||SHREE GAJANAN PRASANNA||

# DUCATI SYSTEM ANALYSER FOR OPTIMIZING PEER TO PEER VIDEO CHAT IN OMAP ENABLED HIGH END MULTIMEDIA MOBILE PHONES

**Dissertation**

Submitted in partial fulfilment of the requirements

for the degree of

Master of Technology (Software Engineering)

by

**Prasad Y. Jondhale**

Roll No: 07/SWE/2K10

Under the guidance of

**Ms Divyashikha**

Assistant Professor,

Department of Software Engineering, DTU

**Mr Karthik Ramanan**

Sr. Software Design Engineer,

Texas Instruments, India (Bangalore)

**Department of Computer Engineering,**
**Delhi Technological University, Delhi**
**Delhi-110042**

**Dedicated to**
**my Family, Friends, Teacher’s**

# Dissertation Approval Sheet

This is to certify that the dissertation titled
**Ducati System Analyser for Peer To Peer Video Chat in OMAP Enabled High End Multimedia Mobile Phones**

By
Prasad Y. Jondhale
(07/SWE/2k10)

is approved for the degree of Master of Technology (Software Engineering)

Prof. Ms. Divyashikha Sethia
(Guide)

Internal Examiner

External Examiner

Chairperson

Date:

## ABSTRACT/SYNOPSIS

OMAP developed by Texas Instruments is a category of proprietary system on chips (SoCs) for portable and mobile multimedia applications. OMAP4 is the fourth generation SOC that is used in many of today’s high performance and low power multimedia devices- which include smart phones, tablets, video conferencing solutions and other consumer products. OMAP4 enables true High Definition multimedia (1080p), very high performance graphics and yet offers a low power advantage over other SoCs. To name a few use cases - today's devices are capable of playing 1080p content, allow streaming of high definition content, capture images with high resolution cameras greater than 8MPixel, and also support high definition video chat in peer to peer networks up to resolutions of 720p. The advent of all these multimedia services has given rise to serious challenges in terms of battery usage (Power) and performance. The most complex of the above all use cases is the video chat, in which device is involved in performing a video decode of the stream coming in from the remote, and simultaneously the camera is operational on the near end and it is performing a video encode to send the compressed bit stream to the remote end. To build an optimized solution, we need to analyse several parameters in the system and continuously work

towards understanding the intricacies. With a thorough study of the current design of the Video Chat application, I developed and designed the Ducati System Analyser which will help developers to actually see trade-offs between all these parameters so they can configure chat application and find out more optimal configurations.

# Acknowledgements

I take this opportunity to express a deep sense of gratitude towards my guide Prof**. Ms. Divyashikha Sethia** and my mentor at Texas Instruments, India **Mr. Karthik Ramanan**, for providing excellent guidance, encouragement and inspiration throughout the project work. Without invaluable guidance of them, this work would never have been a successful one.

Also my special thanks to Prof Daya Gupta, Head, Department of software Engineering, DTU for making this opportunity feasible for project being carried out at TI.

Last but not least, I would like to thank my **family** and **friends** specially **Anuj Kumar & Sagar Jondhale**, who have been a source of encouragement and inspiration throughout the duration of the project.

.

**Prasad Y. Jondhale**
DTU, Delhi
July, 2012

# Contents

# List of Figures, Tables, Illustrations, Symbols and Acronyms

## *List of Figures, Tables*

## Acronyms

API- Application programming Interface

BIOS-Basic I/O System.

CPU-Central Processing Unit

DOMX-Distributed OpenMAX

DRC-Ducati Repo Clean-up

DSA-Ducati System Analyser

DSP-Digital Signal Processor.

GPU-Graphical Processing Unit.

HD- High Definition.

HLOS- High Level Operating System.

IL- Interface layer.

ISS- Imaging SubSystem.

IVA-HD- Image and video accelerator high-definition

LTRP - Long Term Reference Pictures

MIMD- Multiple Instruction Multiple Data.

MP- Megapixel.

MPEG- Motion Pictures Expert Group.

MPU- MicroProcessor Unit

OMAP-Open Multimedia Application Processor.

OMX- OpenMAX

OS- Operating system.

P2P-Peer to Peer

PDA- Personal Digital Assistant.

QoS-quality of service

SIMCOP- Still Image CoProcessor

SIMD- Single Instruction Multiple Data.

SMP- Symmetric Multiprocessing.

SoC- system on chips.

TI- Texas Instruments.

TIITC- Texas Instruments India Technical Conference.

TILER- Tiling and Isometric Lightweight Engine for Rotation

UML –Unified Modelling Language.

WLAN- Wireless Local Area Network.

# Chapter 1: INTRODUCTION

## 1.1 Overview

Mobile Industry is going through a rapid evolution with addition of different multimedia services. Mobile phones is multimedia enabled which runs operating systems that allowed true multitasking. These devices are designed for true High Definition multimedia, to name a few, today's mobile devices is capable of playing 1080p content on the mobile, allow streaming of high definition content from YouTube, Hulu, Netflix etc., allow the user to record content at 1080p resolution, provide applications to edit and share the content on the move, capture images with high resolution cameras greater than 8MPixel, support very advanced imaging algorithms to produce superior picture quality and also support high definition video chat in peer to peer up to resolutions of 720p. There is a serious effort to analyse the performance, power and memory usage of these multimedia use cases and optimize software to improve battery life as well as performance.

OMAP processors, developed by Texas Instruments Inc., are used as the application processor in today's high end mobile phones, tablets and other consumer devices. The current generation OMAP, called the OMAP4 is capable of supporting all of the above mentioned multimedia use cases. The architecture of OMAP is designed to provide best-in-class video, image, and graphics processing for 2.5/3G wireless terminals, high-performance personal digital assistants (PDAs).

The OMAP supports the following functions:

- Streaming video up to full high definition (HD) (1920 × 1080 p, 30 fps)
- 2-dimensional (2D)/3-dimensional (3D) mobile gaming
- Video conferencing
- High-resolution still image (up to 16 MP)

The complex of all above use- case is Video Conferencing as it involves Camera, Video Encoder and video decoder. Video chat is point-to-point (two persons) Video Conferencing. As the two participants speak to one another, their voices are carried over the network and delivered to the other's speakers, and whatever images appear in front of the video camera appear in a screen on the other participant's phone. When the actual video data is exchanged between the two devices, both devices are peers in the network stack hierarchy. So the video chat is kind of peer-to-peer communication.

## 1.2 Thesis Objective and Scope

To build an optimized solution, we need to analyse several parameters in the system and continuously work towards understanding the intricacies. The project aims to build a Ducati System Analyser with the following features:

- Memory analysis.
- OS Resource utilization.
- Peak stack usage (per Task).
- Peak heap usage.
- Interrupt analyser.
- Runtime TILER* memory usage per use case.
- Build a static dependency graph based on:
    - Files.
    - Functions.
    - Data structures.
- Build a runtime dependency graph based on:
    - Functions.

- Use all of the above data to identify critical bottlenecks and scope out functions for further optimizing the system:
    - Highly used functions and scope for optimization.
    - Recommendations for refactoring/rewriting code.

These parameters will be analysed for the peer to peer video chat (may be extended to video conferencing) use case and the solution will be in the form of following

- ✓ Recommendations for optimizations (Example-functions calls causing run time overhead can be optimized to macros or such functions can be put in the cache).
- ✓ Recommendations for redesign and refactoring.

To build an optimized solution, we need to analyse several parameters in the system and continuously work towards understanding the intricacies. With a thorough study of the current design of the Video Chat application, I developed Ducati System Analyser which will help developers to actually see trade-offs between all these parameters so they can configure chat application and find out more optimal configurations.

*The fourth generation Open Multimedia Applications Platform (OMAP) processor from Texas Instruments provides a better way to address video and imaging memory needs using the Dynamic Memory Manager Hardware module (DMM) and its Tiling and Isometric Lightweight Engine for Rotation (TILER) sub-module.

## 1.3 Overview of video encoding

A compression encoder works by identifying the useful part of a signal which is called the entropy and sending this to the decoder. The remainder of the signal is called the redundancy because it can be worked out at the decoder from what is sent. Video compression relies on two basic assumptions. The first is that human sensitivity to noise in the picture is highly dependent on the frequency of the noise. The second is that even in moving pictures there is a great deal of commonality between one picture and the next. Data can be conserved both by raising the noise level where it is less visible and by sending only the difference between one picture and the next. In a typical picture, large objects result in low spatial frequencies whereas small objects result in high spatial frequencies. Human vision detects noise at low spatial frequencies much more readily than at high frequencies. The phenomenon of large-area flicker is an example of this. Spatial frequency analysis also reveals that in many areas of the picture, only a few frequencies dominate and the remainder are largely absent. For example if the picture contains a large, plain object, high frequencies will only be present at the edges. In the body of a plain object, high spatial frequencies are absent and need not be transmitted at all. For example in MPEG, two-dimensional spatial frequency analysis is performed using the Discrete Cosine Transform (DCT). An array of pixels, typically 8 x 8, is converted into an array of coefficients. The magnitude of each coefficient represents the amount of a particular spatial frequency which is present. As shown in figure 1.1, with inter frame prediction; each frame in a sequence of images is classified as a certain type of frame, such as an I-frame, P-frame or B-frame. An I-frame, or intra frame, is a self-contained frame that can be independently decoded without any reference to other images. The first image in a video sequence is always an I-frame. I-frames are needed as starting points for new viewers or resynchronization points if the transmitted bit stream is damaged. I-frames can be used to implement fast-forward, rewind and other random access functions. An encoder will automatically insert I-frames at regular intervals or on demand if new clients are expected to join in viewing a stream. The drawback of I-frames is that they consume much more bits, but on the other hand, they do not generate many artefacts, which are caused by missing data. A P-frame, which stands for predictive inter frame, makes references to parts of earlier I and/or P frame(s) to code the frame. P-frames usually require fewer bits than I-frames, but a drawback is that they are very sensitive to transmission errors because of the complex dependency on earlier P and/or I frames. A B-frame, or bi-predictive inter frame, is a frame that makes references to both an earlier reference frame and a future frame.

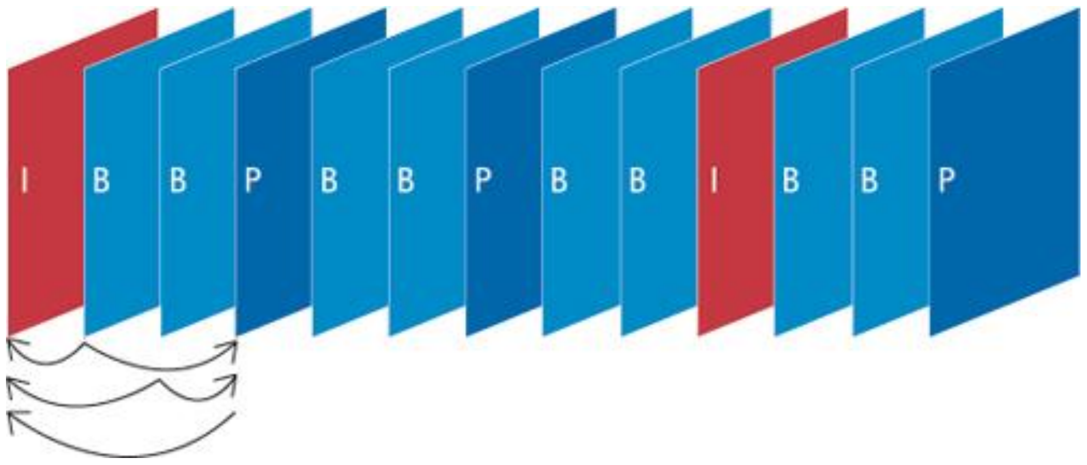


Figure 1.1 A typical sequence with I-, B- and P-frames. A P-frame may only reference preceding I- or P-frames, while a B-frame may reference both preceding and succeeding I- or P-frames

When a video decoder restores a video by decoding the bit stream frame by frame, decoding must always start with an I-frame. P-frames and B-frames, if used, must be decoded together with the reference frame(s).

## 1.4 Video Chat

There has been tremendous growth and new research in the field of digital communication, especially in wireless communication. This along with advanced networking protocols, such as 802.11 & cellular mobile networks have brought multimedia accessibility in our daily lives anytime, anywhere, on any device. With the availability of more bandwidth and efficient protocols, applications like video chat, video conferencing are no more a dream.  But it has also brought in challenges when using it for cellular phones. Basically Multimedia contents are characterized by the quality of service (QoS) - frame rate, supported resolution, and delay. The network QoS is characterized in the following parameters of network; bandwidth, packet loss rate, jitter, and delay. Low bandwidth for cellular phones prohibits the communication in image and/or video format. Also it creates the huge bandwidth gap between wireless and wired networks. This bandwidth gap demands coding technologies which achieve efficient compact representation of video data over wireless networks. So it is obvious that the most essential requirement for "wireless video" is coding efficiency [1]. H.263 developed for videophone achieves good compression ratios but also makes the signal susceptible to transmission errors. Large research is going to achieve high coding efficiency at various levels in hardware as well as in software.

Video services can be of two categories [2], first category is downloadable video, subscribers can download a video clip, but they won't receive the video in real time because cellular networks' limited bandwidth means that the download time will exceed the video's playing time (for example, 10 minutes to download a 2-minute video). The second category combines real time video with real-time voice. As video is just a sequence of still pictures, they differ only in bit rate. Video chat involves two peers simultaneously transmitting and receiving data (audio as well video). In Video conferencing there are more than two parties receiving and transmitting data. A video chat involves two parties (peers), possibly geographically interspersed, which exchange real-time video data; each peer process (peer) can play a role of not only sender but also receiver of video data. In addition, a peer on a mobile network is moving in the network.  It should be possible for peers to efficiently deliver multimedia contents, where quality of service (QoS) supported by the peers and communication channels are dynamically changing, possibly according to the movements of the peers. Video chat has become practical in commercial market because of the advances of technologies in networking and multimedia applications.

### 1.4.1 Adaptable Video Chat in cellular mobile network

Though there are advances in the wireless communication and networking protocols, still there exist lot of challenges and inefficiencies. Wireless video delivery faces several challenges, such as high error rate, bandwidth variation and limitation, battery power limitation, and so on. Specifically for real time video, where errors due to network cannot be corrected, above problems gets highlighted. We all know that clock speed is a key means of increasing CPU performance, but every time the clock speed goes up, so does the power. Also error resilience (The ability of system to recover quickly from error, change) is an important issue, as mobile networks cannot guarantee error-free communication.

To summarize, we can enlist following broad challenges in video chat in cellular mobile network

- Low bandwidth
- Power dissipation
- The data is sent over WLAN and WLAN is a highly lossy medium for video chat, with packet errors/loss at around 7%.[3]
- Late packets are worse than bad (with errors etc.) packets.
- Low Latency requirements.

To deal with above and similar problems like this, we need a highly efficient solution with less overheads, dynamic adaptability for different parameter's like bit rate, frame rate, frame resolution, codec schemes etc.

### 1.4.2 QoS Control

During video chat, systems should dynamically adapt themselves to the variations in the system resources such as network bandwidths, CPU utilization, and memory disk storage. Changes in parameters such as frame sizes, codec schemes, color depths, and frame resolutions can be agreed upon by users interactively based on their requirements for QoS (Quality of Service). Incorporating adaptability into systems minimizes the effects of the variations in system environments on the quality of video chat sessions.

As shown in Figure-1 there are four different layers of QoS parameters, application, system, network, and device. There are two types of QoS parameters, application-dependent parameters and application independent parameters. The former are high-level QoS parameters and the latter are low-level QoS parameters. There is a strong correlation between the parameters in these two subsets. For instance, frame rate is strongly related to the throughput in the network layer. Video transmission applications have to maintain a constant frame rate. The current TV frame rate is about 30 frames per second. The variation in available bandwidth does not allow this frame rate to be maintained without reducing the amount of data by trading off some aspects of video quality. There can be four aspects of video quality that can be changed to adjust to the available bandwidth [4]:

- Colour Depth Compression:

Colour video can be compared to grey-scale video to reduce the size of the data since gray-scale pixels require fewer pixels to encode than colour pixels.

