"IMPLEMENTATION OF REMOTE VIDEO SURVEILLANCE SYSTEM ON CRADLE'S SoC"

A Major Project Report

Submitted in Partial Fulfillment of the Requirements for the Degree of

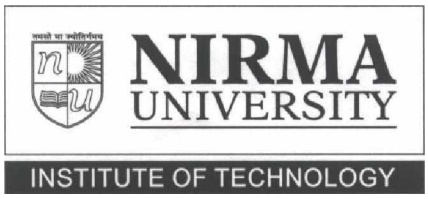
MASTER OF TECHNOLOGY

IN

ELECTRONICS & COMMUNICATION ENGG.

(VLSI Design)

By
Batchu Venkata Durga Mahesh
(03MEC004)



Department of Electronics & Communication Engineering
INSTITUTE OF TECHNOLOGY
NIRMA UNIVERSITY OF SCIENCE & TECHNOLOGY,
AHMEDABAD 382 481

MAY 2005

CERTIFICATE

This is to certify that the Major Project Report (Part-I-II) entitled "**Implementation of remote video surveillance system on Cradle's SoC**" submitted by Mr.Batchu Venkata Durga Mahesh (Roll No.03MEC04) towards the partial fulfillment of the requirements for Semester III-IV of **Master of Technology (Electronics & Communication Engg.)** in the field of **VLSI Design** of **Nirma University of Science and Technology** is the record of work carried out by him under our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for examination. The results embodied in this major project work to the best of our knowledge have not been submitted to any other University or Institution for award of any degree or diploma.

Date:

Project Guide:

Mr. Mohammed M.S.
Manger Execution,
CG-CoreEl Logic Systems Ltd.,
Pune

Facilitator at Institute:

Asst. Prof. Dhaval Pujara
Dept. of Electronics & Comm. Engg.
Institute Of Technology,
Nirma University, Ahmedabad

Dr. M. D. Desai

HOD

Dept. of Electronics & Comm. Engg.
Institute of Technology,Nirma University

Dr. H. V. Trivedi

Director

Institute of Technolgy,
Nirma University, Ahmedabad

Signature of Examiners:

ACKNOWLEDGEMENT

It gives me a great pleasure to take this opportunity to thank to **CG-CoreEl logic systems Ltd., Pune** and **Mr. T.K.Mukharjee** for giving me such a great opportunity to do project in their esteemed organization. I deem it my privilege to have carried out this dissertation work under this well-known quality conscious organization.

I express my deep sense of gratitude to **Mr.Madhukant Patel** and **Mr. Mohammed M.S.** for their personal involvement in every facet of this work and readiness to resolve any point of confusion by mutual discussion and for providing the necessary facilities for carrying out this work.. I would also like to thank **Mr. Tushar Patil** and **Mr. Sumit** for their co-operation.

I would like to thank **Asst. Prof. Dhaval Pujara** for his help, valuable suggestions and moral support. Finally, I would like to thank my parents and all my family members for their constant love and support and for providing me with the opportunity and the encouragement to pursue my goals. At last I will also like to thank all my colleagues for their support during the work.

(Batchu Venkata Durga Mahesh)

Abstract

Video processing loads have driven processors to consider a wide range of architectural approaches to deal with the sheer volume of pixel data. As the preferred medium for communication and entertainment, video resolutions and frame rates show no signs of leveling off. Processor clock speeds, while having dutifully risen per Moore's predictions, still can't keep up with ever increasing demands of the video marketplace. Similarly, compression algorithms, despite an order of magnitude jump in network capacities every couple years and the rapid drop in storage costs have only continued to rise in implementation complexity.

This paradox has brought about a variety of architectural approaches to tackle throughputs demanded by video processing applications. However, few can scale to the unending rise in processing demands. Traditional DSP/CPU architectures increase clock rate and multiple issue rates at the rate defined by Moore's law. But clock speedups have a limited potential as Moore's law hits a breaking point. With that realization, several designers have begun the transition to multi-core multiprocessor architectures. Indeed, multiprocessor chips are becoming more common because of advantages of scalability of design.

