs-h264/. Last access: August 2011

[45] NUNTIUS. H.264 - A New Technology for Video Compression. URL. http://www.nuntius.com/technology3.html. Last access: August 2011

[46] Software Libraries. URL. http://processors.wiki.ti.com/index.php/Software-_libraries Last access: August 2011

[47] Ambarella iOne Camera. URL. http://www.ambarella.com/news/23/74/-Ambarella-iOne-Camera-Aplications-Processors-Enables-a-New-Class-of-Android-Based-Smart-Cameras.html. Last access: August 2011

[48] BOA Vision System: Smart, Small, and Flexible. URL. www.teledynedalsa.-com/ipd/products/boa.aspx. Last access: August 2011

[49] Texas Instruments. TMS320C6000 DSP General-Purpose Input/Output (GPIO) Reference Guide, March 2004. Literature Number: SPRU584A.

[50] BELKIN. N150 Enhanced Wireless USB Network Adapter. URL. http://www.belkin.com/IWCatProductPage.process?Product_Id=492430#. Last access: August 2011

[51] Z.Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction", International Conference Pattern Recognition, Vol.2, pages: 28-31, 2004.

[52] Z.Zivkovic, F. van der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction", Pattern Recognition Letters, vol. 27, no. 7, pages 773-780, 2006.

[53] Beagleboard-XM by BeagleBoard.org. URL http://www.arm.com/community/partners/display_product/rw/ProductId/6089/ . Last access: August 2011

[54] DVSDK 4.01 for BeagleBoard xM. URL http://sourceforge.net/projects/dvsdkbbxm/ . Last access: November 2011

[55] Texas Instruments. Application Report: Video Background/Foreground Detection Implementation on TMS320C64/64x+ DSP, June 2007. Literature Number: SPRAAM6.

[56] Texas Instruments. UserGuideDisplayDrivers PSP 04_02_00_07. URL http://processors.wiki.ti.com/index.php/UserGuideDisplayDrivers_PSP_04.02.00.07 Last access: November 2011

[57] Video for Linux Two API Specification. URL http://linuxtv.org/downloads/legacy/video4linux/API/V4L2_API/spec-single/v4l2.html Last access: November 2011

[58] Texas Instruments. C6Accel Advanced Users Guide. URL http://processors.wiki.ti.com/index.php/C6Accel_Advanced_Users_Guide. Last access: November 2011

[59] Sagasu. Tool To Find Strings In A Set Of Files. URL : http://perso.b2b2c.ca/sarrazip/dev/sagasu.html

[60] Texas Instruments. Changing the DVEVM memory map. URL http://processors.wiki.ti.com/index.php/Changing_the_DVEVM_memory_map

[61] Texas Instruments. User's Guide: TMS320C64/64x+ DSP Cache User Guide, February 2009. Literature Number: SPRU862B.

APPENDIX A

Experimental Codes Developed for ARM and DSP cores

i. CFFT code developed for ARM and DSP cores

CFFT.h

```
#ifndef _CFFT_H_
#define _CFFT_H_
// Prevent C++ name mangling
#ifdef __cplusplus
extern "C" {
#endif
/*************************************************************
* Global Macro Declarations                                  *
*************************************************************/

#define MINPOW2 4

#define MAXPOW2 15

#define ITERATIONS 100

#ifndef M_PI
#define M_PI          3.14159265358979323846
#endif

#define pi_2 1.57079632679489661923F
#endif
#define abs2(v)  (v.r*v.r + v.i*v.i)
#define angle(v) atan2f(v.i,v.r)
#define cmult(c,a,b) c.r=a.r*b.r - a.i*b.i, \
                     c.i=a.r*b.i + a.i*b.r
#define csub(c,a,b)  c.r=a.r - b.r, \
                     c.i=a.i - b.i
#define cadd(c,a,b)  c.r=a.r + b.r, \
                     c.i=a.i + b.i

/*************************************************************
* Global Typedef Declarations                                *
*************************************************************/
typedef struct {
   float r;
   float i;
} complex;
/*************************************************************
* Global Variable Declarations                               *
```

```
************************************************************/
/************************************************************
* Global Function Declarations                             *
************************************************************/
extern void fft_init();
extern void fft_end();
extern void fft_exec(int N, complex* in);
/************************************************************
* End file                                                  *
************************************************************/
#ifdef __cplusplus
}
#endif
#endif //_CFFT_H_
```

CFFT.c

```
/* Code originally taken from the following URL:
    http://svn.arhuaco.org/svn/src/emqbit/tools/emqbit-bench/
*/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "cfft.h"
#include "common.h"

complex *tableW;
int *bndx;
int *ndx;

void fft_init (int N)
{
 int i, j;

 tableW = malloc ((N / 2) * sizeof (complex));
 bndx = malloc (N * sizeof (int));
 ndx = malloc ((N / 2) * sizeof (int));

 ndx[0] = 0;
 for (i = 1; i < N / 2; i = i * 2)
 {
  for (j = 0; j < i; j++)
  {
   ndx[j] *= 2;
   ndx[j + i] = ndx[j] + 1;
  }
 }

void fft_end ()
{
 free (ndx);
 free (bndx);
 free (tableW);
}

void fft_exec (int N, complex * in)
{
 unsigned int n = N;
```

```
  unsigned int a, b, i, j, k, r, s;
  complex w, p;

  for (i = 1; i < N; i = i * 2)
  {
   n = n >> 1;
   for (k = 0; k < i; k++)
   {
    w = tableW[k];

    r = 2 * n * k;
    s = n * (1 + 2 * k);

    for (j = 0; j < n; j++)
    {
     a = j + r;
     b = j + s;
     cmult (p, w, in[b]);      //6 flop
     csub (in[b], in[a], p);   //2 flop
     cadd (in[a], in[a], p);   //2 flop
    }
   }
  }
}
```

Main.c

```
/* Code originally taken from the following URL:
    http://svn.arhuaco.org/svn/src/emqbit/tools/emqbit-bench/
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <time.h>
#if defined(_TMS320C6X)
#elif defined(__GNUC__)
  #include <sys/time.h>
#endif

#include "cfft.h"
#include "common.h"

typedef unsigned long long timestamp_t;

static timestamp_t get_timestamp ()
{
#if defined(_TMS320C6X)
  // There is no gettimeofday in DSP RTS or DSP/BIOS
  return (timestamp_t) clock();
#elif defined(__GNUC__)
  struct timeval now;
  gettimeofday (&now, NULL);
  return  now.tv_usec + (timestamp_t)now.tv_sec * 1000000;
#endif
}

static complex *new_complex_vector(int size);
```

```
int main ()
{
  int i;
  int N, n;
  int nTimes;
  float secs;
  timestamp_t t0, t1;

  for (N = (1 << MINPOW2), n = 0; N < (1 << MAXPOW2); N = N << 1, n++)
  {
    complex *in = new_complex_vector(N);
    complex *out = new_complex_vector(N);

    fft_init (N);
    // Copy input data and do one FFT
    memcpy (out, in, (N) * sizeof (complex));
    fft_exec (N, out);

    nTimes = ITERATIONS;

    t0 = get_timestamp();

    for (i = 0; i < nTimes; i++)
    {
      memcpy (out, in, (N) * sizeof (complex));
      fft_exec (N, out);
    }

    t1 = get_timestamp();

    secs = (t1 - t0) / 1000000.0L;

    free (in);
    free (out);
    fft_end ();

    fprintf (stderr, "N=%d,nTimes=%d: %g s\n", N, nTimes, secs);
  }

  return 0;
}

static complex *new_complex_vector(int size)
{
  int i;

  complex *new;

  new = (complex *) malloc(sizeof(complex) * size);

  for(i = 0; i < size; ++i)
  {
    new[i].r = (float)rand()/(float)RAND_MAX - 0.5;
    new[i].i = (float)rand()/(float)RAND_MAX - 0.5;
  }

  return new;
}
```

ii. Video Loopback code developed for ARM core

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
#include <time.h>
#include <ctype.h>

 // A Simple Camera Capture Framework
 int main() {
   IplImage* frame = NULL;
   CvCapture* capture = NULL;
   int fps=0;
   int framenumber =0;
   int times=0;
   time_t old_time;
   time_t current_time;