- Frame Resolution Reduction:

Replacing every 2x2 matrix of pixels by one pixel can reduce the size of the video frame by a factor of four [4]. The image is reconstructed at the receiver to keep the physical size of the frame unchanged. Since the resolution reduction process is lossy the receiver gets a frame which is an approximation of the original.

| **QoS Layer** | **QoS Parameters** |
|---|---|
| Application | Frame Rate<br>Frame Size/Resolution<br>Color Depth<br>Response Time<br>Presentation Quality |
| System | Buffer Size |

| | |
|---|---|
| | Process Priority<br>Time Quantum |
| Network | Bandwidth<br>Throughput<br>Bit Error Rate<br>End-to-End Delay<br>Delay Jitter<br>Peak Duration |
| Device | Frame Grabbing Frequency |

Table 1: Quality of Service Layers [4]

- Frame Resizing:

  The frame size is changed to reduce the size of the data. For instance, reducing the frame size from 640x480 to 320x240 reduces the bandwidth requirement to 25% of the original.

- Codec Schemes:

  Different coding schemes have different compression ratios. Typically, schemes with high compression ratios require more time to compress but the smaller compressed frames can be transmitted more quickly. If the bandwidth available is extremely limited it might be worthwhile to reduce the communication time at the cost of computation (during compression) time.

Quality of service (QoS) specifications is used by distributed multimedia systems to enable applications and users request a desired level of service. The system attempts to satisfy the specifications, and if that is not possible due to resource availability restrictions, the application can enter into a negotiation with the system. During the negotiation process the QoS specifications are changed so that the system can meet the requirements.

An example of QoS parameters can be found in video-chat applications. This is a real-time application and needs a guaranteed supply of system resources to sustain a uniform level of performance. Some of the parameters are loss rate, throughput, frame rate, response time. This will allow the application or the user to negotiate with the system and arrive at a set of values for the parameters which both satisfy the user and can be supported by the system. The application can trade off some parameters in exchange for others.

The application or user can specify the different parameters desired. Upper and lower bounds can be used to express acceptable situations. From a communication point of view, the goal is to minimize response time and maximize accuracy of information, precision, presentation quality and comprehensiveness of the data. Relaxing the constraint on one of the parameters, which have to be maximised, might help in maximising the other parameters.  Adaptability and re configurability are needed to deal with the performance and reliability requirements of a system.

# Chapter 2 OMAP4 and Ducati

## 2.1 Overview of OMAP4 [5]

OMAP (Open Multimedia Application Processor) developed by Texas Instruments is a category of proprietary system on chips (SoCs) for portable and mobile multimedia applications. OMAP devices generally include a general-purpose ARM architecture processor core plus one or more specialized co-processors. The 4th generation OMAPs, OMAP 4430, 4460 (formerly named 4440), and 4470 all use dual-core ARM Cortex-A9s. The 4470 additionally contains twoCortex-M3s running at 266 MHz to offload the A9s in less computationally intensive tasks to increase power efficiency. 4430 and 4460 use a PowerVRSGX540 integrated 3D graphics accelerator which runs at a clock frequency of 304 and 384 MHz respectively compared to prior versions of SGX540 typically at 200 MHz making them theoretically much faster. [14] 4470 has a PowerVR SGX544 GPU that supports DirectX 9 which enables it for use in Windows 8 as well as a dedicated 2D graphics core for increased power efficiency. All OMAP 4 come with an IVA3 multimedia hardware accelerator with a programmable DSP that enables 1080p Full HD and multi-standard video encode/decode. OMAP 4 uses ARM-Cortex A9s with ARMs SIMD engine (Media Processing Engine, aka NEON) which may have a significant performance advantage in some cases over NVidia Tegra 2s Cortex-A9s with non-vector floating point units.] It also uses a dual-channel LPDDR2 memory controller compared to NVidia Tegra 2s single-channel memory controller.

The OMAP™ 4 platform includes applications processors, a comprehensive software suite and power management technology to bring next-generation Smartphone’s and Mobile Internet Devices (MIDs) quickly to market.

The architecture of OMAP is designed to provide best-in-class video, image, and graphics processing for 2.5/3G wireless terminals, high-performance personal digital assistants (PDAs). For that purpose, the device supports the following functions:

- Streaming video up to full high definition (HD) (1920 × 1080 p, 30 fps)
- 2-dimensional (2D)/3-dimensional (3D) mobile gaming
- Video conferencing
- High-resolution still image (up to 16 MP)

The device supports high-level operating systems (OSs) such as:

- Windows™ CE, WinMobile™
- Symbian OS™
- Linux®
- Palm OS™

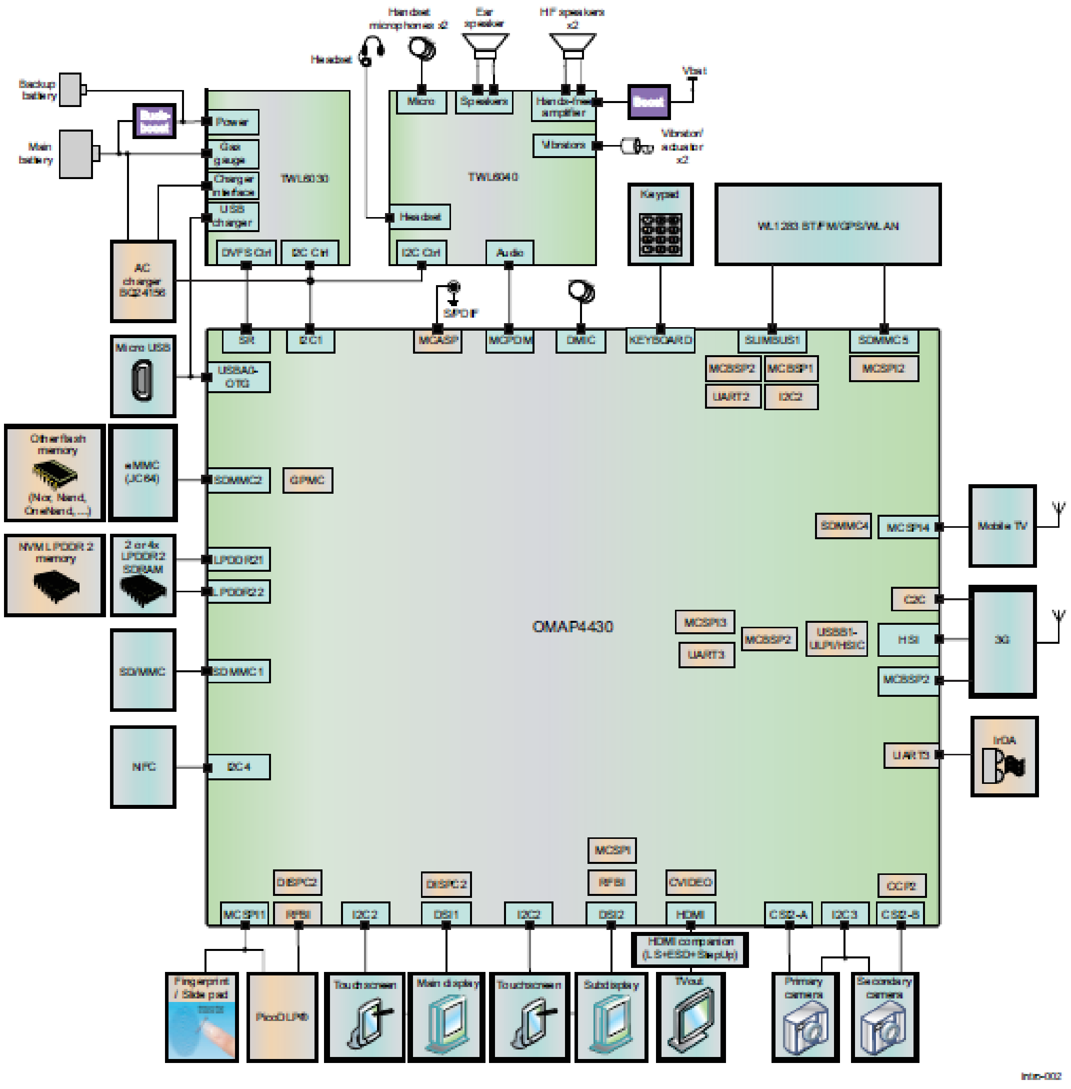


Figure 2-1 OMAP4430 High Tier Environment

The device is composed of the following subsystems:

- Cortex™-A9 microprocessor unit (MPU) subsystem, including two ARM® Cortex-A9 cores
- Digital signal processor (DSP) subsystem
- Image and video accelerator high-definition (IVA-HD) subsystem
- Cortex™-M3 MPU subsystem, including two ARM Cortex-M3 microprocessors

- Display subsystem
- Audio back-end (ABE) subsystem
- Imaging subsystem (ISS), consisting of image signal processor (ISP) and still image coprocessor
- (SIMCOP) block
- 2D/3D graphic accelerator (SGX) subsystem
- Emulation (EMU) subsystem

The device includes state-of-art power-management techniques required for high-performance mobile products. Comprehensive power management is integrated into the device. The device also integrates:

- On-chip memory
- External memory interfaces
- Memory management
- Level 3 (L3) and level 4 (L4) interconnects
- System and connecting peripherals

## 2.2 Overview of Ducati subsystem

Ducati Sub System comprises of two ARM® Cortex-M3 processors (CPUs), IVA-HD subsystem and ISS subsystem. The two Cortex-M3 processors are known as Sys M3 and App M3 and they comprise the MPU of Ducati Subsystem. Figure 3 shows a simplified block diagram of Ducati subsystem, with arrows indicating the software flow.

As shown in the figure Ducati Subsystem contains following blocks

- Dual Core Cortex-A9 MPU is the OMAP4 Host Processor running a High Level Operating System (HLOS) such as Linux or Android in Symmetric Multi-Processing (SMP) mode. It uses the Ducati Sub System for video and image acceleration.
- One of the 2 Cortex-M3 processors (Sys M3) runs the Notify Driver which accepts commands from the HLOS software and then provides these to the other Cortex-M3 processor (App M3).
- The entire Ducati multimedia software executes on App M3. OMX components on App M3 invoke the necessary APIs for video or image processing on IVA-HD subsystem or ISS subsystem.
- On the reverse path, App M3 can directly acknowledge the Dual Core Cortex-A9 MPU.
- BIOS 6.x run on both Cortex-M3 MPUs in non-SMP mode.

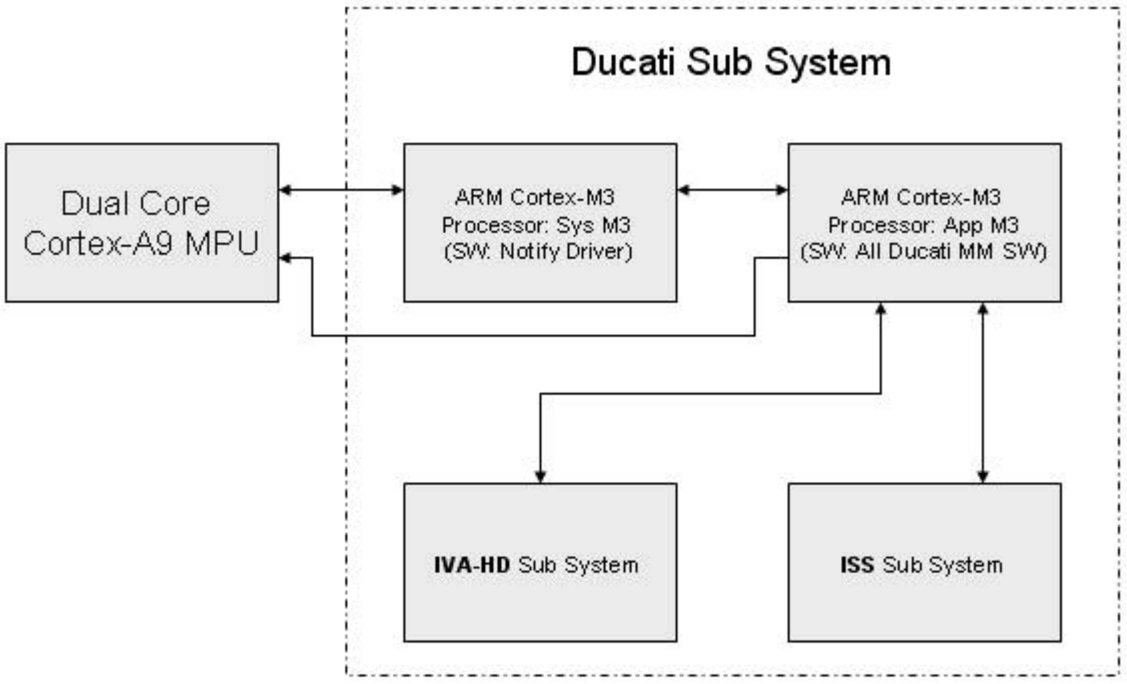

Ducati Sub System
Dual Core Cortex-A9 MPU
ARM Cortex-M3 Processor: Sys M3 (SW: Notify Driver)
ARM Cortex-M3 Processor: App M3 (SW: All Ducati MM SW)
IVA-HD Sub System
ISS Sub System

Figure 2-2 Ducati Subsystem Block Diagram

### 2.2.1 IVA-HD Overview

The Image Video Accelerator - High Definition (IVA-HD) Sub System is composed of hardware accelerators which enable video encoding and decoding up to 1080 p/i resolution at 30 frames per second (or 60 fields per second).

IVA-HD sub-system supports multiple video coding/compression standards. Both encoder and decoder of the following video standards are currently supported, with performance up to 1920x1080 resolutions at 30 fps (1080p30)

### 2.2.2 ISS Overview

The Imaging Sub System (ISS) deals with the processing of pixel data coming from an external image sensor or from memory. ISS as a component forms part of still image capture, camera viewfinder and video record use-cases.

Following are Imaging Subsystem constituents:

- Two CSI2 (Camera Serial Interface): Receives data from sensor.
- CCP2 (Camera compact port): Receives data from sensor or reads from memory.
- ISP (Image Signal Processor): Pre/post processing operations on pixel data (received from sensor or from memory).
- SIMCOP (Still Image Processor): Imaging accelerator.
- BTE: Burst Translation Engine. Converts from raster to 2D tiled order and vice versa, for data write and data read respectively.
- CBUFF: Circular Buffer for linear space, physically located in memory.

Among these, ISP and SIMCOP are the two major processing blocks.

#### 2.2.2.1 ISP (Imaging Signal Processor)

The Image Signal Processor (ISP) subsystem is designed to perform various kinds of pre/post processing operations on pixel data.

#### 2.2.2.2 SIMCOP (Still Image Coprocessor)

The Still Image Coprocessor (SIMCOP) subsystem is designed to encode, decode, and process image data. SIMCOP is a block-based memory-to-memory processing engine. It fetches blocks from system memory and stores them into local memories. Different accelerators take the fetched data, perform processing, and send the processed output back to local memories. From there the data could be further processed by other accelerators or be sent back to system memory. The SIMCOP needs an external central processing unit (CPU) to perform high-level control tasks and configurations; in the current software design it is closely coupled to Cortex App M3.

#### 2.2.2.3 Camera

ISS has two camera interfaces: primary and secondary. The primary interface is CSI2-A and the secondary interface is CSI2-B/CCP2. All interfaces can use the ISP, but not concurrently. When one interface uses the ISP, the other one must send data to memory. However, the ISP can still be used to process this data in memory-to-memory. Time multiplex processing is also possible. Primary sensor may be up to 16 MP and secondary up to 5 MP

The camera subsystem can manage up to two serial image sensors that can be active at the same time. However, only one data flow can use the ISP. Because CSI2-B and CCP2 share pins, they cannot be used simultaneously. CSI2-A and CSI2-B/CCP2 can function at the same time by sending data to memory.

### 2.2.3 Ducati SW overview

The following diagram indicates the main software that will run on Ducati. To the outside world Ducati will look like a single processor.