This project describes the implementation of remote video surveillance on cradle 3SoC. It is a complete solution for a surveillance system of national importance and first of its kind in India. It is a complete wireless system in the range of 5 kms.

This project highlights the advantages of dedicated devices in the System on Chip, which helps in developing a complete real time system eradicating the need of any other external resources.

As a part of the system H.261 based video compression and G.711 based audio compression standards have been implemented on this SoC. The need of the device drivers for the serial mode of communication plays the key role for the HDLC protocol implementation.

CONTENTS

List of Figures

Abbreviations

RISC	Reduced Instruction Set Computer
DSE	Digital Signal Engine
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
PE	Processing Engine
MTE	Memory Transfer Engine
GBus	Global Bus
CT	Counter Timer
MVD	Motion Vector Data
ME	Motion Estimation
API	Application Program Interface
CBP	Coded Block Pattern
HDLC	High-Level Data Link Control

Chapter 1

Introduction

1.1 Overview:

The objective of surveillance is to monitor given environment and report information about relevant activity. Video surveillance systems have predominantly been employed to monitor activities within and around office buildings, airports and other facilities. A basic video surveillance system consists of a collection of video cameras mounted in fixed positions or on pan-tilt devices. The video streams are transmitted to a central location, displayed on monitors and recorded. Security personnel observe the video to determine if there is ongoing activity that warrants a response.

When in operation, the system should provide the user with timely assessments of relevant activity that allow the user to affect the outcome of an ongoing event. For military sectors, real time interpretation is required for the information produced by the system to be valuable. Therefore the challenge is to provide robust real-time video surveillance system that is easy to use.

Some of the issues that must be addressed by these new solutions are: time to market pressure, new standards, a widening range of requirements for quality of service, and multiple levels of communication bandwidth. All of this implies the need for implementations that can be scaled to a wide range of performance.

1.2 Problem Formulation:

The purpose of this project was to implement the **Remote Video Surveillance System** into the Cradle's **System on Chip**. In this the basic video and audio compression standards are used, and serial communication with the help of RF modems is done.

The Cradle 3SOC architecture is a new silicon architecture that provides significant benefits in applications such as communications and multi-media, which have high processing performance requirements. 3SOC is an array of RISC and DSP processors, and DMA

Engines, that give a seamless, scaleable system solution for such applications. 3SOC also has Programmable I/O, which allows the user to configure through software the I/O pins of the chip, thereby implementing any interface and protocol. This one single architecture can be used for routers, 2D and 3D graphics, video encoding and decoding and display, set-top boxes and Internet appliances. This system is basically a parallel type of architecture, with 256 of SDRAM other than instruction memory for each processor.

H.261 is an ITU-T H-series standard applicable to video surveillance or video conferencing. Video coding algorithm defined, in here, is supposed to be able to operate in real-time with limited delay. Video transmission is at p X 64 Kbps (where p is in the range of 1-30) over digital transmission lines.

In this project the team has worked on h.261 which provides variable bit rate of 1 Mbps using the CIF format with 4:2:0 encoding. An Audio/ Video card along with Cradle's RDS-3400 development board is used for providing the proper video format on which it needs to apply the compression algorithm. The technical details of the algorithm are covered in detail in the later chapters along with the implementation strategies.

1.3 Organization of Dissertation:

This dissertation is articulated in five major chapters including this chapter. The first chapter gives a brief overview of the problem formulation in terms of specifications and about the platform on which the specifications are to be met .The second chapter tells about the literature surveyed to give idea of the material resources. The third chapter gives the technical details of h.261 and G.711 and cradle as a platform. In fourth chapter design flow is discussed and the design strategies are discussed keeping the specifications in mind and trying to get the best use of the system architecture. Results are shown in chapter 5. In chapter 6 conclusions withdrawn from this work is given and the future scope is also discussed to provide more flexibility to the system but which may increase the system cost. The standard tables of the h.261 standards are listed in Appendix – A, and more elaborated results in terms of profiler results are added as Appendix – B.