   //capture = cvCaptureFromFile("/home/ArmWorkspace/workspace/out1.avi");
   capture = cvCaptureFromCAM(0);

   if ( !capture ) {
     fprintf( stderr, "ERROR: capture is NULL \n" );
     return -1;
   }
   // Create a window in which the captured images will be presented
   cvNamedWindow( "mywindow", CV_WINDOW_AUTOSIZE );
   // Show the image captured from the camera in the window and repeat
   old_time=time(NULL);
   while ( 1 ) {
     // Get one frame
     frame = cvQueryFrame( capture );

     if ( !frame ) {
       fprintf( stderr, "ERROR: frame is null...\n" );
       break;
     }

    framenumber++;
    current_time=time(NULL);
            times=current_time-old_time;
            old_time=current_time;

    if(times >= 1)
    {
                                fps=framenumber;
                                fprintf(stderr,"FPS = %d Resolution = %dx%d \n",fps,frame->width,frame-
>height);
                                framenumber=0;
    }

     cvShowImage( "mywindow", frame );
     // Do not release the frame!
     //If ESC key pressed, Key=0x10001B under OpenCV 0.9.7(linux version),
     //remove higher bits using AND operator
     if ( (cvWaitKey(10) & 255) == 27 ) break;
   }
   // Release the capture device housekeeping
   cvReleaseCapture( &capture );
```

```
  cvDestroyWindow( "mywindow" );
  return 0;
 }
```

iii. Video Playback code developed for ARM core

```
#include "cv.h"
#include "highgui.h"
#include <stdio.h>
#include <time.h>

 // A Simple Camera Capture Framework
 int main() {
  IplImage* frame = NULL;
  CvCapture* capture = NULL;
  int fps=0;
  int framenumber =0;
  int times=0;
  time_t old_time;
  time_t current_time;

  capture = cvCaptureFromAVI("/home/data/out2.avi");
  //capture = cvCaptureFromCAM( CV_CAP_ANY );

  if ( !capture ) {
   fprintf( stderr, "ERROR: capture is NULL \n" );
   getchar();
   return -1;
  }
  // Create a window in which the captured images will be presented
  cvNamedWindow( "mywindow", CV_WINDOW_AUTOSIZE );
  // Show the image captured from the camera in the window and repeat
  old_time=time(NULL);
  while ( 1 ) {
   // Get one frame
   frame = cvQueryFrame( capture );

   if ( !frame ) {
    fprintf( stderr, "ERROR: frame is null...\n" );
    break;
   }

   framenumber++;
   current_time=time(NULL);
          times=current_time-old_time;
          old_time=current_time;

   if(times >= 1)
   {
                          fps=framenumber;
                          fprintf(stderr,"FPS = %d Resolution = %dx%d \n",fps,frame->width,frame-
>height);
                          framenumber=0;
   }

    cvShowImage( "mywindow", frame );
    // Do not release the frame!
    //If ESC key pressed, Key=0x10001B under OpenCV 0.9.7(linux version),
    //remove higher bits using AND operator
```

```
    if ( (cvWaitKey(10) & 255) == 27 ) break;
  }
  // Release the capture device housekeeping
  cvReleaseCapture( &capture );
  cvDestroyWindow( "mywindow" );
  return 0;
}
```

iv. GMM Background Subraction code developed for ARM core

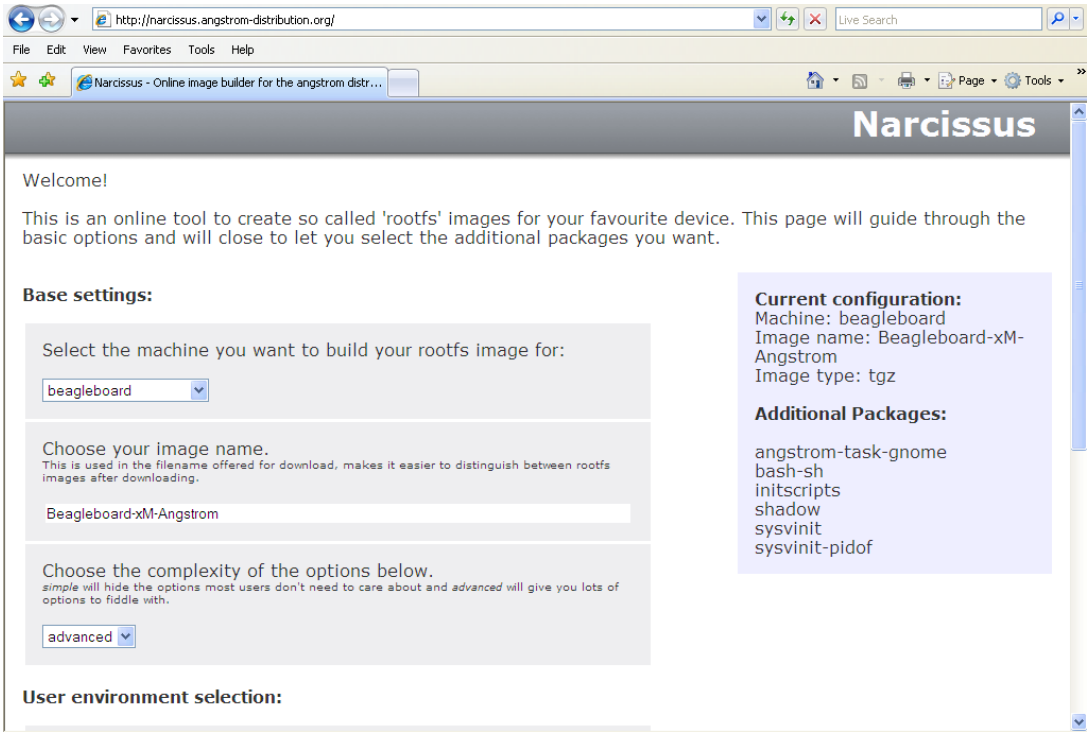
```
// Include header files

#include "cv.h"
#include "cvaux.h"
#include "highgui.h"
#include <ctype.h>
#include <stdio.h>
#include <time.h>
#include "CvPixelBackgroundGMM.h"

int main()
{

   /* Start capturing */

   CvCapture* capture = 0;
   int fps=0;
   int framenumber =0;
   int times=0;
   time_t old_time;
   time_t current_time;

   IplImage *frame = NULL, *frame_copy = NULL,*frame_morphological = NULL, *output =
NULL;

          // Some parameters for the algorithms

   CvPixelBackgroundGMM* pGMM=0;

   // Images to capture the frame from video or camera or from file

  //capture = cvCaptureFromFile("/home/data/out2.avi");
  //capture = cvCaptureFromFile("/home/ArmWorkspace/workspace/out2.avi");
  //int cfps = ( int )cvGetCaptureProperty( capture, CV_CAP_PROP_FPS );
  capture = cvCaptureFromCAM(0);

   if( !capture )
   {
      fprintf(stderr,"Could not initialize...\n");
      return -1;
   }

   // Create a new named window with title: result
   cvNamedWindow( "Original",1 );
   cvMoveWindow( "Original",0,100 );
   cvNamedWindow("Foreground",1);
   cvMoveWindow( "Foreground",704,100 );
```

```
   // Find if the capture is loaded successfully or not.
   // If loaded succesfully, then:
   if( capture )
   {
      old_time=time(NULL);

      frame = cvQueryFrame( capture );                    // for video file

          pGMM=cvCreatePixelBackgroundGMM(frame->width,frame->height);//reserve memory

          // modify some parameters
          pGMM->fAlphaT = 0.01f;

      // Capture from the camera.
      for(;;)
      {
         framenumber++;
         current_time=time(NULL);
          times=current_time-old_time;
          old_time=current_time;

         if(times >= 1)
         {
                              fps=framenumber;
                              fprintf(stderr,"FPS = %d Resolution = %dx%d \n",fps,frame->width,frame-
>height);
                              framenumber=0;
         }
         // Capture the frame and load it in IplImage
         frame = cvQueryFrame( capture );

         // If the frame does not exist, quit the loop
         if( !frame )
            break;