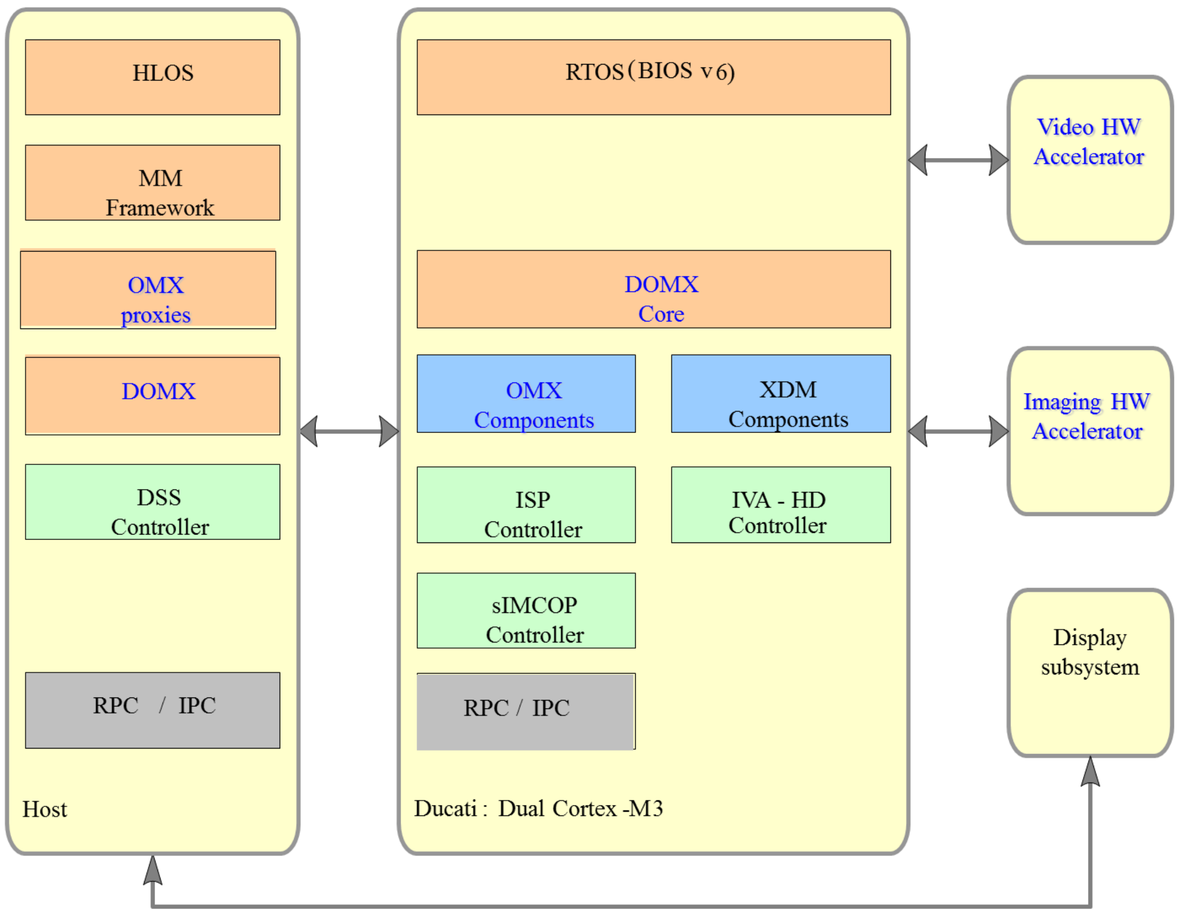


**Figure 2-3 Ducati multimedia software bird's eye view [14]**

## 2.3 Next generation multimedia architecture: OpenMAX IL on multi-processor Platforms

### 2.3.1 Overview of OpenMAX

To provide codec's with a degree of system abstraction for the purpose of portability across operating systems and software stacks, in OMAP4 platform OpenMAX standard is used. OpenMAX (Open Media Acceleration) is a royalty-free, cross-platform set of C-language programming interfaces that provides abstractions for routines especially useful for audio, video, and still images. It's intended for devices that process large amounts of multimedia data in predictable ways.

OpenMAX provides three layers of interfaces: Application Layer (AL), Integration Layer (IL) and Development Layer (DL). OpenMAX AL is the interface between multimedia applications, such as a media player, and the platform media framework. It allows companies that develop applications to easily migrate their applications to different platforms (customers) that support the OpenMAX AL API.

### 2.3.2 OpenMAX IL Layer API

OpenMAX IL is the interface between media framework such as DirectShow or GStreamer and a set of multimedia components (such as an audio or video codec's). It allows companies that build platforms (for example an MP3 player) to easily change components like MP3 decoders and Equalizer effects and buy components for their platform from different vendors. The OpenMAX IL API strives to give media components portability across an array of platforms using the C-language. In the OpenMAX IL, components represent individual blocks of functionality. Components can be sources, sinks, codec's, filters, splitters, mixers, or any other data operator. Depending on the implementation, a component could possibly represent a piece of hardware, a software codec, another processor, or a combination thereof. The interface abstracts the hardware and software architecture in the system. The OpenMAX IL API allows the user to load, control, connect, and unload the individual components. This flexible core architecture allows the Integration Layer to easily implement almost any media use case and mesh with existing graph-based media frameworks. The key focus of the OpenMAX IL API is portability of media components OpenMAX DL is the interface between physical hardware, such as DSP chips and CPUs, and software, like video codec's and 3D engines. It allows companies to easily integrate new hardware that supports OpenMAX DL without reoptimizing their low level software. The OpenMAX IL API gives applications and media frameworks the ability to interface with multimedia codec's and supporting components (i.e., sources and sinks) in a unified manner. The components themselves may be any combination of hardware or software and are completely transparent to the user. Without a standardized interface of this nature, component vendors have little alternative than to write to proprietary or closed interfaces to integrate into mobile devices. In this case, the portability of the component is minimal at best, costing many development-years of effort in re-tooling these solutions between systems.

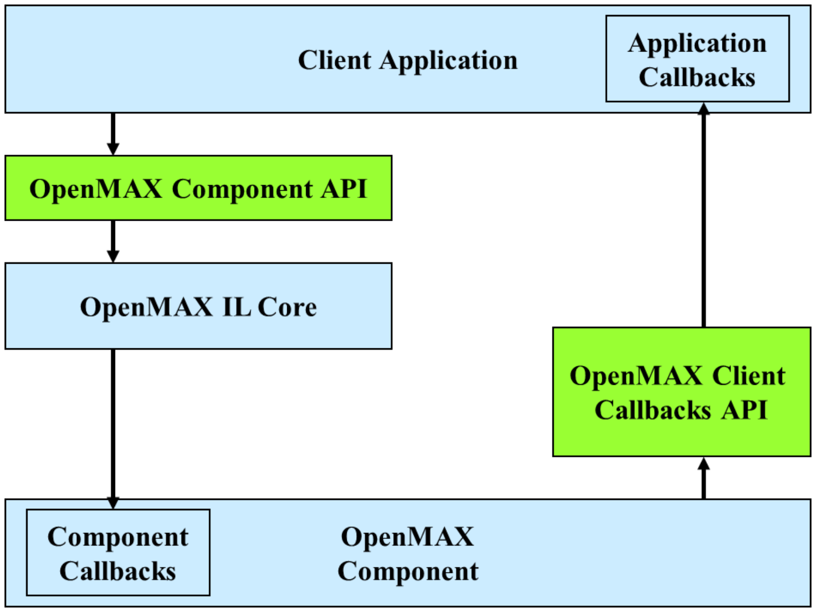


**Figure 2-4 Communications between OpenMAX Components and Application on Single Processor [15]**

Figure 4 shows communications between OpenMAX components and application on single system. As component and client application resides in the same system IL client application directly communicates with the component through component interface by using OMX core. The OpenMAX IL API gives applications and media frameworks the ability to interface with multimedia codec's and supporting components (i.e., sources and sinks) in a unified manner. The components themselves may be any combination of hardware or software and are completely transparent to the user. Without a standardized interface of this nature, component vendors have little alternative than to write to proprietary or closed interfaces to integrate into mobile devices. In this case, the portability of the component is minimal at best, costing many development-years of effort in re-tooling these solutions between systems

OMX IL FEATURES:

a) OpenMAX IL has
    i. OMX Core: A Thin layer between an application framework and the OMX components. OMX Core is responsible for Initialization of the components, De-Initialization of the components, Connecting (or) Tunnelling of OMX components and Providing APIs and macros to application so that application can interact with the component.
    ii. OMX Component Provides a common interface to the codecs, drivers and services
    iii. IL Client Integrates with most multimedia frameworks, consumer of OpenMAX Component services
b) OpenMAX IL components have ports to receive and send buffers.
c) Application communicates to OpenMAX IL components using "OpenMAX IL Core".
d) OpenMAX IL 1.1 supports 3 modes of buffer communication
    i. Non-Tunnelled – All buffer passing between components is handled by application

   ii. Tunnelled – Components pass buffers to each other without application involvement
   iii. Proprietary – Components pass buffers to each other using non-standard proprietary method.

e) Components are grouped into 2 profiles
   i. Base Profile - Shall support Non-Tunnelled communication and may support Proprietary communication
   ii. Interop Profile – Shall support Non-Tunnelled and Tunnelled communication and may support Proprietary communication.

f) Three categories  of API's
   i. Control: There is a large number of predefined parameter/config. structures for audio/video/image codecs and algorithms Vendors can define their own parameter/config. Structures.

      Example of synchronous control APIs
      - SetParameter
      - GetParameter
      - SetConfig
      - GetConfig
      - GetState
      - GetComponentVersion
      - GetExtensionIndex

      Example of asynchronous control APIs
      - SetState
      - EnablePort
      - DisablePort
      - Flush

– MarkBuffer

ii. Data Buffer Allocation: In OpenMAX, a buffer could be allocated in any one of the three places –

1) Client, 2) Producer or 3) Consumer

In Non-Tunnelled mode, a client uses ‘OMX_AllocateBuffer’ to request a component to allocate buffer. ‘OMX_UseBuffer’ call is used to pass a buffer to a component for use [after allocating it in the client or in some other component]

iii. Data Buffer Passing

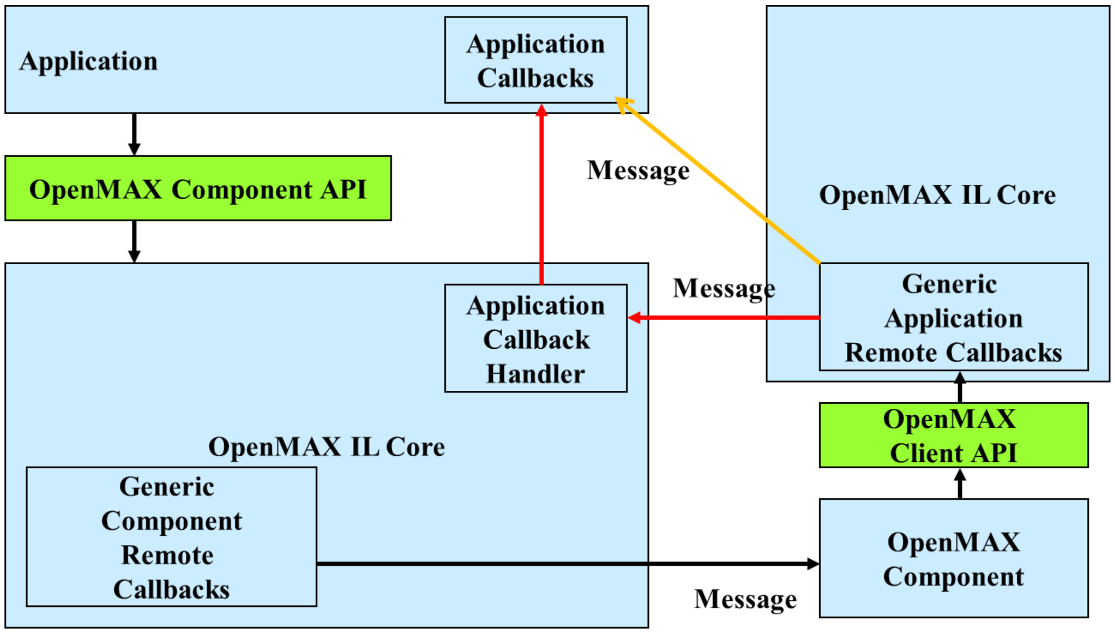


**Figure 2-5Communication between OpenMAX Components and Application on Multiple Processors – Solution 1 (Red Lines where application uses call back to receive notification from components), Solution 2 (Yellow where application directly receives messages from components).**

# Chapter 3 Literature Survey

There has been a lot of research in the field of efficiency and Quality of Service (QoS) in video chat and video conferencing. This chapter provides a detailed view of the works done in the literature in the field of adaptability and QoS of video chat. The result of the literature survey is summarized in the end of the chapter.

## 3.1 Studies carried out in Literature

**Phanish, Radha, Sherin, Parag, Joe Meehan, Mike Hannah (TIITC2011)** proposed a method to enable Low Latency Processing Path during Video Record use case at OMX Level. They succeeded to improve latencies 16% without VNF (Video Noise Filter) case and by 25% in VNF case over the frame based approach. In their proposed they used Pipelined Processing by enabling Sub Frame Processing at each processing blocks (ISP, VNF & Video Encode) and used OpenMAX Tunnelling Interface Standard for Direct Communication between OMX Camera & OMX Video Encoder. In their method they tried to optimize normal AV (Audio Video) record use-case on OMAP4 using innovative Design and/or Implementation approach at each Processing components level as reduction in End to End system latency depends upon individual blocks processing time in the system and the way data flow happens in these blocks (sequential vs. pipelined). They tried to reduce overhead of IPC by enabling direct communication channel between the Camera & Video Encoder while Processing the Data without client intervention as both components are on same subsystem. In their work, they identified that sequential approach (Frame based processing) have several frame latencies which can be optimized like

– Each Individual blocks ISP, VNF & Video Encoder is working on Frame Inputs and Generating Frame outputs

  - Between CAM and Encoder, ISP is works on different frame than VNF and Video Encoder.

Their proposed solution suggests operating on sub frame level, which is summarized as follows

- Sub-frames (segmented frames) are processed from camera through ISP (Image Signal Processor) and VNF (Video Noise Filter) to reduce latency.
- Each Individual blocks ISP, VNF & Video Encoder is working on Sub Frame Inputs and Generating Sub Frame outputs.
- IVA-HD works on slice input/output also DSS (Display Subsystem) works on slice input.
- The call flow is as follows

  a. IL client Provides Buffer to OMX camera to dump the data.
  b. OMX camera sends the sub frame availability information to the Video Encoder
  c. OMX video encoder Processes that Sub frame input and generates the Sub frame output.
  d. This way both Camera and Video Encoder operates in parallel at Sub frame level.
  e. As soon as OMX camera finishes processing of the entire frame/last sub Frame data it sends to the IL client for Display

**Mahant Siddaramanna, Naveen Srinivasamurthy and Yashwant Dutt (TIITC2010)** discussed about optimizing video encoders for low delay and channel constrained transmissions. In their solution they addressed problems like Need for low end to end delay and information frame flicker/pulsing at low bitrates for Video Conferencing and gaming applications. They proposed Gradual Intra Refresh (GDR) instead of IDR (Instantaneous Decode Refresh). In GDR for every picture only part of frame is coded with intra MBs. Proposed solution can be summarized as follows

a. Use row level multiple slices for GDR.
b. One by one, slices from top to bottom are refreshed until all the MBs (Micro Blocks) in a frame are refreshed.
c. Along with the current slice, one row above the current slice being refreshed should also be refreshed.
d. Filtering between slices should be avoided.

**Aditya Monga, Sarthak Aggarwal, Sudhir Kasargod (TIITC2010)** discussed about Adapting TI Multimedia solution for Generic HLOS (High Level Operating System) frameworks. They presented how the OMX based TI multimedia solution was made broad based to ensure seamless integration with HLOS multimedia frameworks like GStreamer (Linux), OpenCore/StageFright (Android) etc. They addressed Adaptations required at OMX layer required in the TI multimedia stack to adapt to HLOS frameworks and make best use of the available hardware features.

**Karthik Ramanan, Rob Clark, Mukund Mittal** in their paper Innovations and challenges in ramping up the first OMAP4 product – Playbook (**TIITC2011**) tried to improve the quality of video chat in their implementation.