Chapter-2

Literature Review

This project was the mapping of the h.261 and G.711 algorithm into the cradle platform and for this the main issue was to know the basics of the algorithms and basic architecture of the cradle chip along with the programmable IO s for interfacing the system with the RF modems. Also the need was to know the configuration tools and debugger and cradle specific languages to configure the chip according to the need for mapping the algorithms. So the literature used in this project were the reference manuals of the cradle technology downloaded from their site, which are discussed below here and the ITU-T standards. Other literature was taken from IEEE transactions on the basics of various blocks of the video compression given below.

2.1 Cradle (Platform) specific literature:

3400 Hardware Architecture reference: This reference manual gives the basic architecture of the platform used herein this project. It discusses about the various blocks of the SoC along with their specialized features. This manual was the complete reference guide to each unit that were worked on like PE, MTE, DSE

3400 DSE Programmers Reference: This describes the DSE instructions of the Cradle 3SOC Architecture.

3400 MTE Programmers Reference: This give details of the MTE instructions of the Cradle 3SOC Architecture.

3400 PE Programmers Reference: This describes the PE instructions of the Cradle 3SOC Architecture.

PIO TOOL GUIDE: This PIO manual for the Cradle Development Environment provides the following information:

- Descriptions of PIO hardware and firmware architecture.
- A guide for using PIOconfig, the tool used to configure the 3400 chips from Cradle Technologies. This guide helps the user configure the PIO hardware and firmware.

INSPECTOR'S USER GUIDE: This User Guide for the Cradle Development Environment describes for Cradle developers how to use Inspector to perform debugging tasks.

Bool manual for programming the CPLD, this is like any other hardware description language but syntax are like that of high-level language like C. This manual is complete guide to BOOL programming language.

CLASM Language Reference: provides details of the language used to program applications for the Cradle Technologies 3SOC Architecture.

CLASM - C Interface Reference: The CLASM Language Reference provides details of the language used to program applications for the Cradle Technologies 3SOC Architecture.

Cradle System Library Guide: The Cradle System Library Guide describes system specific functions.

Getting Started with the Cradle :Overview of the Cradle 3SOC architecture and the software tools included with the Cradle Development Environment

2.2 Algorithm specific Literature:

ITU-T h.261 standard [1] is an ITU-T H-series standard applicable to videophone or video conferencing. Video coding algorithm defined, in here, is supposed to be able to operate in real-time with limited delay. Video transmission is at p X 64 Kbps (where p is in the range of 1-30) over digital transmission lines.

Deepak Turaga & Mohamed Alkanhal [16] has discussed the various Block Matching Algorithms and also the matching criteria like cross- correlation, mean absolute difference, mean squared difference and other criteria. They also have given the comparison of these algorithm and thus to find the algorithm on the basis of computational complexity and computation time. Evaluation was based on two measures, CPU time for computational complexity and AMAD for quality.

Wen-Hsiung Chen, C. Harrison Smith, And S. C. Fralick [7] developed a fast Discrete Cosine Transform algorithm, which provides a factor of six improvement in computational

complexity when compared to conventional Discrete Cosine Transform algorithms using the Fast Fourier Transform.

Christoph Loeffler, Adriaan Liebenberg, and George S. Moschytz [4] introduced a new class of practical fast algorithms for the Discrete Cosine Transform (DCT. For an 8-point DCT only *11* multiplications and 29 additions were required according the butterfly structure they used. A systematic approach is presented to generate the different members in this class all having the same minimum arithmetic complexity.

Amit Gulati and George Campbell [3] various aspects of algorithm partitioning, reference frame coherency, and synchronization issues are discussed in this paper. This paper shows flexible methods for mapping the algorithm onto MDSPs, which allow scalability over coding tools, resolutions, and computation/bandwidth availability.

G.711 standard [27] Understanding PCM Coding-Application note of AN574.1, HARRIS Semiconductor.

Chapter- 3

System Theory

3.1 Cradle Architecture Overview

The CT-3400 chip has one compute quad whose DSEs have PIMAC and SIMD units, and one IO quad with scatter/gather DMA capabilities and 16 PIO pin groups. The number 3400 indicates the version of the architecture and the number of DSPs in the chip.