         // Allocate framecopy as the same size of the frame
         if( !frame_copy )
            frame_copy = cvCreateImage( cvSize(frame->width,frame->height), IPL_DEPTH_8U,
frame->nChannels );
         if( !output )
            output = cvCreateImage( cvSize(frame->width,frame->height), IPL_DEPTH_8U, 1 );
         // Check the origin of image. If top left, copy the image frame to frame_copy.
         if( frame->origin == IPL_ORIGIN_TL )
            cvCopy( frame, frame_copy, 0 );
         // Else flip and copy the image
         else
            cvFlip( frame, frame_copy, 0 );

         cvUpdatePixelBackgroundGMM(pGMM,  (unsigned char *)frame_copy->imageData,
(unsigned char *)output->imageData);
         // Call the function to detect and draw the shadows
       //   frame_morphological = cvCloneImage(output);
       //   cvDilate(output, frame_morphological);
       //   cvErode(frame_morphological, output);
         cvShowImage( "Original", frame_copy );
         cvShowImage( "Foreground", output);
         //system ("pause"); // MS-DOS pause command
```

```
            // Wait for a while before proceeding to the next frame
            if( cvWaitKey( 10 ) >= 0 )
                break;
        }

        // Release the images, and capture memory
        cvReleaseImage( &frame_copy );
        cvReleaseCapture( &capture );
        cvReleasePixelBackgroundGMM(&pGMM);

    }


    // Destroy the window previously created with filename: "result"
    cvDestroyWindow("Original");
    cvDestroyWindow("Foreground");

    return 0;
}
```

APPENDIX B

I- Settings of Narcissus Online Image Builder for System Angstrom OS



Narcissus online Image Builder setting consist of three main blocks namely Base Settings, User Environment Selection and Additional Package selection.

1. Base Settings:

Here selected setting for base setting will be given for creating the file system of Angstrom distrubition used in this thesis.

1.1 Select the machine you want to build your rootfs image for:

For this setting “beagleboard” is selected

1.2 Choose your image name:

For this setting “Beagleboard-xM-Angstrom” is selected

1.3 Choose the complexity of the options below :

For this setting “advanced” is selected

2. User Environment Selection:

Here selected settings for user environment selection will be given for creating the file system of Angstrom distrubition used in this thesis.

2.1 Console gives you a bare command line interface where you can install a GUI into later on. X11 will install an X-window environment and present you with a Desktop Environment option below. Opie is a qt/e 2.0 based environment for PDA style devices:

For this setting “X11” is selected

2.2 X11 Desktop Environments:

For this setting “GNOME” is selected

3. Additional packages selection:

Here selected settings for user environment selection will be given for creating the file system of Angstrom distrubition used in this thesis.

3.1 Additional X11 packages:

For this setting below packages are selected

- GNOME gedit
- GNOME MPlayer
- Midori web browser

3.2 Development packages:

For this setting below packages are selected

- Toolchain
- Native (on-target) u-boot mkimage
- Boost development headers and libraries
- Beagleboard GSoC 2010 XBMC build dependencies
- OpenCV headers and libs

3.3 Additional console packages:

For this setting below packages are selected

- All kernel modules
- Alsa utils

- Cpufrequtils
- FFmpeg
- Gstreamer
- Gstreamer GLES Plugin
- Htop
- Memtester
- MPlayer
- OpenCV

3.4 Network related packages:

For this setting below packages are selected

- Dropbear SSH server
- Lighttpd
- NetworkManager
- Wireless-tools

3.5 Java packages:

For this setting no packages are selected

3.6 Platform specific packages:

For this setting below packages are selected

- Bootloader Files (x-load/u-boot/scripts)
- AM/OMAP benchmarks / system info
- Matrix GUI for QT/embedded
- Matrix GUI for QT/X11
- Matrix TUI
- OMAP Display Sub System (DSS) Documentation
- FFmpeg based Media Player (omapfbplay) with Display Sub-System Support
- FFmpeg based Media Player (omapfbplay) with Distributed CodecEngine cupport
- PowerVR SGX drivers for OMAP3
- PowerVR SGX demos for framebuffer
- PowerVR SGX demos for X11
- TI texture streaming demo for X11
- TI texture streaming demo for framebuffer
- TI DSPLINK Example Applications
- TI Codec Engine Example Applications
- TI DMAI (Davinci/OMAP Multimedia Interface) Examples/Tests
- Texas Instruments Gstreamer plugins
- Julius demo for Texas Instruments
- TI SYSLINK Example Applications
- Original beagleboard demo
- Beagleboard validation GNOME image

APPENDIX C

U-Boot User Guide

Monitor Commands - Overview:
==========