They proposed following solutions to avoid noise in the video chat and also studied trade-off’s while implementing it.

a. Increase bit-rate to 768kbps or 1Mbps.
b. Periodic I-Frame refreshes to prevent error propagation.
c. LTRP (Long Term Reference Pictures)

They selected LTRP as solution which is described as follows

a. Every Nth frame serves as a long-term reference.

b. Only have I (Information) frame at the beginning of the sequence.

c. Receiver acknowledges all long-term reference frames received properly to the transmitter.

d. The encoder only uses as a reference a long-term reference frame that has been acknowledged by the receiver (if not acknowledged, keep using previous long term reference).

e. For example, in figure 3-1 if the first P frame that was going to serve as a long term reference in the example figure is not acknowledged (3rd P from the left), then encoding of the 5th, 6th and 7th P (one marked with X for loss) would have been relative to the previous long term reference frame (I) instead. Because of this, only frames that are dropped are missing and we can decode and display the immediate next frame. I P P P X P P P P

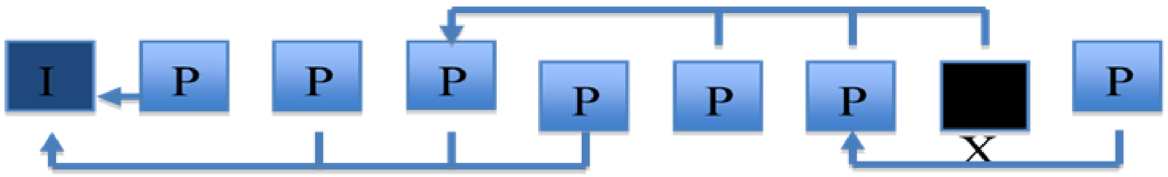


Figure 0-1 Long Term Reference Pictures

f. Authors list following advantages for using LTRP over conventional techniques in which its needed to send I frames periodically

- Only frames that are not displayed to the user are those that are missing
- No artefacts or added latency added
- No need for I frames except for first – this improves compression

LTRP is not perfect for Video Chat as it has some disadvantages like Less compression efficiency due encoding relative to reference farther in the past but keeping them fairly close, should not be too bad and risks like in LTRP Encoder must ensure that frames are not skipped (all are encoded); if a frame would be skipped, the scene would go back in time a few frames from the previous frame which could be quite noticeable.

LTRP can be combined with following options

- Whenever there is a packet loss in the network, the codec can use the last I-Frame as the reference. This prevents errors from being propagated in further decoded frames.

**Deepak Poddar, Pramod Swami, Keshava Prasad, Pavan Shastry, Adireddy Ramakrishna, Mody Mihir (TIITC2010)** tried to reduce latency at video codec level. In their paper, Ultra Low Latency Video Codecs for IVAHD Based SOC's: Design Challenges and Solutions, they proposed a callback based mechanism in the codec for sub frame level data exchange. This Paper highlights the encoder/decoder design challenges and the details of the low latency solutions available on IVAHD. The codec supports various sub-frame data exchange modes and the application can choose a suitable mode. They claim in this solution, the codec latency can be reduced from 33 ms to less than 1 ms. In the callback based mechanism Encoding/Decoding can begin without getting data of the entire frame and the codec makes callbacks when it needs more data. This method is called as data sync (data synchronization between application and codec).

**Bharat Bhargava (Adaptable Software for Communications in Video Conferencing)** has discussed techniques for adaptability at the application, system, and network layer to meet the quality of service requirements. He tried to incorporate adaptability into a video conferencing systems minimizes the effects of the variations in system environments on the quality of video conference sessions. In this paper he identified four aspects of video quality that can be changed to adjust to the available bandwidth like colour depth compression, Frame Resolution Reduction, codec schemes and frame resizing.

**Minoru Etoh, and Takeshi Yoshimura (Advances in Wireless Video Delivery, IEEE invited paper)** reviewed practical video delivery technologies, examining existing mobile networks, commercialized or standardized transport, and coding technologies. Compression efficiency, power dissipation, and error control are intrinsic issues in wireless video delivery. Among these issues, error control technologies are evaluated at four layers layer-1/2 transport, end-to-end transport layer such as TCP/IP or RTP/UDP/IP, error-resilience tool and network adaptation layer and source coder layer. They also identified adaptive rate control and error recovery as more crucial issues for future research.

## 3.2 Summary of the Literature Survey

| **Author Name & Year** | **Efficiency criteria/Strategy Focused** |
|---|---|
| Phanish, Radha, Sherin and others | Low Latency Processing Path during Video Record use case at OMX Level |
| Mahant Siddaramanna, Naveen Srinivasamurthy and Yashwant Dutt | Optimizing video encoders using Gradual Intra Refresh (GDR) |
| Aditya Monga, Sarthak Aggarwal, Sudhir Kasargod | Adapting TI Multimedia solution for Generic HLOS (High Level Operating System) frameworks. |
| Karthik Ramanan, Rob Clark, Mukund Mittal | LTRP (Long Tern Reference Picturing) |
| Deepak Poddar, Pramod Swami and others | Reduce latency at video codec level |
| Bharat Bhargava | Adaptable Software for Communications in Video Conferencing |

Table 3-2 Summary of the Literature Survey

# Chapter 4 Implementation and Result

Understanding system architecture is essential for the feature enhancements, bug fixes and the redesigning critical blocks of the system. Most systems are continuously evolving and changing on a daily basis with new requirements being added to cater to the market needs. In these scenarios it is fairly impossible to maintain documentation that reflects the current architecture. So you would find that in any real world project, the architecture would never reflect the current state of the system; it must be retrieved from the system implementation using program analysis – which could involve code walk-through, API documents and other low level system analysis

As these analyses are not unique by nature, system engineers have to be involved to accept or reject certain results proposed by the automatic analyses. Hence, the result of such analyses ought to be presented in a form that is intuitive to the system engineer. Therefore, it is needed to provide multiple graphical views combining different aspects of the software to understand. Views can be static or dynamic views: Static program information captures the program structure, but even elaborated analysis techniques obtain only little information on the runtime behaviour of the program in advance. Hence, we additionally need dynamic information visualizing the behaviour of an example run of the program. The major task in understanding the architecture of a system is the identification of components and the essential communications between them. Components are larger units of "coherent" modules or classes. The notion of coherency usually mixes static and dynamic system properties: it includes structural connection between the modules or classes in the call or inheritances graphs (static information). Additionally, it requires strong interactions between the modules or classes by procedure calls (dynamic information). The essential communications between the components define the transfer of data independently of their implementations by simple calls, shared memory accesses, events, or callbacks. Many analyses only picture this implementation of communication. E.g. static structure analysis often comes up with misleading results.

Combining graphic views of static and dynamic aspects of software and has already been established for software documentation. A typical means to illustrate programs are UML diagrams. There are a couple of different diagram types, such as static class diagrams, but also the dynamic sequence diagrams which developers have found useful for communication.

For understanding large software systems, it's necessary to reduce the amount of information displayed. Filters as well as aggregations perform such a reduction:

- *Information filtering*: users want to disregard parts of the software system or phases of the execution. They should be able to define, enable and disable filters.
- *Information aggregation*: Even if some information is analysed, it is not necessarily displayed in detail. A single representative entity can visualize several computed objects and relations.

Like most of complex systems. Ducati software architecture is not well-documented. There aren't enough comments in the code to aid the understanding either. In order to understand the Ducati software, the developers have to start with repository that contains 3 million lines of code. Ducati software is the controller for two critical blocks in the OMAP4. The functionality of Ducati software ranges from high level OMX API's to low level drivers that control the various peripherals, hardware accelerators and external hardware modules like sensors, external ISPs and flash controllers. This makes the software fairly complex, it is not very easy to understand and analyse the code using traditional methods. Therefore, tools to automatically extract design and architectural information are required. The need of the hour was a tool that could not only make it easy to understand the design of such a large system but also provide some insights and inputs to optimize. Clearly, there was a need for visual graphical views combining different aspects of the software which will present both the static as well as dynamic information.

The Ducati System Analyser was designed and developed for exactly this purpose (visualization of software structures and behaviour). The Ducati System Analyser combines dynamic and static aspects of the software system using a visualization framework for the graphical presentation of the generated information. It allows for information filtering and aggregation. Using this architecture as a basis, we analyse components and essential communications. We visualize the results of the analyses in a graphical form that is intuitive to the system engineer enabling to accept or reject these results.

The design and development of Ducati System Analyser was done in three phases

I. Phase one (static analysis) removes all redundant files, dead code (Ex. code in between #if 0 and #endif), functions, data structures (Ex- structures), hash defines from the source repository to make it clean.

II. Phase two (dynamic analysis) shows call sequence for interaction between host processor (A9) and Ducati (M3). This is mostly an API level interaction with the details on the structures and parameters exchanged. The output is text based.

III. Phase three (dynamic analysis): implements the full system call flow depicting function call flow from the API level the driver level. The tool also supports “task aware” call flow analysis. User has the option to choose different visualizations: Trace2UML and Microsoft Excel.

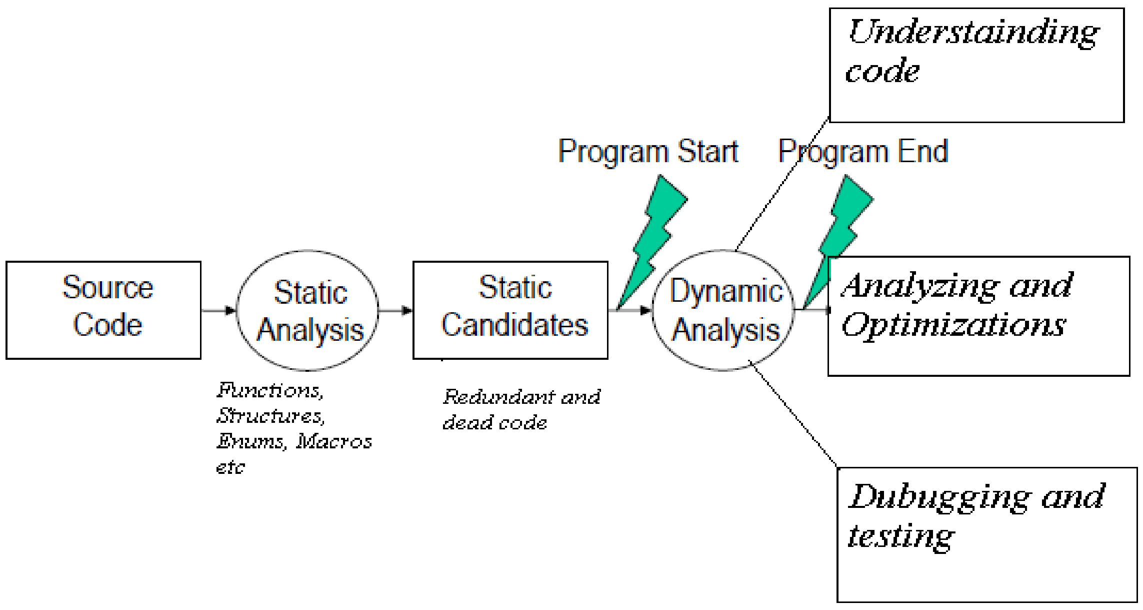


Figure 4- 1 Ducati System Analyser

### 4.1 Ducati Repository Clean-up (DRC)

During the course of software development, many modules (files, structures, enums, hash define etc.) get obsolete during the project life cycle. This obsolete or redundant code unnecessarily increase code size and in many cases run time overhead For example including header file unnecessarily causes compiler overhead of adding the content of header file to source file. So it's necessary to have a clean repository; it will not only make code reading easy, but it will also have run time performance benefits. Thus there is a need to do some static analysis to remove redundant data structures, functions etc. periodically. In the phase we have removed redundant macros, structures, functions, dead code and files that are redundant.

#### 4.1.1 DRC for Dead code (Code in between #if 0 and #endif)

Complex systems like Ducati software take years to develop and mature. So in this process new code gets added and old code gets removed. But it's common programming practice that instead of removing code, programmers just put it in hash if zero block (conditional compilation directive). Though this does not add runtime overhead unnecessarily it increases code size. In this phase of DRC, we have removed all dead code. Surprisingly it's more than 5 percent of all repository code size.

### 4.1.2 DRC for Macro's

The first stage of the compilation is to invoke the C pre-processor to expand macro definitions. Most likely use of it is to using the pre-processor to include files directly into other files, or #define constants, but the pre-processor can also be used to create "inlined" code using macros expanded at compile time and to prevent code from being compiled twice.  There are essentially three uses of the pre-processor directives, constants, and macros. During the development life cycle new macros get added and some get obsolete which should be removed. In this we have written a Perl script which will find unused macros and will also remove it from the repository. Also it will help to identify some interesting coding practices like nesting of macros, concatenation operators to  make macro name which make code hard to understand also adds runtime overhead (As to resolve nested macros compiler needs several passes).

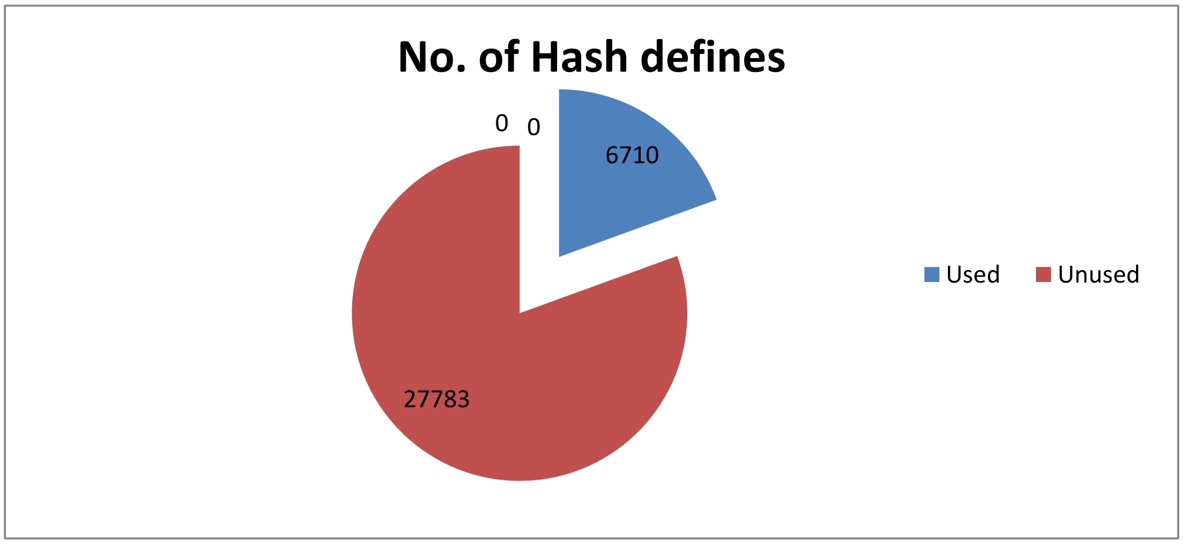


Figure 4- 2 Static analyses of Macro's

Figure 4-2 shows statistics about the macros in the Ducati source code. As it can be seen more than 80 percent of macros are unused. Which if removed will not only reduce code size but also make it easy to understand. Also with statistics like no. of nested and concatenated macros, code can be made unambiguous

### 4.1.3 .DRC for Structures

Like macros, structures also get obsolete or may be redundant because of used in different phases of software development life cycle for Example Some data structures may be used for testing purposes only . This kind of structures should be removed from the code as they serve no purpose and increase code size. Figure 4-3 shows ‘redundant’ to ‘used structure’ comparison and it can be seen that 20 percent structures are unused. Here also, interesting coding practices like concatenation can be found which can be analysed and removed.

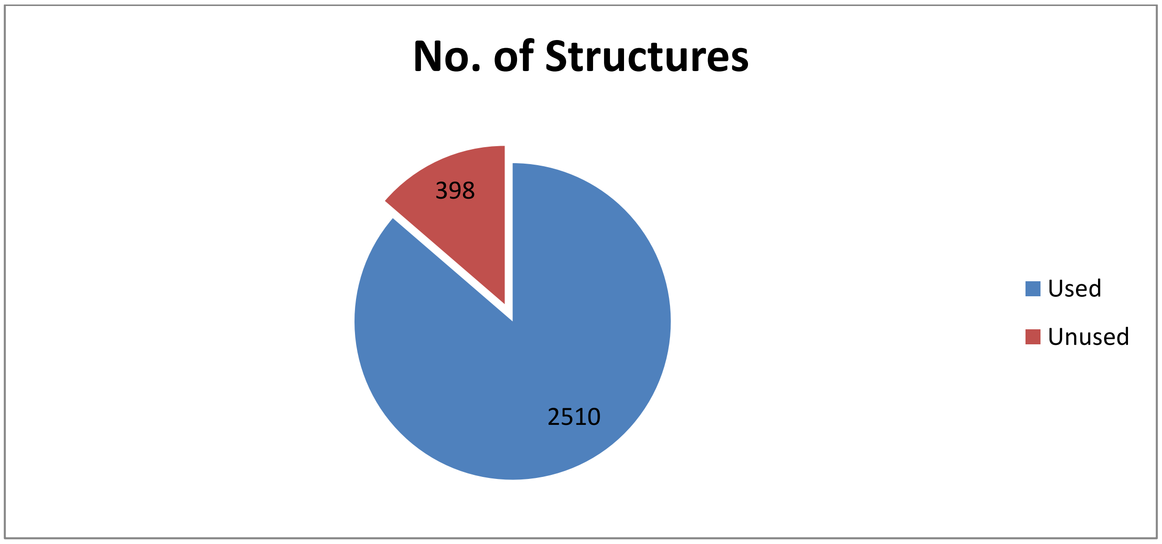


**Figure 4- 3 Static analyses of structures**

### 4.1.4 DRC for functions

Similar to structure and macros, functions also get obsolete or may be used for only some point of time like for testing purpose. DRC for functions removes all such functions. Interestingly there are function calls which are made by concatenating the strings, making it hard to recognize for source code editor and analyser tools like Source Insight. Through the script written for DRC for functions, such calls can be identified and modified.