```
go       - start application at address 'addr'
run      - run commands in an environment variable
bootm    - boot application image from memory
bootp    - boot image via network using BootP/TFTP protocol
tftpboot- boot image via network using TFTP protocol
                and env variables "ipaddr" and "serverip"
                (and eventually "gatewayip")
rarpboot- boot image via network using RARP/TFTP protocol
diskboot- boot from IDE devicebootd   - boot default, i.e., run 'bootcmd'
loads    - load S-Record file over serial line
loadb    - load binary file over serial line (kermit mode)
md       - memory display
mm       - memory modify (auto-incrementing)
nm       - memory modify (constant address)
mw       - memory write (fill)
cp       - memory copy
cmp      - memory compare
crc32    - checksum calculation
i2c      - I2C sub-system
sspi     - SPI utility commands
base     - print or set address offset
printenv- print environment variables
setenv   - set environment variables
saveenv - save environment variables to persistent storage
protect - enable or disable FLASH write protection
erase    - erase FLASH memory
flinfo   - print FLASH memory information
bdinfo   - print Board Info structure
iminfo   - print header information for application image
coninfo - print console devices and informations
ide      - IDE sub-system
loop     - infinite loop on address range
loopw    - infinite write loop on address range
mtest    - simple RAM test
icache   - enable or disable instruction cache
dcache   - enable or disable data cache
reset    - Perform RESET of the CPU
echo     - echo args to console
version - print monitor version
help     - print online help
?        - alias for 'help'
```

Monitor Commands - Detailed Description:
==

TODO.

For now: just type "help <command>".

Environment Variables:
======================

U-Boot supports user configuration using Environment Variables which can be made persistent by saving to Flash memory.

Environment Variables are set using "setenv", printed using "printenv", and saved to Flash using "saveenv". Using "setenv" without a value can be used to delete a variable from the environment. As long as you don't save the environment you are working with an in-memory copy. In case the Flash area containing the environment is erased by accident, a default environment is provided.

Some configuration options can be set using Environment Variables:

baudrate	- see CONFIG_BAUDRATE
bootdelay	- see CONFIG_BOOTDELAY
bootcmd	- see CONFIG_BOOTCOMMAND
bootargs	- Boot arguments when booting an RTOS image
bootfile	- Name of the image to load with TFTP
bootm_low	- Memory range available for image processing in the bootm command can be restricted. This variable is given as a hexadecimal number and defines lowest address allowed for use by the bootm command. See also "bootm_size" environment variable. Address defined by "bootm_low" is also the base of the initial memory mapping for the Linux kernel -- see the description of CONFIG_SYS_BOOTMAPSZ.
bootm_size	- Memory range available for image processing in the bootm command can be restricted. This variable is given as a hexadecimal number and defines the size of the region allowed for use by the bootm command. See also "bootm_low" environment variable.
updatefile	- Location of the software update file on a TFTP server, used by the automatic software update feature. Please refer to documentation in doc/README.update for more details.
autoload	- if set to "no" (any string beginning with 'n'), "bootp" will just load perform a lookup of the configuration from the BOOTP server, but not try to load any image using TFTP
autostart	- if set to "yes", an image loaded using the "bootp", "rarpboot", "tftpboot" or "diskboot" commands will be automatically started (by internally calling "bootm")

If set to "no", a standalone image passed to the
"bootm" command will be copied to the load address
(and eventually uncompressed), but NOT be started.
This can be used to load and uncompress arbitrary
data.

i2cfast - (PPC405GP|PPC405EP only)
if set to 'y' configures Linux I2C driver for fast
mode (400kHZ). This environment variable is used in
initialization code. So, for changes to be effective
it must be saved and board must be reset.

initrd_high - restrict positioning of initrd images:
If this variable is not set, initrd images will be
copied to the highest possible address in RAM; this
is usually what you want since it allows for
maximum initrd size. If for some reason you want to
make sure that the initrd image is loaded below the
CONFIG_SYS_BOOTMAPSZ limit, you can set this environment
variable to a value of "no" or "off" or "0".
Alternatively, you can set it to a maximum upper
address to use (U-Boot will still check that it
does not overwrite the U-Boot stack and data).

For instance, when you have a system with 16 MB
RAM, and want to reserve 4 MB from use by Linux,
you can do this by adding "mem=12M" to the value of
the "bootargs" variable. However, now you must make
sure that the initrd image is placed in the first
12 MB as well - this can be done with

setenv initrd_high 00c00000

If you set initrd_high to 0xFFFFFFFF, this is an
indication to U-Boot that all addresses are legal
for the Linux kernel, including addresses in flash
memory. In this case U-Boot will NOT COPY the
ramdisk at all. This may be useful to reduce the
boot time on your system, but requires that this
feature is supported by your Linux kernel.

ipaddr - IP address; needed for tftpboot command

loadaddr - Default load address for commands like "bootp",
"rarpboot", "tftpboot", "loadb" or "diskboot"

loads_echo - see CONFIG_LOADS_ECHO

serverip - TFTP server IP address; needed for tftpboot command

bootretry - see CONFIG_BOOT_RETRY_TIME

bootdelaykey- see CONFIG_AUTOBOOT_DELAY_STR

bootstopkey - see CONFIG_AUTOBOOT_STOP_STR

ethprime - When CONFIG_NET_MULTI is enabled controls which
interface is used first.

ethact - When CONFIG_NET_MULTI is enabled controls which
interface is currently active. For example you
can do the following

=> setenv ethact FEC ETHERNET
=> ping 192.168.0.1 # traffic sent on FEC ETHERNET
=> setenv ethact SCC ETHERNET
=> ping 10.0.0.1 # traffic sent on SCC ETHERNET

ethrotate - When set to "no" U-Boot does not go through all
available network interfaces.
It just stays at the currently selected interface.

netretry - When set to "no" each network operation will
either succeed or fail without retrying.
When set to "once" the network operation will
fail when all the available network interfaces
are tried once without success.
Useful on scripts which control the retry operation
themselves.

npe_ucode - set load address for the NPE microcode

tftpsrcport - If this is set, the value is used for TFTP's
UDP source port.

tftpdstport - If this is set, the value is used for TFTP's UDP
destination port instead of the Well Know Port 69.

vlan - When set to a value < 4095 the traffic over
Ethernet is encapsulated/received over 802.1q
VLAN tagged frames.

The following environment variables may be used and automatically updated by the network boot commands ("bootp" and "rarpboot"), depending the information provided by your boot server:

bootfile - see above
dnsip - IP address of your Domain Name Server
dnsip2 - IP address of your secondary Domain Name Server
gatewayip - IP address of the Gateway (Router) to use
hostname - Target hostname
ipaddr - see above
netmask - Subnet Mask
rootpath - Pathname of the root filesystem on the NFS server
serverip - see above

There are two special Environment Variables:

serial# - contains hardware identification information such
as type string and/or serial number
ethaddr- Ethernet address

These variables can be set only once (usually during manufacturing of the board). U-Boot refuses to delete or overwrite these variables once they have been set once.

Further special Environment Variables:

ver - Contains the U-Boot version string as printed
with the "version" command. This variable is

Image Formats:
==========

U-Boot is capable of booting (and performing other auxiliary operations on) images in two formats:

New uImage format (FIT)

Flexible and powerful format based on Flattened Image Tree -- FIT (similar to Flattened Device Tree). It allows the use of images with multiple components (several kernels, ramdisks, etc.), with contents protected by SHA1, MD5 or CRC32. More details are found in the doc/uImage.FIT directory.

Old uImage format

Old image format is based on binary files which can be basically anything, preceded by a special header; see the definitions in include/image.h for details; basically, the header defines the following image properties:

* Target Operating System (Provisions for OpenBSD, NetBSD, FreeBSD, 4.4BSD, Linux, SVR4, Esix, Solaris, Irix, SCO, Dell, NCR, VxWorks, LynxOS, pSOS, QNX, RTEMS, INTEGRITY;Currently supported: Linux, NetBSD, VxWorks, QNX, RTEMS, LynxOS, INTEGRITY).

* Target CPU Architecture (Provisions for Alpha, ARM, AVR32, Intel x86, IA64, MIPS, NIOS, PowerPC, IBM S390, SuperH, Sparc, Sparc 64 Bit; Currently supported: ARM, AVR32, Intel x86, MIPS, NIOS, PowerPC).

* Compression Type (uncompressed, gzip, bzip2)

* Load Address

* Entry Point

* Image Name

* Image Timestamp

The header is marked by a special Magic Number, and both the header and the data portions of the image are secured against corruption by CRC32 checksums.

Linux Support:
==========

Although U-Boot should support any OS or standalone application easily, the main focus has always been on Linux during the design of U-Boot.

U-Boot includes many features that so far have been part of some special "boot loader" code within the Linux kernel. Also, any "initrd" images to be used are no longer part of one big Linux image; instead, kernel and "initrd" are separate images. This implementation serves several purposes:

- the same features can be used for other OS or standalone applications (for instance: using compressed images to reduce the Flash memory footprint)

- it becomes much easier to port new Linux kernel versions because lots of low-level, hardware dependent stuff are done by U-Boot

- the same Linux kernel image can now be used with different "initrd" images; of course this also means that different kernel images can be run with the same "initrd". This makes testing easier (you don't have to build a new "zImage.initrd" Linux image when you just change a file in your "initrd"). Also, a field-upgrade of the software is easier now.

Boot Linux:

The "bootm" command is used to boot an application that is stored in memory (RAM or Flash). In case of a Linux kernel image, the contents of the "bootargs" environment variable is passed to the kernel as parameters. You can check and modify this variable using the "printenv" and "setenv" commands:

```
=> printenv bootargs
bootargs=root=/dev/ram

=> setenv bootargs root=/dev/nfs rw nfsroot=10.0.0.2:/LinuxPPC    nfsaddrs=10.0.0.99:10.0.0.2

=> printenv bootargs
bootargs=root=/dev/nfs rw nfsroot=10.0.0.2:/LinuxPPC nfsaddrs=10.0.0.99:10.0.0.2

=> bootm 40020000
## Booting Linux kernel at 40020000 ...
```

If you want to boot a Linux kernel with initial RAM disk, you pass
the memory addresses of both the kernel and the initrd image (PPBCOOT
format!) to the "bootm" command:

```
=> imi 40100000 40200000

=> bootm 40100000 40200000
```

More About U-Boot Image Types:

U-Boot supports the following image types:

"Standalone Programs" are directly runnable in the environment provided by U-Boot; it is expected that (if they behave well) you can continue to work in U-Boot after return from the Standalone Program.

"OS Kernel Images" are usually images of some Embedded OS which will take over control completely. Usually these programs will install their own set of exception handlers, device drivers, set up the MMU, etc. - this means, that you cannot expect to re-enter U-Boot except by resetting the CPU.

"RAMDisk Images" are more or less just data blocks, and their parameters (address, size) are passed to an OS kernel that is being started.

"Multi-File Images" contain several images, typically an OS (Linux) kernel image and one or more data images like RAMDisks. This construct is useful for instance when you want to boot over the network using BOOTP etc., where the boot server provides just a single image file, but you want to get for instance an OS kernel and a RAMDisk image. "Multi-File Images" start with a list of image sizes, each image size (in bytes) specified by an "uint32_t" in network byte order. This list is terminated by an "(uint32_t)0". Immediately after the terminating 0 follow the images, one by one, all aligned on "uint32_t" boundaries (size rounded up to a multiple of 4 bytes).

"Firmware Images" are binary images containing firmware (like U-Boot or FPGA images) which usually will be programmed to flash memory.

"Script files" are command sequences that will be executed by U-Boot's command interpreter; this feature is especially useful when you configure U-Boot to use a real shell (hush) as command interpreter.

APPENDIX D

A simple Make file

i. For ARM Core

arm:

g++ -I/usr/include/opencv -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"main.d" -MT"main.d" -o"main.o" "main.cpp"

g++ -I/usr/include/opencv -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"CvPixelBackgroundGMM.d" -MT"CvPixelBackgroundGMM.d" -o CvPixelBackgroundGMM.o CvPixelBackgroundGMM.cpp

g++ -L/usr/local/lib -o"BGFG_MoG_arm" ./main.o ./CvPixelBackgroundGMM.o `pkg-config --cflags --libs opencv`

neon:

g++ -I/usr/include/opencv -O0 -g3 -Wall -march=armv7-a -mtune=cortex-a8 -mfpu=neon -ftree-vectorize -mfloat-abi=softfp -c -fmessage-length=0 -MMD -MP -MF"main.d" -MT"main.d" -o"main.o" "main.cpp"

g++ -I/usr/include/opencv -O0 -g3 -Wall -march=armv7-a -mtune=cortex-a8 -mfpu=neon -ftree-vectorize -mfloat-abi=softfp -c -fmessage-length=0 -MMD -MP -MF"CvPixelBackgroundGMM.d" -MT"CvPixelBackgroundGMM.d" -o CvPixelBackgroundGMM.o CvPixelBackgroundGMM.cpp

g++ -L/usr/local/lib -g3 -Wall -march=armv7-a -mtune=cortex-a8 -mfpu=neon -ftree-vectorize -mfloat-abi=softfp -o"BGFG_MoG_neon" ./main.o ./CvPixelBackgroundGMM.o `pkg-config --cflags --libs opencv`

clean:

rm main.o
rm main.d
rm CvPixelBackgroundGMM.o
rm CvPixelBackgroundGMM.d
rm BGFG_MoG_arm
rm BGFG_MoG_neon

ii. For ARM+DSP Core

```
#  Copyright (C) 2010 Texas Instruments Incorporated
#    http://www.ti.com/

#  ----------------------------------------------------------------------------
#  Name of the ARM GCC cross compiler
```

```
#   ------------------------------------------------------------------------
ARM_TOOLCHAIN_PREFIX  ?= arm-none-linux-gnueabi-
ifdef ARM_TOOLCHAIN_PATH
ARM_CC := $(ARM_TOOLCHAIN_PATH)/bin/$(ARM_TOOLCHAIN_PREFIX)gcc
else
ARM_CC := $(ARM_TOOLCHAIN_PREFIX)gcc
endif

# Pick up any ARM compiler and linker flags from the environment
ARM_CFLAGS = $(CFLAGS)
ARM_CFLAGS += -std=gnu99 \
-Wdeclaration-after-statement -Wall -Wno-trigraphs \
-fno-strict-aliasing -fno-common -fno-omit-frame-pointer \
-c -O3
ARM_LDFLAGS = $(LDFLAGS)
ARM_LDFLAGS+=-lm


#   ------------------------------------------------------------------------
#   Name of the DSP compiler
#   TI C6RunApp Frontend (if path variable provided, use it, otherwise assume
#   the tools are in the path)
#   ------------------------------------------------------------------------
C6RUN_TOOLCHAIN_PREFIX=c6runapp-
ifdef C6RUN_TOOLCHAIN_PATH
C6RUN_CC := $(C6RUN_TOOLCHAIN_PATH)/bin/$(C6RUN_TOOLCHAIN_PREFIX)cc
else
C6RUN_CC := $(C6RUN_TOOLCHAIN_PREFIX)cc
endif

C6RUN_CFLAGS = -c -O3
C6RUN_LDFLAGS=


#   ------------------------------------------------------------------------
#   List of source files
#   ------------------------------------------------------------------------
SRCS := main_cfft.c main_bench.c cfft.c distance.c
ARM_OBJS := $(SRCS:%.c=gpp/%.o)
DSP_OBJS := $(SRCS:%.c=dsp/%.o)

#   ------------------------------------------------------------------------
#   Makefile targets
#   ------------------------------------------------------------------------
.PHONY : dsp gpp dsp_clean gpp_clean all clean

all: dsp gpp
clean: dsp_clean gpp_clean


gpp: gpp/.created $(ARM_OBJS)
	$(ARM_CC) $(ARM_LDFLAGS) -o bench_arm  gpp/main_bench.o gpp/distance.o
	$(ARM_CC) $(ARM_LDFLAGS) -o cfft_arm gpp/main_cfft.o gpp/cfft.o

gpp/%.o : %.c
	$(ARM_CC) $(ARM_CFLAGS) $(CINCLUDES) -o $@ $<

gpp/.created:
	@mkdir -p gpp
```

```
	@touch gpp/.created

gpp_clean:
	@rm -Rf bench_arm cfft_arm
	@rm -Rf gpp


dsp: dsp/.created $(DSP_OBJS)
	$(C6RUN_CC) $(C6RUN_LDFLAGS) -o bench_dsp dsp/main_bench.o dsp/distance.o
	$(C6RUN_CC) $(C6RUN_LDFLAGS) -o cfft_dsp dsp/main_cfft.o dsp/cfft.o

dsp/%.o : %.c
	$(C6RUN_CC) $(C6RUN_CFLAGS) $(CINCLUDES) -o $@ $<

dsp/.created:
	@mkdir -p dsp
	@touch dsp/.created

dsp_clean:
	@rm -Rf bench_dsp cfft_dsp
	@rm -Rf dsp
```

APPENDIX E

Sample NFS Files

i. Sample Host PC EXPORTS file

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).

# Example for NFSv2 and NFSv3:
# /srv/homes hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)

# Example for NFSv4:
# /srv/nfs4       gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)

#Share Below files with Beagleboard-XM and all NFS client (*means all client)

/root/DVSDK1 *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
/root/DVSDK2 *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
/home/ArmWorkspace *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
/root/install/targetfs *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
```

ii. Sample Target Board FSTAB file

```
# stock fstab - you probably want to override this with a machine specific one
rootfs          /               auto    defaults            1  1
proc            /proc           proc    defaults            0  0
devpts          /dev/pts        devpts  mode=0620,gid=5     0  0
usbfs           /proc/bus/usb   usbfs   defaults            0  0
tmpfs           /var/volatile   tmpfs   defaults            0  0
tmpfs           /dev/shm        tmpfs   mode=0777           0  0
tmpfs           /media/ram      tmpfs   defaults            0  0

# uncomment this if your device has a SD/MMC/Transflash slot
#/dev/mmcblk0p1  /media/card     auto    defaults,sync,noauto  0  0
#Connect 192.168.2.5 Host PC files
192.168.2.5:/root/DVSDK1 /mnt/DSP nfs rsize=8192,wsize=8192,timeo=14,intr
192.168.2.5:/home/ArmWorkspace /mnt/ARM nfs
rsize=8192,wsize=8192,timeo=14,intr
#144.122.167.204:/home/ArmWorkspace /mnt/ARM nfs
rsize=8192,wsize=8192,timeo=14r
#144.122.167.204:/root/DVSDK1 /mnt/DSP nfs
rsize=8192,wsize=8192,timeo=14,intr
```

APPENDIX F

Steps to port DVSDK 4.01 onto Beagleboard-xM rev B/C.

i. Steps to install Toolchain and DVSDK Software

a) Download CodeSourcery GCC toolchain from the link provided below

https://sourcery.mentor.com/sgpp/lite/arm/portal/package4573/public/arm-none-linux-gnueabi/arm-2009q1-203-arm-none-linux-gnueabi.bin

b) Execute the installer with .bin on the host host $./arm-2009q1-203-arm-none-linux-gnueabi.bin

c) Download TMS320DM3730 DVSDK 4.01 Installer software from TI website link provided below,

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/zdvsdk/DVSDK_4_00/4_01_00_09/index_FDS.html

d) After the download is complete, the first change the file permissions to executable

host $chmod 755 dvsdk_dm365-evm_4_00_00_22_setuplinux

e) Execute the installer on the host, follow the procedure on screen and complete the installation

host $./dvsdk_dm365-evm_4_00_00_22_setuplinux

ii. After installing DVSDK 4.01 download RevC-Source.tar file from our project and extract the files into psp folder of DVSDK from Sourceforge link provided below,

a) For Rev-B

https://sourceforge.net/projects/dvsdkbbxm/files/DVSDK-4.01%20for%20BeagleBoard/DVSDK%20for%20BB%20xM-B/Source%20files/RevB-Source.tar/download

b) For Rev-C

https://sourceforge.net/projects/dvsdkbbxm/files/DVSDK-4.01%20for%20BeagleBoard/DVSDK%20for%20BB%20xM-C/Source%20files/RevC-Source.tar/download

iii. Installed DVSDK-4.01 by default is for evm boards. In order to change it to for Beagleboard-xM we need to run the script.