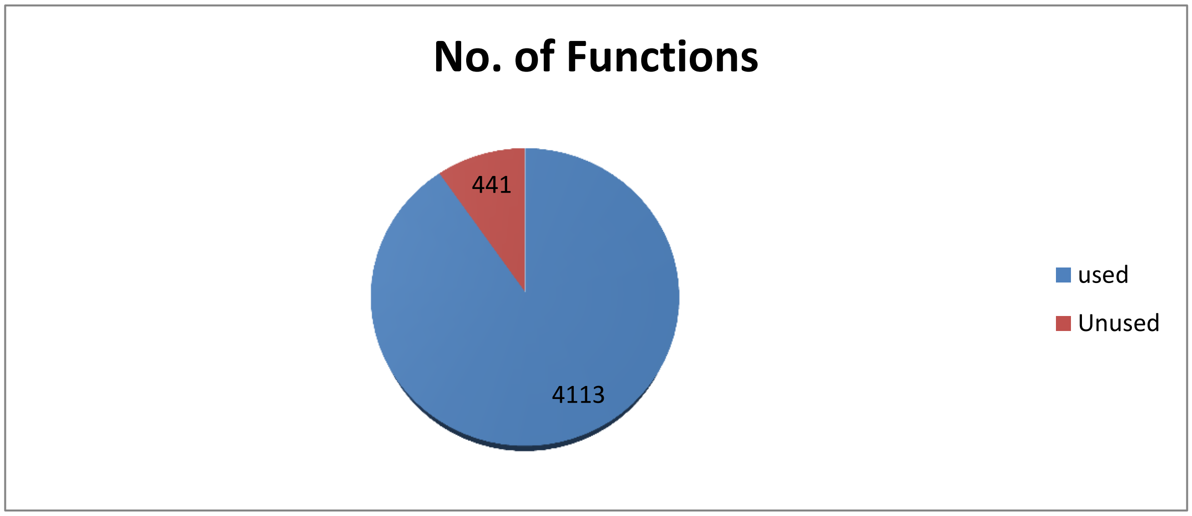


**Figure 4- 4 Static analyses of functions**

## 4.2 Ducati System Analyser Phase-II

Real world multimedia use-cases demand great performance targets usually achieved using remote processors and/or hardware accelerators. This typically mandates the need for an inter-processor framework to access the remote processor from a host processor. For similar reasons OMAP uses Ducati Subsystem for multimedia use cases. To understand such subsystem, which are controlled by host system running on remote processor one need to understand and analyse the interaction points between these systems. As all IPC happens through these interaction points, one can get slight idea of control flow *with* Ducati by looking at these interaction points, though it does not give the details of the call flow *within* Ducati.

In phase two of Ducati System Analyser, we have manually inserted instrumentation code at the Ducati interface. The interface for the Ducati to communicate with the host processor in either direction happens through the functions defined in omx_rpc_skel.c and omx_rpc_stub.c. As shown in figure 4-5, for every use-case that involves the Ducati, the host operating system invoke a remote procedure call through IPC mechanism to call one of the functions in omx_rpc_skel.c, which then directs it to appropriate function to do necessary processing and every return call goes through omx_rpc_stub.c.

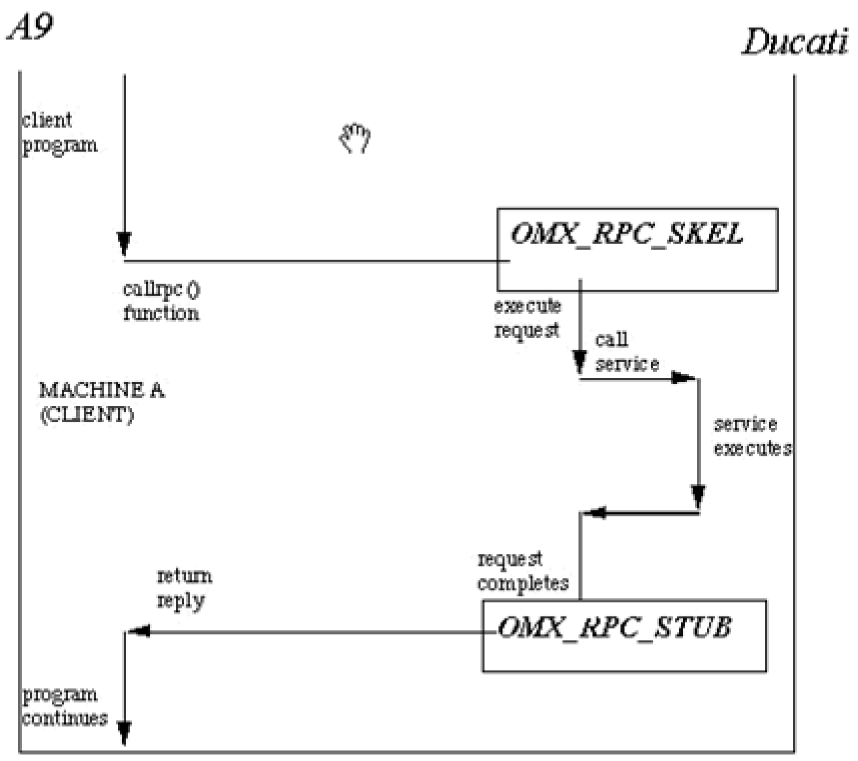


**Figure 4- 5 Interfaces for Ducati**

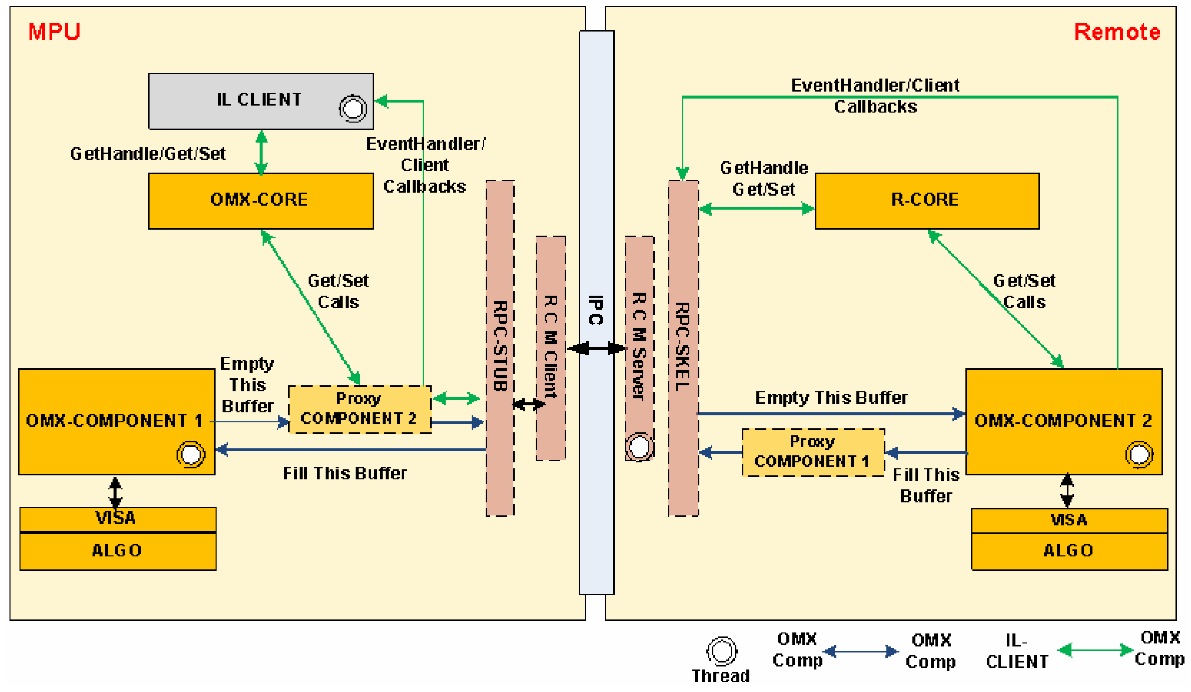


Figure 4- 6Distributed OpenMAX (DOMX).

Figure 4-6 shows IPC architecture for Ducati in more detail with DOMX (Distributed OpenMAX). It shows that the local call from IL client is directed to Ducati via RPC_STUB (client side) and RPC_SKEL (Ducati side).

Steps for Ducati System Analyser Phase II

a) Apply patch "Added-support-for-Ducati-callflow-graph. Patch" (from Patches folder) (This patch basically adds entry/exits logs to Omx_rpc_skel.c and Omx_rpc_stub.c, adds file Omx_structures.c, omx_structures.h, which basically does parameter processing for structure and enum variables)
b) Build Ducati.
c) Collect logs(For more accuracy/to avoid missing log collect logs to file instead of console) Use [syslink_trace_daemon -l filename] to collect logs to filename.
d) Dependencies

Keep following files in the same directory as of callflowmap.pl

1. OMX_COMMANDTYPE.pl

2. OMX_EVENTTYPE.pl

3. OMX_nCONFIGINDEXTYPE.pl

4. OMX_TI_INDEXTYPE.pl

5.PortIndex.pl

e) Keep following in the same directory or enter path after script starts execution

Relative path of base_image_app_m3-winchester.xem3.map is

WTSD_DucatiMMSW/platform/base_image/package/cfg/out/app_m3/release

f) Run script callflow_map.pl

USAGE

Perl callflowmap.pl log_filename.txt where log_filename is file which contains logs.

g) Output is stored in filename_output.txt file in the same directory as of log_filename.txt

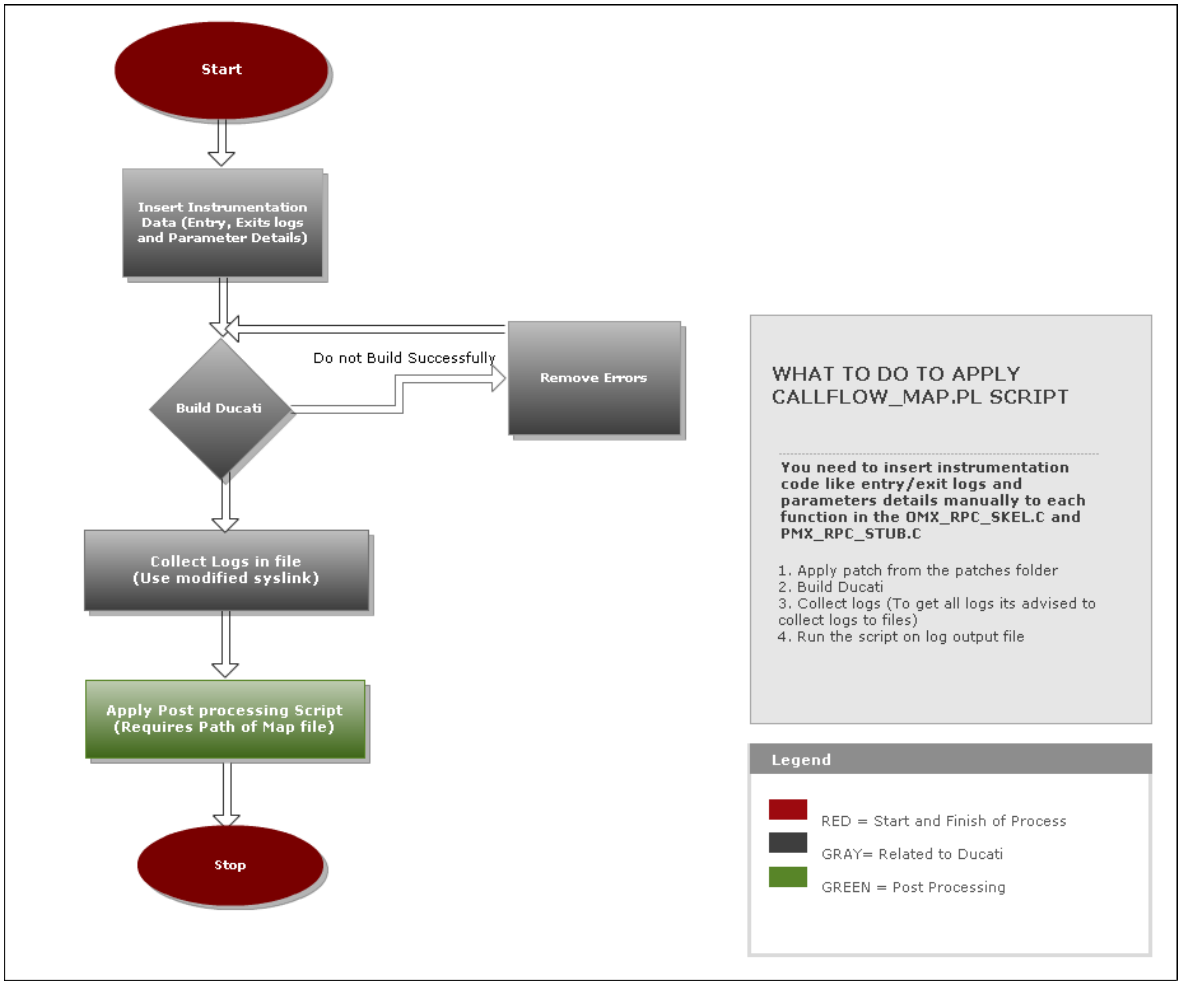
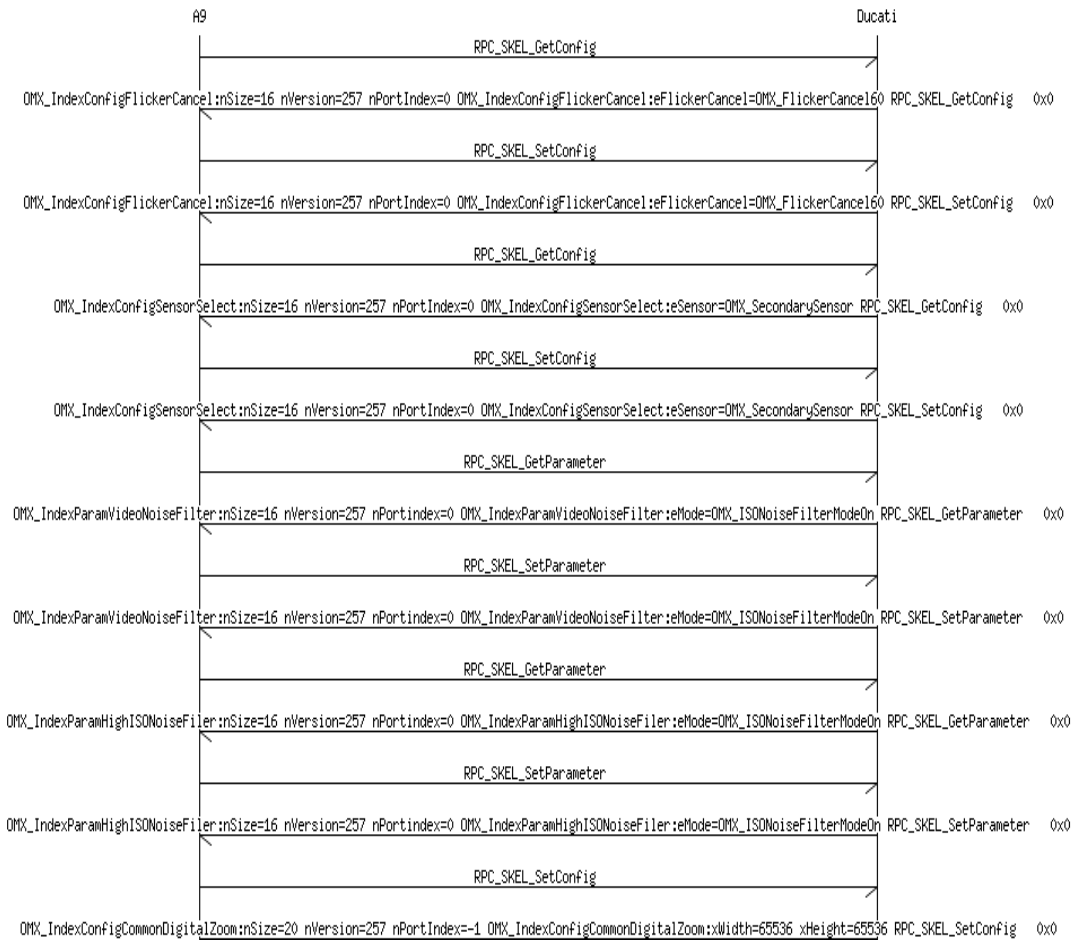