a) Copy the script to the DVSDK folder

host $cd ${HOME}/ti-dvsdk_dm3730-evm_4_01_00_09

host $./setup-dvsdk-4-01-beaglexm.sh

Note:

i) Comment out these 4 lines if you have direct internet connection

```
# PROXYPORT="foo.com"
# PROXYPORT="80"
# export http_proxy="http://${PROXYHOST}:${PROXYPORT}"
```

ii) If you have connected to a network then add the proxy connection in the script

iv. After running the script, Build the DVSDK

host $make clean
host $make all

v. Insert SD/MMC card to your host and run df -h, to check the device node

host $df -h

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda2	61G	36G	23G	62%	/
none	939M	284K	938M	1%	/dev
none	943M	636K	942M	1%	/dev/shm
none	943M	112K	943M	1%	/var/run
none	943M	4.0K	943M	1%	/var/lock
none	943M	0	943M	0%	/lib/init/rw
/dev/sdb1	233G	105M	233G	1%	/media/New Volume

vi. Write to MMC card, by executing this script present in bin folder of DVSDK

host $sudo ./bin/mksdboot.sh --device /dev/sd? --sdk `pwd`
Note: It takes around 30 mins to finish portioning and writing to memory card.

vii. Unmount the memory card, insert the BB-xM and power up.

viii. Login with user name : root.

APPENDIX G

Adding the Running Gaussian Average Background/Foreground Detection DSP Application to Codec Server with C6ACCEL

..\DVSDK_4_01_00_09\c6accel_1_01_00_02\dsp\alg\include

C6Accel.h

```
#define RGA_DSP_FXN_ID                0x00000069
```

iC6Accel_ti.h

```
#define IMG_RGA_DSP_FXN_ID            0x01020069

/* Function call : void IMG_RGA_DSP(const unsigned char *in_data, unsigned char
*out_data, short cols, short rows)*/
typedef struct IMG_RGA_DSP_Params{
                                        unsigned int indata_InArrID1;
                                        unsigned int outdata_OutArrID1;
                                        unsigned int outdata_OutArrID2;
                                        unsigned int outdata_OutArrID3;
                                        int Col;
                                        int Row;
                                        }IMG_RGA_DSP_Params;
```

..\DVSDK_4_01_00_09\c6accel_1_01_00_02\soc\packages\ti\c6accel

iC6Accel_ti.h

```
/* Function call : void IMG_RGA_DSP(const unsigned char *in_data, unsigned char
*out_data, short cols, short rows)*/
typedef struct IMG_RGA_DSP_Params{
                                        unsigned int indata_InArrID1;
                                        unsigned int outdata_OutArrID1;
                                        unsigned int outdata_OutArrID2;
                                        unsigned int outdata_OutArrID3;
                                        int Col;
                                        int Row;
                                        }IMG_RGA_DSP_Params;
```

..\DVSDK_4_01_00_09\c6accel_1_01_00_02\soc\c6accelw

c6accelw.c

```
int C6ACCEL_IMG_RGA_DSP(C6accel_Handle hC6accel, const unsigned char
*in_data, unsigned char *out_data, short cols, short rows)
Arguments
* hC6accel C6accel Handle
* in_data[ ] Input image of size cols * rows.
* out_data[ ] Output image of size cols * (rows-2).
* cols Number of columns in the input image. Must be multiple of 2.
* rows Number of rows in the input image. cols * (rows-2) must be
multiple of 8.

Return value:  API returns status of the function call.
               ==1 Pass
               <0  Fail

Description This routine applies YUYV to ARGB convertion and Running
Gaussian Average background detection algoritm to the input
image and produces an output image which is 3/2 greater than the
input image.

*/
int C6accel_IMG_RGA_DSP
(   C6accel_Handle hC6accel,
    const unsigned char *restrict in,   /* Input image data   */
    unsigned char       *restrict meanData,  /* Output image data
*/
    unsigned char       *restrict varData,  /* Output image data  */
    unsigned char       *restrict out,  /* Output image data  */
    short cols, short rows              /* Image dimensions   */
)
{
    XDM1_BufDesc                inBufDesc;
    XDM1_BufDesc                outBufDesc;
    XDAS_Int32                  InArg_Buf_size;
    IC6Accel_InArgs             *CInArgs;
    UNIVERSAL_OutArgs           uniOutArgs;
    int status;
    /* Define pointer to function parameter structure */
    IMG_RGA_DSP_Params      *fp0;
    XDAS_Int8 *pAlloc;

    ACQUIRE_CODEC_ENGINE;

    /* Allocate the InArgs structure as it varies in size
    (Needs to be changed everytime we make a API call)*/
    InArg_Buf_size=  sizeof(Fxn_struct)+
                     sizeof(IMG_RGA_DSP_Params)+
                     sizeof(CInArgs->size)+
                     sizeof(CInArgs->Num_fxns);

    /* Request contiguous heap memory allocation for the extended
input structure */
    pAlloc = (XDAS_Int8 *)Memory_alloc(InArg_Buf_size,
&wrapperMemParams);
```

```
    CInArgs= (IC6Accel_InArgs *)pAlloc;

    /* Initialize .size fields for dummy input and output arguments
*/
    uniOutArgs.size = sizeof(uniOutArgs);

    /* Set up buffers to pass buffers in and out to alg  */
    inBufDesc.numBufs = 1;
    outBufDesc.numBufs = 3;

    /* Fill in input/output buffer descriptor parameters and manage
ARM cache*/
    /* See wrapper_c6accel_i.h for more details of operation
*/
    CACHE_WB_INV_INPUT_BUFFERS_AND_SETUP_FOR_C6ACCEL(in,0,2*cols *
rows*sizeof(char));
    CACHE_INV_OUTPUT_BUFFERS_AND_SETUP_FOR_C6ACCEL(meanData,0,cols *
rows*sizeof(char));
    CACHE_INV_OUTPUT_BUFFERS_AND_SETUP_FOR_C6ACCEL(varData,1,cols *
rows*sizeof(char));
    CACHE_INV_OUTPUT_BUFFERS_AND_SETUP_FOR_C6ACCEL(out,2,4*cols *
rows*sizeof(char));

     /* Initialize the extended InArgs structure */
    CInArgs->Num_fxns=1;
    CInArgs->size= InArg_Buf_size;

    /* Set function Id and parameter pointers for first function
call */
    CInArgs->fxn[0].FxnID= IMG_RGA_DSP_FXN_ID;
    CInArgs->fxn[0].Param_ptr_offset=sizeof(CInArgs-
>size)+sizeof(CInArgs->Num_fxns)+sizeof(Fxn_struct);

    /* Initialize pointers to function parameters */
    fp0 = (IMG_RGA_DSP_Params *)((XDAS_Int8*)CInArgs + CInArgs-
>fxn[0].Param_ptr_offset);

    /* Fill in the fields in the parameter structure */
    fp0->indata_InArrID1= INBUF0;
    fp0->outdata_OutArrID1= OUTBUF0;
    fp0->outdata_OutArrID2= OUTBUF1;
    fp0->outdata_OutArrID3= OUTBUF2;
    fp0->Col= cols;
    fp0->Row= rows;

    /* Call the actual algorithm */
    if (hC6accel->callType == ASYNC)
      {

       /* Update async structure */
       if (c6accelAsyncParams.asyncCallCount!=0){
            status = UNIVERSAL_EFAIL;
            printf("Async call failed as %d are still pending\n");
          }
       else{
           /* Context Saving */
           c6accelAsyncParams.asyncCallCount++;
           memcpy(&(c6accelAsyncParams.inBufs),&inBufDesc, sizeof
(XDM1_BufDesc));
```

```
            memcpy(&(c6accelAsyncParams.outBufs),
&outBufDesc,sizeof(XDM1_BufDesc));
            memcpy(&(c6accelAsyncParams.inArgs),
CInArgs,sizeof(UNIVERSAL_InArgs));

memcpy(&(c6accelAsyncParams.outArgs),&uniOutArgs,sizeof(UNIVERSAL_Ou
tArgs));
            c6accelAsyncParams.pBuf = pAlloc;
            c6accelAsyncParams.pBufSize = InArg_Buf_size;
            /* Asynchronous Call to the actual algorithm */
            status = UNIVERSAL_processAsync(hC6accel->hUni,
&inBufDesc, &outBufDesc, NULL,(UNIVERSAL_InArgs *)CInArgs,
&uniOutArgs);
            }
        }
      else{
        /* Synchronous Call to the actual algorithm */
        status = UNIVERSAL_process(hC6accel->hUni, &inBufDesc,
&outBufDesc, NULL,(UNIVERSAL_InArgs *)CInArgs, &uniOutArgs);

        /* Free the InArgs structure */
        Memory_free(pAlloc, InArg_Buf_size, &wrapperMemParams);
       }

      RELEASE_CODEC_ENGINE;

      return status;

}
/*
```