Start
Insert Instrumentation Data (Entry, Exits logs and Parameter Details)
Build Ducati
Do not Build Successfully
Remove Errors
Collect Logs in file (Use modified syslink)
Apply Post processing Script (Requires Path of Map file)
Stop
WHAT TO DO TO APPLY CALLFLOW_MAP.PL SCRIPT
You need to insert instrumentation code like entry/exit logs and parameters details manually to each function in the OMX_RPC_SKEL.C and PMX_RPC_STUB.C
1. Apply patch from the patches folder
2. Build Ducati
3. Collect logs (To get all logs its advised to collect logs to files)
4. Run the script on log output file
Legend
RED = Start and Finish of Process
GRAY= Related to Ducati
GREEN = Post Processing

**Figure 4- 7 Ducati System Analyser Phase II**

Figure 4-7 shows steps for Ducati System Analyser Phase II and Figure 4-8 shows sample output.



```
A9                                                                          Ducati
RPC_SKEL_GetConfig
OMX_IndexConfigFlickerCancel:nSize=16 nVersion=257 nPortIndex=0 OMX_IndexConfigFlickerCancel:eFlickerCancel=OMX_FlickerCancel60 RPC_SKEL_GetConfig 0x0
RPC_SKEL_SetConfig
OMX_IndexConfigFlickerCancel:nSize=16 nVersion=257 nPortIndex=0 OMX_IndexConfigFlickerCancel:eFlickerCancel=OMX_FlickerCancel60 RPC_SKEL_SetConfig 0x0
RPC_SKEL_GetConfig
OMX_IndexConfigSensorSelect:nSize=16 nVersion=257 nPortIndex=0 OMX_IndexConfigSensorSelect:eSensor=OMX_SecondarySensor RPC_SKEL_GetConfig 0x0
RPC_SKEL_SetConfig
OMX_IndexConfigSensorSelect:nSize=16 nVersion=257 nPortIndex=0 OMX_IndexConfigSensorSelect:eSensor=OMX_SecondarySensor RPC_SKEL_SetConfig 0x0
RPC_SKEL_GetParameter
OMX_IndexParamVideoNoiseFilter:nSize=16 nVersion=257 nPortindex=0 OMX_IndexParamVideoNoiseFilter:eMode=OMX_ISONoiseFilterModeOn RPC_SKEL_GetParameter 0x0
RPC_SKEL_SetParameter
OMX_IndexParamVideoNoiseFilter:nSize=16 nVersion=257 nPortindex=0 OMX_IndexParamVideoNoiseFilter:eMode=OMX_ISONoiseFilterModeOn RPC_SKEL_SetParameter 0x0
RPC_SKEL_GetParameter
OMX_IndexParamHighISONoiseFiler:nSize=16 nVersion=257 nPortindex=0 OMX_IndexParamHighISONoiseFiler:eMode=OMX_ISONoiseFilterModeOn RPC_SKEL_GetParameter 0x0
RPC_SKEL_SetParameter
OMX_IndexParamHighISONoiseFiler:nSize=16 nVersion=257 nPortindex=0 OMX_IndexParamHighISONoiseFiler:eMode=OMX_ISONoiseFilterModeOn RPC_SKEL_SetParameter 0x0
RPC_SKEL_SetConfig
OMX_IndexConfigCommonDigitalZoom:nSize=20 nVersion=257 nPortIndex=-1 OMX_IndexConfigCommonDigitalZoom:xWidth=65536 xHeight=65536 RPC_SKEL_SetConfig 0x0
```

Figure 4- 8 Output of Ducati System Analyser Phase II

### 4.2.1 *Limitations*

a. Static in nature may get useless if large change is made in the system.

b. Shows only the interface interactions of the Ducati system, it does not give the details of the call flow within Ducati.

c. Missing logs can alter the flow and can misguide the user.

## 4.3 Ducati System Analyser Phase-III

Though Ducati System Analyser-I is useful to understand Ducati code for beginners, it has its own limitations. It is clear that to understand the Control flow of Ducati we need to analyse the whole function call sequence (Message Sequence) for the use-case of interest.

The approach of manually inserting instrumentation code is not practical for following reasons

- Ducati Code contains more than 1000 files, it will be very time consuming and error prone activity. It's almost impossible.
- As files and functions are constantly changing, there will be need to keep track of these new functions and files.
- A single error in instrumentation will cause whole process to repeat.
- Changes in type of instrumentation code (Example instead of entry/exits logs, timing information needs to be added)

For these and many more reasons, the process of adding instrumentation code needs to be automated. Automating will not only make process faster but allows the changes in the instrumentation code. Also considering the size of Ducati source code repository and its dynamic nature (code gets added &/removed everyday), automatic logging would be very efficient tool for analysis purpose as you can modify instrumentation code.

Ducati System Analyser-III works in two phases

– *Automatic insertion of instrumentation code*

*(Refer figure 4-9 Ducati System Analyzer-Auto_instrumentation)*

This is completely developed in Perl, the script inserts the instrumentation code (entry and exit logs with additional information like task context etc...) are inserted in the source code. The script finds functions in the file along with entry/exit points. For each entry/exit point, logs will be inserted. These logs are actually print statement which helps us when function gets called and when it exits. There are some functions which have time constraints on execution like ISR’s (interrupt Service Routine), these functions must be excluded from the logging as it add runtime overheads. To enable this concept of function filter is introduced. The function filter file would contain functions, where logs should not be added. Like function_filter, file filter allows you to include/exclude files from logging.

- Positive filter file will contain file names in which logs are to be added; rest will be ignored from logging.
- Negative filter file will contain file names which are to be excluded from the logging; rest will be considered for logging.

The log statement can contain different information like function names, task ids, timing information etc. For our script we use two types of log statements

- Log statements without task id’s: These log statements contain only function names.

  Example:

  TIMM_OSAL_ErrorExt(TIMM_OSAL_TRACEGRP_SYSTEM,\"Ex:%08x\", $fun_name);
- Log statements with task id’s: These log statements contain function names as well as task ids.

  Example:

  TIMM_OSAL_ErrorExt(TIMM_OSAL_TRACEGRP_SYSTEM,\"Ex:%08x,%d\", $fun_name,\(unsigned int\) Task_self\(\)\);

- *Post Processing of Logs*

  *(Refer figure 4-10 Ducati System Analyzer-Post_processing)*

After adding log statements to the repository, and building, verifying on the hardware, logs should be collected for different use cases. To collect logs you need to run the syslink_trace_daemon (Trace system for Ducati) which by default writes output the console (screen). As logs with minimal errors are very important for the accuracy of the final output, following measures are employed to increase accuracy of logs

- Syslink_trace_daemon interrupt time reduced.
- Collecting logs to files instead of to the console as writing to file is faster than writing to console. (Syslink_trace_daemaon modified to by default write to file and not to console).

Logs collected can be processed by following two tools

- **Trace2UML**:

  This is an application which can convert a trace file in text format to UML sequence diagrams. The application can also modify the diagrams and save in the PDF, PNG format. Though searching can be done in text code, it's not efficient as it involves complex code. Graphical output produced by this is in the form of UML diagram which lets us find function calls and returns easily.

  Trace2UML can be downloaded from

  http://trace2uml.tigris.org/servlets/ProjectDocumentList?folderID=6208

  The features of Trace2UML:

  a. Reads special trace files which can easily be generated by any application.
  b. the trace can easily be edited by hand.
  c. draws nice (graphical) UML sequence diagrams.
  d. You can export in .pdf, .png and .svg.
  e. can modify the diagrams
  f. The source (trace file) is pretty usable as a debug trace, too.

g. The planned sequences and the recorded ones can be compared by a text diff.

– **MS Excel**

Spread sheet::WriteExcel module is can be used to create native Excel binary files. Formatted text and numbers can be written to multiple worksheets in a workbook. Formulas and functions are also supported. It is 100% Perl and doesn't require any Windows libraries or a copy of Excel.

Perl have support module which enables writing to Excel sheet from Perl script. Excel have its own advantages like searching is easy, lots of formatting options are available, high maximum size limitations etc.

Spread sheet::WriteExcel can be downloaded as the zipped tar file from one of the following:

http://search.cpan.org/search?dist=Spreadsheet-WriteExcel

http://theoryx5.uwinnipeg.ca/mod_perl/cpan-search?idinfo=154

http://ftp.funet.fi/pub/languages/perl/CPAN/authors/id/J/JM/JMCNAMARA/

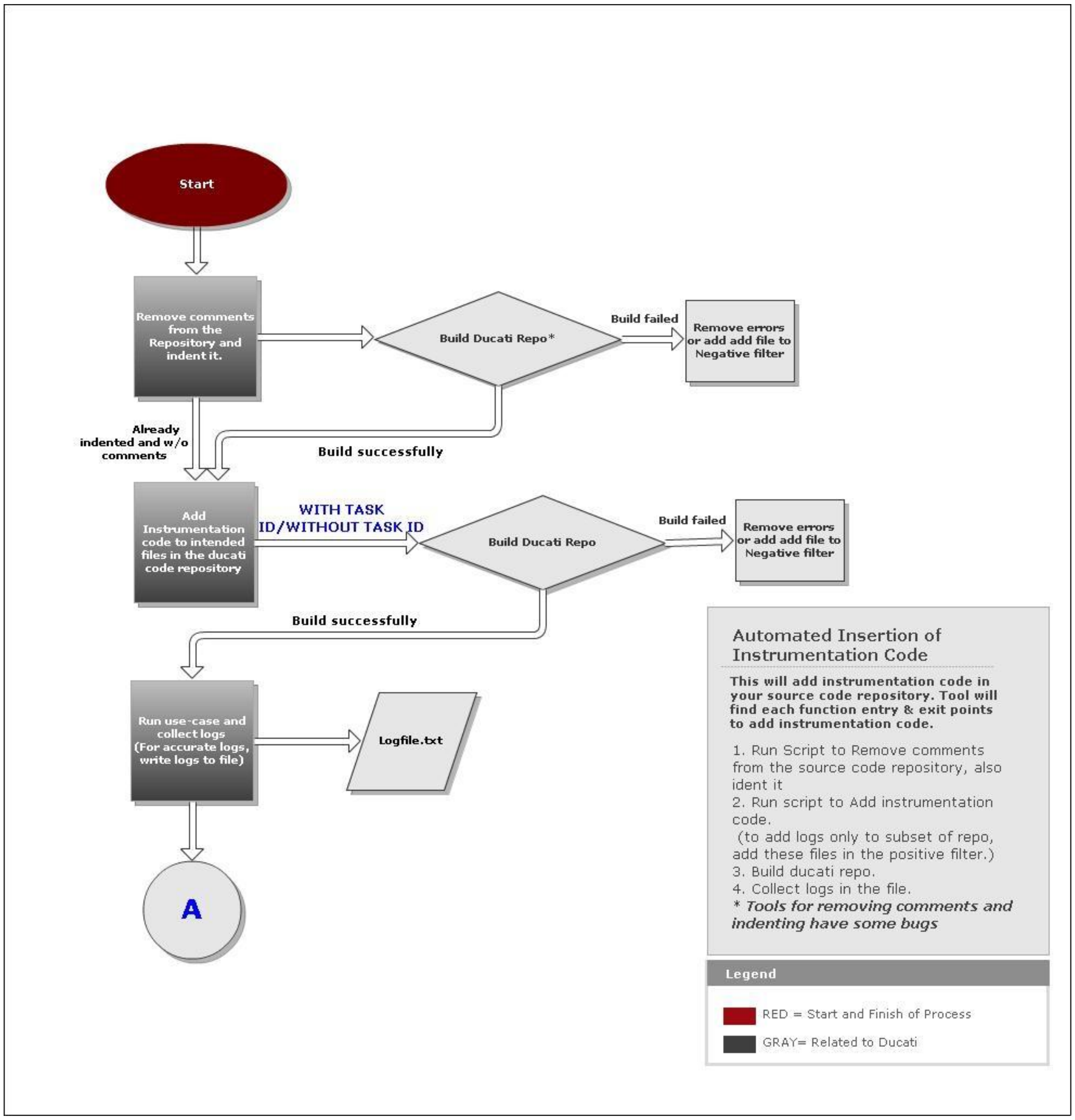

Start
Remove comments from the Repository and indent it.
Build Ducati Repo*
Build failed
Remove errors or add add file to Negative filter
Already indented and w/o comments
Build successfully
Add Instrumentation code to intended files in the ducati code repository
WITH TASK ID/WITHOUT TASK ID
Build Ducati Repo
Build failed
Remove errors or add add file to Negative filter
Build successfully
Run use-case and collect logs (For accurate logs, write logs to file)
Logfile.txt
A
Automated Insertion of Instrumentation Code
This will add instrumentation code in your source code repository. Tool will find each function entry & exit points to add instrumentation code.
1. Run Script to Remove comments from the source code repository, also ident it
2. Run script to Add instrumentation code.
(to add logs only to subset of repo, add these files in the positive filter.)
3. Build ducati repo.
4. Collect logs in the file.
* Tools for removing comments and indenting have some bugs
Legend
RED = Start and Finish of Process
GRAY= Related to Ducati

**Figure 4-9 Ducati System Analyser-III Auto instrumentation**

Figure 4-9 and 4-10shows steps for Ducati System Analyser Phase II and Figure 4-11 shows sample Trace3UML output and Figure 4-12 shows sample MS Excel output.

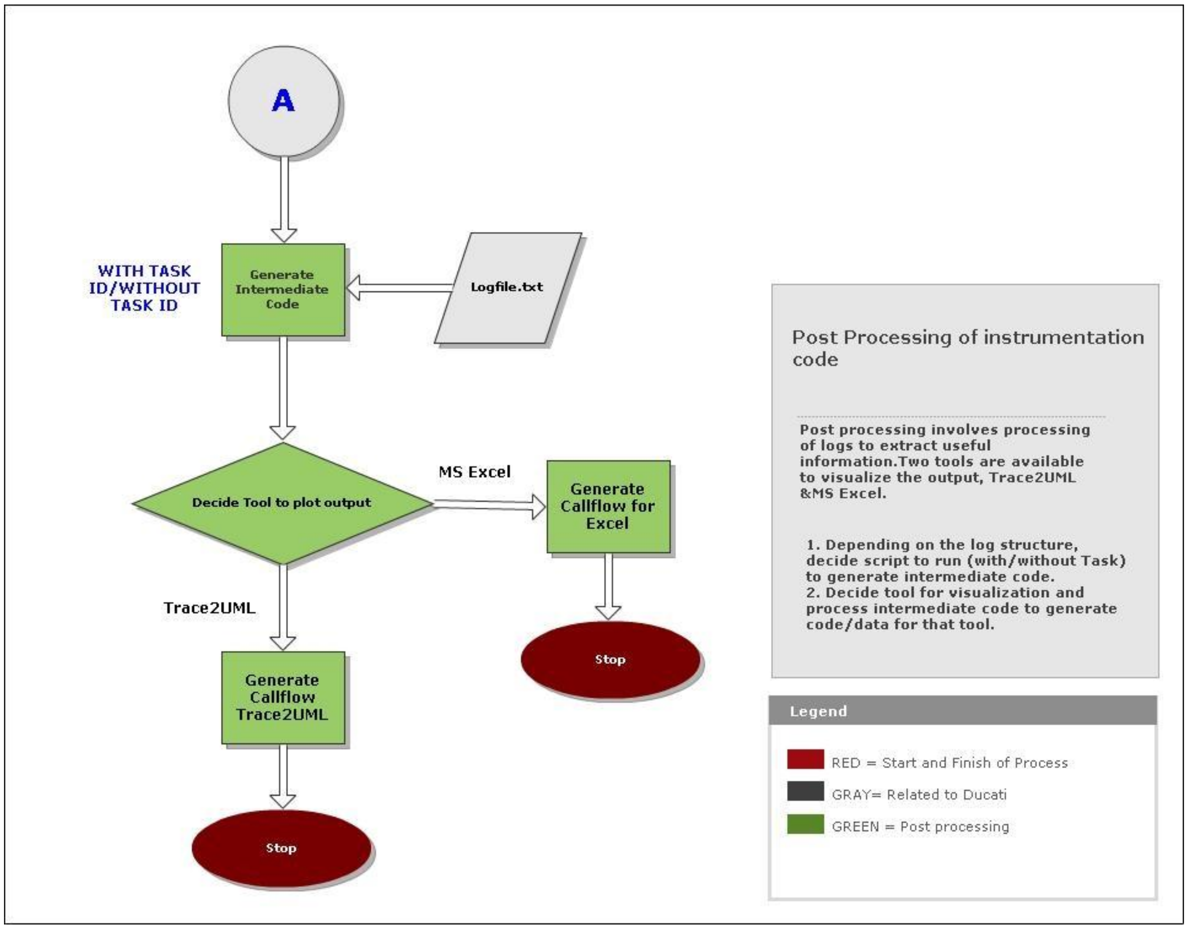

A
WITH TASK ID/WITHOUT TASK ID
Generate Intermediate Code
Logfile.txt
Decide Tool to plot output
MS Excel
Generate Callflow for Excel
Trace2UML
Generate Callflow Trace2UML
Stop
Stop
Post Processing of instrumentation code
Post processing involves processing of logs to extract useful information.Two tools are available to visualize the output, Trace2UML &MS Excel.
1. Depending on the log structure, decide script to run (with/without Task) to generate intermediate code.
2. Decide tool for visualization and process intermediate code to generate code/data for that tool.
Legend
RED = Start and Finish of Process
GRAY= Related to Ducati
GREEN = Post processing

**Figure 4-10 Ducati System Analyser-III Post Processing**

Steps for Ducati System Analyser Phase II (Automatic of Instrumentation and post processing of logs)

a) The script to insert code operates on the indented and uncommented (All Comments removed). So to make Ducati Source Code Repository indented and uncommented we need to apply different tools like cloc-1.53.exe and indent.exe. Another script has been written to apply these tools on the repo. (Dependencies/indent_remove_comments.pl).

b) The auto_log script can be applied in two modes

   I. With task id's

   II. Without task id's

Select whether you want to add log statements with task id's or without it. Apply corresponding Perl script.

c) Build Ducati repository.

d) Run use-case of interest and collect logs (preferably to file directly).

e) Generate intermediate code from the logs.

f) Now depending on the tool you want to use to visualize output apply Genarate_Trace2UML or Generate Excel.