..\DVSDK_4_01_00_09\c6accel_1_01_00_02\dsp\alg\src

C6accel_ti_imglibFunctionCall.c

```
case (RGA_DSP_FXN_ID):{
    /* Unmarshal Parameters */
    IMG_RGA_DSP_Params *C6ACCEL_TI_IMG_RGA_DSP_paramPtr;
    C6ACCEL_TI_IMG_RGA_DSP_paramPtr= pFnArray;
    /*Parameter check*/
    if (((C6ACCEL_TI_IMG_RGA_DSP_paramPtr->indata_InArrID1)>INBUF15)|
      ((C6ACCEL_TI_IMG_RGA_DSP_paramPtr-
>outdata_OutArrID1)>OUTBUF15)|
      ((C6ACCEL_TI_IMG_RGA_DSP_paramPtr-
>outdata_OutArrID2)>OUTBUF15)|
      ((C6ACCEL_TI_IMG_RGA_DSP_paramPtr-
>outdata_OutArrID3)>OUTBUF15)|
      ((C6ACCEL_TI_IMG_RGA_DSP_paramPtr->Col)%2 != 0)|
      (((C6ACCEL_TI_IMG_RGA_DSP_paramPtr- >Col)*
          (C6ACCEL_TI_IMG_RGA_DSP_paramPtr->Row))%8 !=0)){
            return(IUNIVERSAL_EPARAMFAIL);
      }

    else
```

```
    /* Call underlying kernel */
    IMG_RGA_DSP((const unsigned char *)inBufs-
>descs[C6ACCEL_TI_IMG_RGA_DSP_paramPtr->indata_InArrID1].buf,
    (unsigned char *)outBufs->descs[C6ACCEL_TI_IMG_RGA_DSP_paramPtr-
>outdata_OutArrID1].buf,
    (unsigned char *)outBufs->descs[C6ACCEL_TI_IMG_RGA_DSP_paramPtr-
>outdata_OutArrID2].buf,
    (unsigned char *)outBufs->descs[C6ACCEL_TI_IMG_RGA_DSP_paramPtr-
>outdata_OutArrID3].buf,
    C6ACCEL_TI_IMG_RGA_DSP_paramPtr->Col,
    C6ACCEL_TI_IMG_RGA_DSP_paramPtr->Row);
      }
```

RGA_DSP.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "fastrts_i.h"
#include <xdc/std.h>

void IMG_RGA_DSP
(
  const unsigned char *restrict in,   /* Input image data   */
  unsigned char       *restrict meanData,  /* Output image data  */
  unsigned char       *restrict varData,  /* Output image data  */
  unsigned char       *restrict out,  /* Output image data  */
  short cols, short rows
)
{
  Int32 i,numPixels;                  /* Loop counter                  */
  Int32 y0=0, y1=0;                /* Individual Y components         */
  Int32 cb=0, cr=0;               /* Color difference components     */
  Int32 y0t,y1t;               /* Temporary Y values              */
  Int32 rt, gt, bt;            /* Temporary RGB values            */
  Int32 r0, g0, b0;            /* Individual RGB components       */
  Int32 r1, g1, b1;            /* Individual RGB components       */
  Int32 r0t,g0t,b0t;           /* Truncated RGB components        */
  Int32 r1t,g1t,b1t;           /* Truncated RGB components        */
  Int32 r0s=0,g0s=0,b0s=0;           /* Saturated RGB components        */
  Int32 r1s=0,g1s=0,b1s=0;           /* Saturated RGB components        */
  Int16 luma = 0x2543;        /* Luma scaling coefficient.       */
  Int16 r_cr = 0x3313;        /* Cr's contribution to Red.       */
  Int16 g_cb = -0x0C8A;        /* Cb's contribution to Green.     */
  Int16 g_cr = -0x1A04;        /* Cr's contribution to Green.     */
  Int16 b_cb = 0x408D;        /* Cb's contribution to Blue.      */
  UInt32 calc2;
  UInt16 THR,THR_calc;            /* unPacked 32 bit ARGB pixel data           */
```

```
  double alfa,beta,calc1,calc3,calc4;
  UInt8 v,m, THR_k,f;
//   Int8 *meanData,*varData;


  /* ---------------------------------------------------------------- */
  /*  Iterate for numPixels/2 iters, since we process pixels in pairs.   */
  /* -----
   * ---------------------------------------------------------------- */
  numPixels=cols*rows;
  THR_k=2;
  alfa= 0.0625;
  beta= subsp_i(1,alfa);
  i = numPixels >> 1;

  while (i-->0)
  {
     /* ---------------------------------------------------------------- */
     /*  Read in YCbCr data from the separate data planes.              */
     /*                                                                 */
     /*  The Cb and Cr channels come in biased upwards by 128, so       */
     /*  subtract the bias here before performing the multiplies for    */
     /*  the color space conversion itself.  Also handle Y's upward     */
     /*  bias of 16 here.                                               */
     /* ---------------------------------------------------------------- */

     y0 = *in++;
     cb = *in++;
     y1 = *in++;
     cr = *in++;

     m=*meanData++;
     v=*varData++;

     f=y0;

     calc1 = mpysp_i(alfa, f) + mpysp_i(beta, m);
     calc2 =(f-m)*(f-m);
     calc3 = mpysp_i(alfa, calc2);
     calc4 = mpysp_i(beta, v);
     calc4 = addsp_i(calc3, calc4);

     v = spuint_i(sqrtsp_i(calc2));
     m = spuint_i(calc1);
     *varData = (UInt8)v;
     *meanData = (UInt8)m;

     THR =THR_k*v;
```

```
        if(v<1)
           goto cont1;

        if(f < m)
           THR_calc = ((m-f) / v);
        else
           THR_calc = ((f-m) / v);

        if (THR_calc < THR)
                              y0=0;


cont1:
                      m=*meanData++;
                      v=*varData++;

        f=y1;

        calc1 = mpysp_i(alfa, f) + mpysp_i(beta, m);
        calc2 =(f-m)*(f-m);
        calc3 = mpysp_i(alfa, calc2);
        calc4 = mpysp_i(beta, v);
        calc4 = addsp_i(calc3, calc4);

        v = spuint_i(sqrtsp_i(calc2));
        m = spuint_i(calc1);
        *varData = (UInt8)v;
        *meanData = (UInt8)m;

        THR =THR_k*v;

        if(v<1)
           goto cont1;

        if(f < m)
           THR_calc = ((m-f) / v);
        else
           THR_calc = ((f-m) / v);

        if (THR_calc < THR)
                              y1=0;

cont2:
        y0-=16;
        cb=0;
        y1-=16;
        cr=0;
```

```
    /*
================================================================
==== */
    /*  Convert YCrCb data to RGB format using the following matrix:    */
    /*                                                                  */
    /*     [ coeff[0] 0.0000   coeff[1] ]   [ Y' -  16 ]     [ R']     */
    /*     [ coeff[0] coeff[2] coeff[3] ] * [ Cb - 128 ]  =  [ G']     */
    /*     [ coeff[0] coeff[4] 0.0000   ]   [ Cr - 128 ]     [ B']     */
    /*                                                                  */
    /*  We use signed Q13 coefficients for the coefficients to make     */
    /*  good use of our 16-bit multiplier.  Although a larger Q-point   */
    /*  may be used with unsigned coefficients, signed coefficients     */
    /*  add a bit of flexibility to the kernel without significant      */
    /*  loss of precision.                                              */
    /*
================================================================
==== */

    /* ---------------------------------------------------------------- */
    /*  Calculate chroma channel's contribution to RGB.                 */
    /* ---------------------------------------------------------------- */
    rt  = r_cr * (Int16)cr;
    gt  = g_cb * (Int16)cb + g_cr * (Int16)cr;
    bt  = b_cb * (Int16)cb;