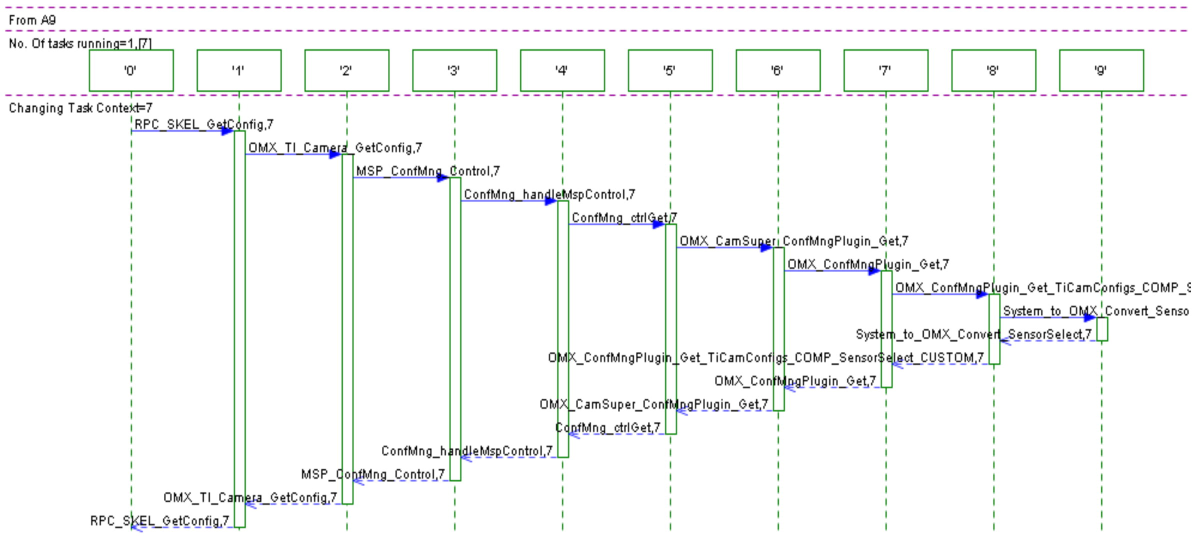

From A9
No. Of tasks running=1,[7]
'0'
'1'
'2'
'3'
'4'
'5'
'6'
'7'
'8'
'9'
Changing Task Context=7
RPC_SKEL_GetConfig,7
OMX_TI_Camera_GetConfig,7
MSP_ConfMng_Control,7
ConfMng_handleMspControl,7
ConfMng_ctrlGet,7
OMX_CamSuper_ConfMngPlugin_Get,7
OMX_ConfMngPlugin_Get,7
OMX_ConfMngPlugin_Get_TiCamConfigs_COMP_S
System_to_OMX_Convert_Senso
System_to_OMX_Convert_SensorSelect,7
OMX_ConfMngPlugin_Get_TiCamConfigs_COMP_SensorSelect_CUSTOM,7
OMX_ConfMngPlugin_Get,7
OMX_CamSuper_ConfMngPlugin_Get,7
ConfMng_ctrlGet,7
ConfMng_handleMspControl,7
MSP_ConfMng_Control,7
OMX_TI_Camera_GetConfig,7
RPC_SKEL_GetConfig,7

Figure 4- 11 Ducati System Analyser- Generate_Trace2UML

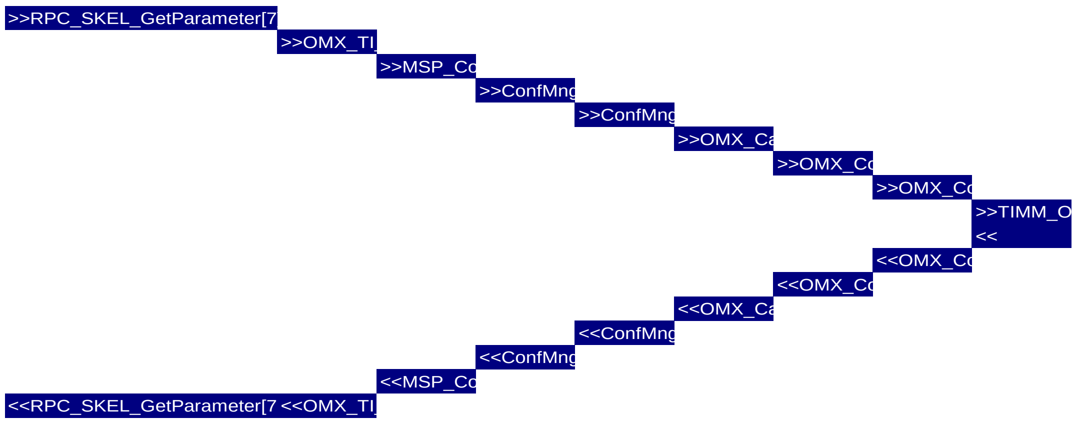


Figure 4- 12 Ducati System Analyser- Generate Excel

## 4.4 Challenges

### *4.4.1 Code section not fitting in the available memory*

As we added tremendous no. of instrumentation statements in the Ducati Source repository, it caused code section to be blown up. Ducati is slave processor and host processor has allocated limited memory from main memory to this slave processor. So we add our instrumentation code to the Ducati repo it causes code section to exceed allocated size (4MB) which causes compile time crash.

Remedy to above problem is

- Limit no. of log statements.
- Reduce size of log statement.
- Remove some unnecessary (for analysis) code from repository.

We applied all three measures to tackle the problem of low memory. We, instead of printing string for the function name, printed address (Address takes 4 bytes only compared to string which depends on string length). Also we removed some codecs like MPEG decoder etc. to free the memory. Though this caused change in functionality of Ducati, for analysis and optimization purpose it was perfectly OK, as this codecs are not used in the video chat.

### 4.4.2 Missing logs

We used syslink_trace_daemaon to collect the logs but as this was not configured for such tremendous amount of logs it tended to loose logs. Even logs were not 50 % accurate. For analysis purpose we needed more accurate logs as otherwise it would alter whole control flow. We modified syslink in following ways.

- Modified interrupt time from 1 Second to 1 mSecond.
- Instead of printing to console we printed logs to file as printing to file is much faster than printing to console.

With above two solutions we got logs with almost 98 % accuracy.

### 4.4.3 Logical problems for script to detect functions

Considering size of Ducati Source code repo (over 4 million lines of code), it was challenging to operate script to find function definitions. Function can be on single line, can be on multiple lines, can contain function pointers, can have function as a parameter, and can be long enough to occupy whole line and many more. This caused script to have logic to deal with all above conditions making it more complex to understand.

### 4.4.4 Support for information filtering

Many times we are interested in analysing a limited domain of whole repository, such as limited files, limited functions etc. And to support this feature, tool should have ability to filter. Filtering may be applied to files, or functions.

We added following types of filters to aid in limited analysing

- Positive filter- Add instrumentation code only to the specified files.
- Negative filter- Don’t add instrumentation code to the specified files.

- Function filters- Functions to be excluded. Example- Functions corresponding to HWI, SWI etc.

## 4.5 Recommendations for optimization

In the context of performance, there is a distinct correlation between memory usage and efficiency. The more memory application occupies, the more inefficient it is going to be. More memory means more memory allocations, more code, and a greater potential for paging.

The focus of this tool is on the reduction of executable code. Reducing code footprint is not just a matter of turning on code optimizations in compiler, although that does help. Code can also be reduced by organizing code so that only the minimum set of required functions is in memory at any given time. This optimization can be implemented by profiling code. The main aim of all implementation done is to enable knowledge gain for the purpose of optimization and analysis. Also as a side product it can help new developer to Ducati to learn Ducati and understand control flow at interface level or at a system level by studying the message sequence diagrams.

Optimizations can be made in following

a. Code size reduction.
b. Macro overhead.
c. Function overhead.
d. Use of cache/in built memory for frequently used functions.
e. Replacing function by macro.
f. Making function inline.

### 4.5.1 Code Size Reduction:

As we have seen in the section 4.1 in which we removed all dead code (code between #if 0 and #endif) redundant macros (hash defines), structures, functions, and enumerations and also found out structures with similar name, similar content etc. This has benefited us in two ways, firstly it reduced code size making code more compact, understandable and secondly it removed some runtime overhead (as during this analysis we replaced concatenated macros, function calls and structure by actual names). We found unused functions, structures and removed them. As pointed out in the section 4.1 nearly 20 percent code was redundant. During this many interesting coding practices were seen which if possible can be optimized. Some of them are include list dependency, concatenated macros, function calls, structures etc.

### 4.5.2 Macro overhead

A macro is a rule or pattern that specifies how a certain input sequence should be mapped to a replacement input sequence according to a defined procedure. The mapping processes that instantiates (transforms) a macro use into a specific sequence is known as macro expansion. Such macro expansion has compile time overhead and if multiple levels of macro's are used it can consume more compile time as it involves replacing chain of macros. Also concatenation of macro can cause some compile time overhead if multiple levels are involved. So it is necessary to analyse and if possible optimize such macros. The first phase of Ducati System Analyser provides data which can be used to analyse and find macros involving chain, concatenations etc.

### 4.5.3 Function Overhead

Function, is just a piece of code which can be referred using a name. The computer stores the data of the function (such as arguments, local variables, return address etc.,) when a function is invoked and destroys them when the function returns. Similarly when a function calls other functions, the caller function returns only after all the callee functions are returned. We can infer a LIFO (last-in-first-out) scenario in this, i.e., the function that is invoked last completes first and the one invoked first completes last. So the stack becomes an obvious choice for function calls implementation.

In the memory, the stack grows from the higher memory address to the lower memory addresses. So we have to visualize an inverted stack.

The main difference between macro and function is that in case of macro, whatever you define as a macro gets verbatim posted into code, before the compiler gets to see it, by the pre-processor. Compiler is not aware of existence of macro. But in case of function compiler knows existence of function and has to do some work in order to enable function call mechanism.

For example to call a function such as foo (6, x+1)...

- Evaluate the actual parameter expressions, such as the x+1, in the caller's context.
- Allocate memory for foo ()'s locals by pushing a suitable "local block" of memory onto a runtime "call stack" dedicated to this purpose. For parameters but not local variables, store the values from step (1) into the appropriate slot in foo ()'s local block.
- Store the caller's current address of execution (its "return address") and switch execution to foo ().
- foo () executes with its local block conveniently available at the end of the call stack.
- When foo () is finished, it exits by popping its locals off the stack and "returns" to the caller using the previously stored return address. Now the caller's locals are on the end of the stack and it can resume executing.

So from this we can see that function call causes run-time overhead on system. By analysing functions in the system, we can optimize it in following ways

#### *4.5.3.1 Use of cache/in built memory for frequently used functions.*

With the help of function list produced in the Ducati System Analyser phase III, we can find no. times a function get called for particular use-case, a most frequently used function, least frequently used function, average used function and many more. With this information we can place these most frequently used functions in the cache/build in memory so as to memory access time. It helps in putting right function in the cache.

#### *4.5.3.2 Replacing function by macro*

If we want to reduce function call overhead, we can replace such function by macro. To replace function by macro we need to analyse function as all functions can’t be replaced by macros. Constraints to replace macros by functions are macros can't be pointed to and we can give functions as parameters to other functions, but not macros.

If these constraints are not applicable you can replace function by macro to avoid function call overhead.

#### *4.5.3.3 Making function inline*

Inline function is the optimization technique used by the compilers. One can simply prepend inline keyword to function prototype to make a function inline. Inline function instruct compiler to insert complete body of the function wherever that function got used in code.

Advantages:-

a. It does not require function calling overhead.
b. It also save overhead of variables push/pop on the stack, while function calling.
c. It also save overhead of return call from a function.
d. It increases locality of reference by utilizing instruction cache.

## 4.6 Use of Open Source Tools

During the all three development phases of Ducati System Analyser I have used many open source tools. These tools are freely available in market either in source code or executable/binary files. In this section these open source tools are discussed like their use, location for downloading, functionality etc.

### 4.6.1. Cloc 1.53

Cloc counts blank lines, comment lines, and physical lines of source code in many programming languages. Given two versions of a code base, cloc can compute differences in blank, comment, and source lines. It is written entirely in Perl with no dependencies outside the standard distribution of Perl v5.6.

Cloc has many features that make it easy to use, thorough, extensible, and portable:

a. Exists as a single, self-contained file that requires minimal installation effort---just download the file and run it.
b. Can read language comment definitions from a file and thus potentially work with computer languages that do not yet exist.
c. Allows results from multiple runs to be summed together by language and by project.
d. Can produce results in a variety of formats: plain text, SQL, XML, YAML, comma separated values.
e. Can count code within compressed archives (tar balls, Zip files, Java .ear files).
f. Has numerous troubleshooting options.
g. Handles file and directory names with spaces and other unusual characters.
h. Have no dependencies outside the standard Perl distribution.
i. Runs on Linux, FreeBSD, NetBSD, Mac OS X, AIX, HP-UX, Solaris, IRIX, and z/OS systems that have Perl 5.6 or higher. The source version runs on Windows with either ActiveState Perl or cygwin. Alternatively on Windows one can run the Windows binary which has no dependencies.

We used cloc for removing comments (c as well as cpp style) from the source code repository.

Cloc can be downloaded from

http://sourceforge.net/projects/cloc/files/latest/download?source=files

### 4.6.2 Indent.exe

The indent program changes the appearance of a C program by inserting or deleting white-space. It can be used to make code easier to read. It can also convert from one style of writing C to another. Indent understands a substantial amount about the syntax of C, but it also attempts to cope with incomplete and misformed syntax.

Indent.exe can be downloaded from

http://indent.isidore-it.eu/indent-2.2.11.tar.gz

### 4.6.3 Trace2UML.

This tool is used to draw graphical output. Details regarding this are presented in the section 4.3

### 4.6.4 Spreadsheet-WriteExcel-2.37.

This tool is used to draw graphical output for task based output. Details regarding this are presented in the section 4.3

# Chapter 5 Improvements and Impact

In the Ducati System Analyser I presented an approach to support the understanding of software. It merges results from static program analysis with dynamic information about program execution. This combined information is presented visually with graph structures. I argued that neither static nor dynamic views by themselves provide an adequate understanding of software systems. Large software systems require a reduction of information displayed at a time. I discussed filtering and aggregation techniques. The former reduces the data computed in the static analysis the latter reduces the information in a view while the data is still present in the view model. Both techniques are applied in our approach. I instantiated two frameworks, one for analysing static information from program representations , the other for extracting dynamic information from program executions and for the visualization of information (Trace2UML and MS Excel). The resulting tool, the Ducati System Analyser, supports the analysis of design flaws and supports the analysis phase of the reengineering process in software systems. In the future, the architecture could be used as a visual tool to also control source code modifications.