    /* ---------------------------------------------------------------- */
    /*  Calculate intermediate luma values.  Include bias of 16 here.   */
    /* ---------------------------------------------------------------- */
    y0t = luma * (Int16)y0;
    y1t = luma * (Int16)y1;

    /* ---------------------------------------------------------------- */
    /*  Mix luma, chroma channels.                                      */
    /* ---------------------------------------------------------------- */
    r0  = y0t + rt; r1 = y1t + rt;
    g0  = y0t + gt; g1 = y1t + gt;
    b0  = y0t + bt; b1 = y1t + bt;

    /*
================================================================
==== */
    /*  At this point in the calculation, the RGB components are        */
    /*  nominally in the format below.  If the color is outside the     */
    /*  our RGB gamut, some of the sign bits may be non-zero,           */
    /*  triggering saturation.                                          */
    /*                                                                  */
    /*              3    2 2       1 1                                 */
    /*              1    1 0       3 2       0                         */
    /*             [ SIGN  | COLOR   | FRACTION ]                       */
```

```
    /*                                                    */
    /*  This gives us an 8-bit range for each of the R, G, and B        */
    /*  components.  (The transform matrix is designed to transform     */
    /*  8-bit Y/C values into 8-bit R,G,B values.)  To get our final    */
    /*  5:6:5 result, we "divide" our R, G and B components by 4, 8,    */
    /*  and 4, respectively, by reinterpreting the numbers in the       */
    /*  format below:                                                   */
    /*                                                    */
    /*        Red,   3    2 2    1 1                          */
    /*        Blue   1    1 0    6 5          0               */
    /*             [ SIGN  | COLOR | FRACTION    ]              */
    /*                                                    */
    /*             3    2 2     1 1                          */
    /*        Green  1    1 0     5 4          0               */
    /*             [ SIGN  | COLOR  | FRACTION   ]              */
    /*                                                    */
    /*  "Divide" is in quotation marks because this step requires no    */
    /*  actual work.  The code merely treats the numbers as having a    */
    /*  different Q-point.                                              */
    /*
================================================================
==== */

    /* ---------------------------------------------------------------- */
    /*  Shift away the fractional portion, and then saturate to the     */
    /*  RGB 5:6:5 gamut.                                                */
    /* ---------------------------------------------------------------- */
    r0t = r0 >> 13;
    g0t = g0 >> 13;
    b0t = b0 >> 13;
    r1t = r1 >> 13;
    g1t = g1 >> 13;
    b1t = b1 >> 13;

    r0s = r0t < 0 ? 0 : r0t > 255 ? 255 : r0t;
    g0s = g0t < 0 ? 0 : g0t > 255 ? 255 : g0t;
    b0s = b0t < 0 ? 0 : b0t > 255 ? 255 : b0t;
    r1s = r1t < 0 ? 0 : r1t > 255 ? 255 : r1t;
    g1s = g1t < 0 ? 0 : g1t > 255 ? 255 : g1t;
    b1s = b1t < 0 ? 0 : b1t > 255 ? 255 : b1t;

    /* ---------------------------------------------------------------- */
    /*  Store resulting pixels to memory.                               */
    /* ---------------------------------------------------------------- */

    *out++=(unsigned char)b0s;
    *out++=(unsigned char)g0s;
    *out++=(unsigned char)r0s;
    *out++=(unsigned char)0x00;
```

```
        *out++=(unsigned char)b1s;
        *out++=(unsigned char)g1s;
        *out++=(unsigned char)r1s;
        *out++=(unsigned char)0x00;

    }

    return;
}
```

APPENDIX H

Configuration of Server Map Files of RGA_DSP Application

Memmap.tci File

```
/*
 *  ======== memmap.tci ========
 *
 * Setup platform-specific memory map:
 *
 */

var mem_ext = [

{
    comment:    "DDRALGHEAP: off-chip memory for dynamic algmem
allocation",
    name:       "DDRALGHEAP",
    base:       0x85900000,
    len:        0x02000000,
    space:      "code/data"
},
{
    comment:    "DDR2: off-chip memory for application code and
data",
    name:       "DDR2",
    base:       0x87900000,
    len:        0x00600000,
    space:      "code/data"
},
{
    comment:    "DSPLINK: off-chip memory reserved for DSPLINK code
and data",
    name:       "DSPLINKMEM",
    base:       0x87F01000,
    len:        0x000FF000,
    space:      "code/data"
},
{
    comment:    "RESET_VECTOR: off-chip memory for the reset vector
table",
    name:       "RESET_VECTOR",
    base:       0x87F00000,
    len:        0x00001000,
    space:      "code/data"
},

{
    comment:    "L4CORE: L4-Core Interconnect Address Space",
    name:       "L4CORE",
    base:       0x48000000,
    len:        0x01000000,
```

```
    space:      "data"
},
{
    comment:    "L4PER: L4-Peripheral Interconnect Address Space",
    name:       "L4PER",
    base:       0x49000000,
    len:        0x00100000,
    space:      "data"
},
];
```

Server.tcf File

```
/*
 *  ======== server.tcf ========
 */

var platform = environment["config.platform"];
print("platform   = " + platform);

utils.importFile('./memmap.tci');

    var device_regs = {
        l1PMode: "16k",
        l1DMode: "80k",
        l2Mode: "64k",
        l1DHeapSize: 0
    };

    var params = {
        clockRate: 360,
        catalogName: "ti.catalog.c6000",
        deviceName: "3530",
        regs: device_regs,
        mem: mem_ext
    };

/* Now customize the generic platform with parameters specified
above. */
utils.loadPlatform("ti.platforms.generic", params);

/* Enable heaps and tasks */
bios.enableMemoryHeaps(prog);
bios.enableTskManager(prog);

/* Create heaps in memory segments that are to have heap */
bios.DDR2.createHeap = true;
bios.DDR2.heapSize   = 0x20000; // 128K

bios.DDRALGHEAP.createHeap = true;
bios.DDRALGHEAP.heapSize   = bios.DDRALGHEAP.len;

/* L1DSRAM */
bios.L1DSRAM.createHeap       = true;
bios.L1DSRAM.enableHeapLabel  = true;
bios.L1DSRAM["heapLabel"]     = prog.extern("L1DHEAP");
bios.L1DSRAM.heapSize         = bios.L1DSRAM.len;

/* Enable power management, whilst ensuring DSP CPU load reporting
accuracy */
```

```
bios.PWRM.ENABLE = true;
bios.PWRM.IDLECPU = true;
bios.PWRM.LOADENABLE = true;
bios.PWRM.USECLKPRD = true;
bios.PWRM.NUMSLOTS = 10 + 1;
bios.PWRM.CLKTICKSPERSLOT = 50;

/* GBL */
bios.GBL.ENABLEALLTRC    = false;
bios.GBL.PROCID          = 0;

/* set MAR register to cache external memory 0x80000000-0x8FFFFFFF
*/
bios.GBL.C64PLUSCONFIGURE   = true;
bios.GBL.C64PLUSMAR128to159 = 0x0fffffff;

/* MEM */
bios.MEM.STACKSIZE = 0x1000;

/* Global Settings */
bios.MEM.ARGSSIZE = 256;

/* Enable MSGQ and POOL Managers */
bios.MSGQ.ENABLEMSGQ = true;
bios.POOL.ENABLEPOOL = true;

/* Set all code and data sections to use DDR2 */
bios.setMemCodeSections(prog, bios.DDR2);
bios.setMemDataNoHeapSections(prog, bios.DDR2);
bios.setMemDataHeapSections(prog, bios.DDR2);

/* MEM : Global */
bios.MEM.BIOSOBJSEG = bios.DDR2;
bios.MEM.MALLOCSEG  = bios.DDR2;

/* TSK : Global */
bios.TSK.STACKSEG = bios.DDR2;
bios.TSK.STACKSIZE = 0x1000;
bios.TSK.instance("TSK_idle").stackSize = 0x1000;

/* Generate configuration files... */
if (config.hasReportedError == false) {
    prog.gen();
}
/*
 *  @(#) ti.sdo.ce.wizards.genserver; 1, 0, 0,81; 9-20-2010
16:43:30; /db/atree/library/trees/ce/ce-r09x/src/ xlibrary

 */
```