## 5.1 Impact on Code size

During the static analysis phase of Ducati System Analyser, we removed redundant structures, enum, hash defines, dead code, unused functions, unused files like test files etc. It caused reduction in code size of repository making it compact and understandable.

Figure 5.1 shows comparison of code size for all these data structures.

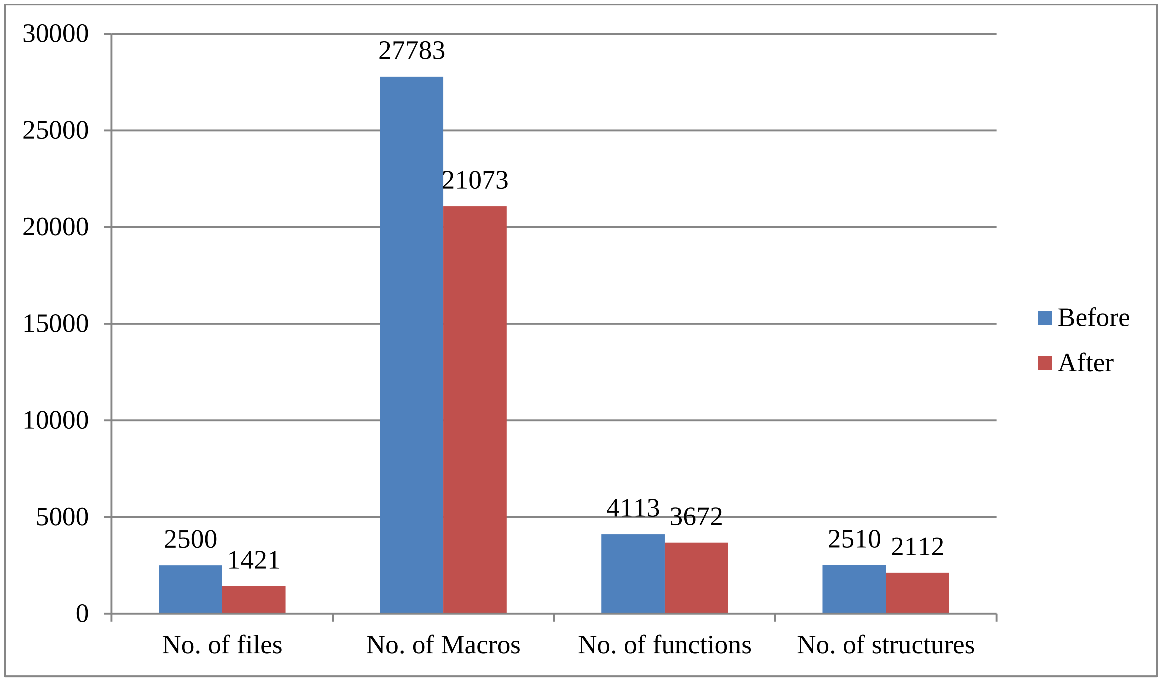


Figure 5 1 Ducati System Analyser-Impact on code

## 5.2 Improvements in code understanding and readability

Though improvements in code understanding and readability are something that doesn't have a metric, we can say that making code compact makes it more readable. , As with the help of code size reduction, we have made code repository so after static analysis code become more readable. Now from understanding point of view, as we have implemented dynamic profiling mechanism it can be used for getting insight into the code. Also it can aid code walkthrough. Understanding was hard in case of complex repositories like Ducati because of following reasons

- Ducati is controlled from the A9 by a bunch of about 15 APIs (OMX) but not many really understand what level of complexity is hidden behind these 15 APIs. Interaction with the host processor involves IPC mechanisms and API call sequence
- There are at least 15-18 tasks running in the system, function depths can go up to 85.

- Task synchronization is implemented by a bunch of very complex synchronization mechanisms.
- Ducati source code is over 3 million lines of code containing thousands of files.
- .Even though some documentation exists for Ducati, there is changes/adaptations that are not captured in the documents to reflect the current state of the code/design.
- Some parts of the Ducati source code are quite difficult to understand, analyse and optimize.
- Code is not readable otherwise, due to the use of abstraction interfaces like MSP, usage of ## (concatenation macro) kind of things for function names and also due to use of function pointers.
- Because of above reasons some support tool was needed which is provided by Ducati System Analyser. This implantation has provided an opportunity to depict all complex call sequences, tasks etc. pictorially through Trace2UML diagrams as well as through Excel sheets providing an extremely easy interface to understand the code and thereby improving the readability.

## 5.3 Impact on efficiency

For video chat use-case, from the tool output it can be seen that some functions are very frequently used even some functions are used for almost 80% of time. If we put these functions in the cache or make it inline/macro, significant improvement in the performance can be seen. For example- for video chat use case frame rate can increase from 25 fps to 30 fps (which in itself is major performance benefit). This makes video chat efficient in power as well as in performance. By analysing message sequence diagram, optimizations can be made in by reducing function call stack depth (maximum size of stack for the use-case of interest). Many times stack depth grows as large as 85 meaning a to return to main caller 85 new function calls are made, which have very high performance impact as it consumes memory and also requires task switches. As video chat involves many components like camera, video encoder, video decoder, microphone etc. The output generated by tool not only helps to optimize video chat but any other use-case like video decoding, image capture etc. In addition to functions many other data structures can be optimized. For example removing macro concatenation, nesting of macro can save compile time. To make video chat use case of Ducati use-case efficient in performance and power, it’s not sufficient to optimize only algorithms but whole system. To optimize whole system we need some tooling which can provide support data. With the help of this support data optimizations are made. For example we can keep frequently occurred data in the local memory and can see significant increase in performance. Most of unused code in the code repository unnecessarily complicates the code repository. This code if removed can make repository understandable and readable. To understand complex system where proper documentation does not exists, it’s needed to have some dynamic tooling which can generate information at run-time. With the help of these tooling algorithms can be modified to make it efficient in power and performance.

Following points summarize impact on efficiency

- opportunity to optimize and improve the efficiency
- Less code, more efficient code translates to faster code and lower power consumption.
- The knowledge of the most used functions in the code at runtime video chat application can help in deciding which functions to optimize and also help in determining the functions that should be placed in the internal RAM etc. etc..
- In addition to the above, all of the things like low latency encoder, LTRP, system wide optimization using Ducati system analyser have helped implement a very efficient video chat solution.

# Chapter 6 Conclusion and Future Work

In this chapter, the detailed review of each chapter of the thesis is presented, the results of the thesis are summarized, applications of the work are discussed, contributions to the published literature are mentioned and the future directions of the work are described.

## 6.1. Review of the Thesis

In chapter 1 introduction to project/thesis is given. In the introduction I have discussed motivation for the work also revised some basic concepts of video chat, video conferencing, video streaming etc. In cellular video chat in there are many challenges and inefficiencies like as high error rate, bandwidth variation and limitation, battery power limitation. Next review of video chat concepts like Adaptable Video Chat in cellular mobile network, QoS Control is covered. Also basic terms involved in the video chat are covered like QoS parameters, their definitions, trade-offs between these parameters, need of Adaptable Video Chat, network constraints like bandwidth and  The overall goal of this thesis is to study extensively existing framework for video chat in the Ducati and optimize depending on the experimental results.

In chapter two introductions to OMAP and Ducati subsystem is given. OMAP (Open Multimedia Access Platform) processors are developed by the Texas Instruments. OMAP is a category of proprietary system on chips (SoCs) for portable and mobile multimedia applications. OMAP4 is the fourth generation SOC that is used in many of today's high performance and low power multimedia devices- which include smart phones, tablets, video conferencing solutions and other consumer products. OMAP4 enables true High Definition multimedia (1080p), very high performance graphics and yet offers a low power advantage over other SOCS. Ducati Sub System comprises of two ARM® Cortex-M3 processors (CPUs), IVA-HD subsystem and ISS subsystem. The two Cortex-M3 processors are known as Sys M3 and App M3 and they comprise the MPU of Ducati Subsystem.

Chapter 3 presented literature review contains recent work done in the field of Video chat; video streaming also described the key research concepts of the work. Literature review covers existing methodologies used in TI as well as work done in the open conferences. Many algorithms/techniques are studied and it also described the strengths and limitations of the all techniques/algorithms.

Chapter 4 described the implementation work done for the analysis of all these algorithms. The whole work is carried in three phases. In the first phase all source code repository scanned to remove all redundant code and make repository understandable and analysable. In the second phase I implemented tool for run-time analysis at interface level. In the third phase tool’s functionality extended to system level. Now analysis can be done at system level by looking at the various function call overheads, timing parameters and many more depending on the instrumentation data. Also challenges are discusses along with optimization scope.

Chapter 5 covers improvements and impact of the work carried out. Topics like impact on code size, impact on code understanding and impact on the efficiency are elaborated. Also improvements seen after applying optimizations are shown.

## 6.2. Summary of results/Conclusion

To optimize any specific functionality (read video chat) complex system it’s not sufficient to optimize only part of system which is carrying out this functionality butt whole system itself. Same applies to Ducati Video Chat use-case, while trying to optimize and improving efficiency for video chat, it was not sufficient only to optimize only algorithms involved but whole system itself. So in this project I studied all the video chat algorithm s and implemented tool to analyse performance of these algorithms and also provided data to aid optimization to make video algorithms faster (It also have helped to make other use-cases like still-image capture faster). Also it helped to make code base understandable, readable for any new as well existing developers.

I have presented an approach to software visualization supporting the understanding of structure and behaviour of software systems. To do so, I merge information from static program analysis with dynamic information obtained during the execution of the programs. The merged information is presented graphically in different views, where users can interactively choose between more abstract or more concrete graphical views. This technique is implemented in an analysing tool; its open architecture enables the easy integration of new views such as metric or profiling views. Using these techniques, tool detects components and interaction patterns in code.

The results obtained from the analysis of repository can be summarized as follows:

a. Almost 25 % of code in the repository is useless. Either it is in the form of unused hash defines (macros), structures, enums, functions, or it may be code enclosed in between conditional compilation block which will never execute (code between #if 0 and #endif).

b. For video chat use-case, from the tool output it can be seen that some functions are very frequently used even some functions are used for almost 80% of time. If we put these functions in the cache or make it inline/macro, significant improvement in the performance can be seen.

c. For example- for video chat use case frame rate can increase from 25 fps to 30 fps (which in itself is major performance benefit). This makes video chat efficient in power as well as in performance.

d. By analysing message sequence diagram, optimizations can be made in by reducing function call stack depth (maximum size of stack for the use-case of interest). Many times stack depth grows as large as 85 meaning a to return to main caller 85 new function calls are made, which have very high performance impact as it consumes memory and also requires task switches.

e. As video chat involves many components like camera, video encoder, video decoder, microphone etc. The output generated by tool not only helps to optimize video chat but any other use-case like video decoding, image capture etc.

f. In addition to functions many other data structures can be optimized. For example removing macro concatenation, nesting of macro can save compile time.

g. To make video chat use case of Ducati use-case efficient in performance and power, it’s not sufficient to optimize only algorithms but whole system. To optimize whole system we need some tooling which can provide support data. With the help of this support data optimizations are made. For example we can keep frequently occurred data in the local memory and can see significant increase in performance.

h. Most of unused code in the code repository unnecessarily complicates the code repository. This code if removed can make repository understandable and readable.

i. To understand complex system where proper documentation does not exists, it’s needed to have some dynamic tooling which can generate information at run-time.

j. With the help of these tooling algorithms can be modified to make it efficient in power and performance.

k. In the multiprocessor system where more than single processor is involved in carrying out a single task, it is necessary to understand interaction of the host processor and slave processor. This involves complex IPC mechanisms and API call sequences.

## 6.3. Summary of the Results

In this work, I have focused upon optimizing code for efficient video chat use case in the OMAP enabled mobile phones. I have basically emphasized upon generating a message sequence diagrams for video chat use case (can be generated for other use cases) using these two techniques. In the first only interface level message sequence diagrams are drawn and in the second whole system level message sequence diagrams are drawn. I have applied knowledge gained from these message sequence charts to optimize source code repository. I reduce the additional effort that is required to generate these charts each time by automating insertion of instrumentation code. Moreover, different tools are used to visualize output generated.

## 6.4. Application of the Work

After design and evaluation of the implemented tools for generating message sequence charts, we can conclude that the work in this thesis will allow researchers in video chat for OMAP and developer to:

a) Use the implanted tool to efficiently and quickly generate support data for optimization.
b) Reduce time and effort in testing as from generated date results can be compared.
c) Adequately generate effective and efficient mechanism to understand source code repository.
d) Removing redundant source code and making repository clean.
e) Save time while generating message sequence charts in a reasonable amount of time.

## 6.5. Contribution to Published Literature

For research publication purpose, there are a lot of prestigious TI internal conferences I will target to present to internal conferences first. Examples of TI internal conferences are TIITC (Texas Instruments India Technical Conference (happens in the Nov-Dec timeframe) and TI Software

Symposium. The reviewers of the above conference will recommend other conferences where this can be presented. If I do come up with some good solutions, they can be patented also provided there is enough scope, novelty and revenue potential.

## 6.6. Future Work

While the analysis results shown in this work are encouraging, further analysis would be useful and would add to the strength of the proposed tools. Future directions involve use of these tools for different video chat algorithms and analysing results. Also these tools can be used for debugging which involves making syslink_trace_daemaon robust so that it would not miss a single log on more larger and complex instrumentation codes. Also different tools can be tried to present output in the graphical format. Static analysis of code can be extended to find functions, structures having similar contents. Also to save compile time unnecessary inclusion of header files can be found.

Following support can be added to tool

- Cache hit/miss ratio
- Memory analysis.
- OS Resource utilization
- Peak stack usage (per Task)
- Peak heap usage.

## ***References***

[1] Etoh M. Yoshimura T., Advances in Wireless Video Delivery, Proceedings of the IEEE Jan. 2005

[2] Mir Md. Jahangir Kabir, Md. Shaifullah Zubayer, Zinat Sayeeda, Real-time video chatting in low bandwidth by Facial Action Coding System, Proceedings of 14th International Conference on Computer and Information Technology (ICCIT 2011) 22-24 December, 2011, Dhaka, Bangladesh

[3] Karthik Ramanan, Rob Clark, Mukund Mittal, Innovations and challenges in ramping up the first OMAP4 product – Playbook, TIITC2011

[4] Alireza Goudarzi Nematiy and Makoto Takizawa,Application Level QoS in Multimedia Peer-to-Peer (P2P) Networks, 22nd International Conference on Advanced Information Networking and Applications – Workshops

[5]OMAP Technical Reference Manual-(http://www.ti.com)

[6] Dirk Heuzeroth, Understanding a System's Architecture, University of Karlsruhe IPD, Program Structures group

[7] Next Gen multimedia architecture: OpenMAX IL on Multi-Processor Platforms (TIITC2009)

[8] Takashi Oshiba and Kazuaki Nakajima, Quick and simultaneous estimation of available bandwidth and UDP throughput in real time communication, 978-1-4577-0681-3/11/$26.00 ©2011 IEEE

[9] Bharat Bhargava, Adaptable Software for Communications in Video Conferencing, Application-Specific Software Engineering Technology, 1998. ASSET-98. Proceedings. 1998 IEEE Workshop

[10] Phanish and others, Enabling Low Latency Processing Path during Video Record UC at OMX Level, Texas Instruments India Technical Conference 2011 (TIITC2011).

[11] Mugdha Kamoolkar, Ravindranath Andela, and SysLink: Use case driven design and performance optimizations, Texas Instruments India Technical Conference 2010 (TIITC2010).

[12] Aditya Monga, Sarthak Aggarwal, Sudhir Kasargod, Adapting TI Multimedia solution for Generic HLOS frameworks, Texas Instruments India Technical Conference 2010 (TIITC2010).

[13] Mahant Siddaramanna, Naveen Srinivasamurthy and Yashwant Dutt , Optimizing video encoders for low delay and channel constrained transmissions, Texas Instruments India Technical Conference 2010 (TIITC2010).

[14] Karthik Ramanan Ducati Software Training

[15] OpenMAX IL Specification Document.