The Cradle 3SOC architecture is a new silicon architecture that provides significant benefits in applications such as communications and multi-media, which have high processing performance requirements. 3SOC is an array of RISC and DSP processors, and DMA Engines, that give a seamless, scaleable system solution for such applications. 3SOC also has programmable I/O, which allows the user to configure these I/O pins of the chip through software, thereby implementing any interface and protocol. This one single architecture can be used for routers, 2D and 3D graphics, MPEG decoding and display, set-top boxes and Internet appliances. The Cradle family has a common architecture that is scaled by varying the number of processors and the number of I/O pins. The 3SOC architecture provides

- Reduced time to market
- Predictable performance and cost
- Cost-effective solutions
- Adaptability to new system and memory architectures
- Increased designer productivity

3.1.1 Cradle Architecture

It consists of one cluster (called Quad) of RISC, DSP and DMA processors on a high speed Global Bus, an SDRAM memory interface, and a programmable interface for I/O devices such as a PCI bus, etc. The processors are called Processing Element (RISC), Digital Signal Engine (DSP) and Memory Transfer Engine (DMA). The SDRAM interface supports 8 banks of SDRAM, and is 64-bits wide. The programmable I/O interface has 16 pin groups with 8 pins in each group, for a total of 128 software configurable I/O pins. The I/O interfaces shown below are for typical Communications (Audio&Video) applications.

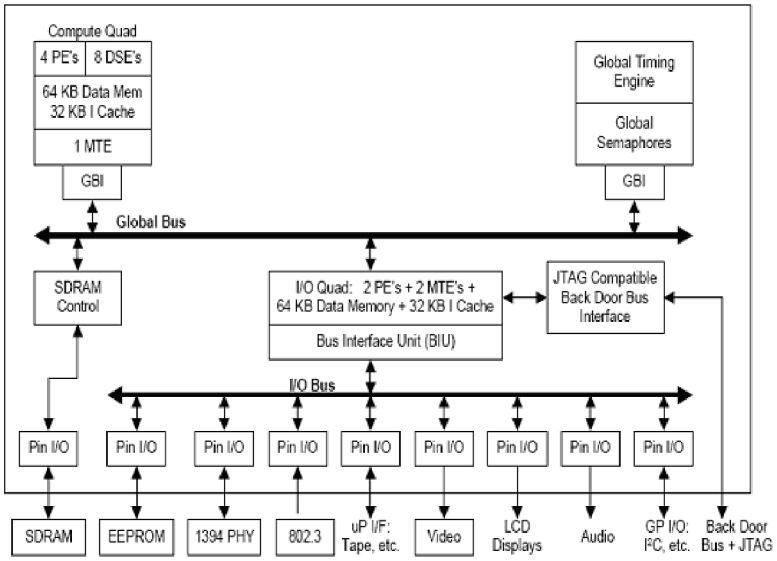


Figure 3.1 Cradle Architecture

3.1.2 Global Bus Architecture

The 3SOC bus structure consists of a single Global Bus (GBus) for the chip and local buses in each Quad. Gbus devices communicate with each other through the Global Bus. Examples of GBus devices include DRAM memories, global registers such as the global timing reference registers, and the Quads of PEs and DSEs. Each Quad has a local bus to allow local communication between Quad elements such as Processing Elements (PEs), Digital Signal Engines (DSEs) and Memory Transfer Engines (MTEs).

Buses allow bus elements to transfer information. Bus elements consist of two types: masters and targets. The processing elements (PE and DSE) and the memory transfer controllers (MTC) are examples of bus master devices. Memories (Quad data memories and DRAM memories) and registers (Quad hardware registers for PEs, DSEs, MTCs and CTs, and system registers) are examples of target devices. All information transfer is between masters and targets. Masters initiate transfers of data to and from targets.

3.1.3 Compute Quad Architecture

A Quad consists of Processing Elements (PEs), Digital Signal Engines (DSEs), data and instruction memories, Memory Transfer Engines (MTEs), a Counter-Timer interrupt register (CT), hardware semaphores and a Global Bus interface. The memories consist of an

instruction cache, a data cache and a local data memory. A Memory Transfer Engine (MTE) contains 4 Memory Transfer Controllers (MTCs). A Counter Timer Engine outside the Quad controls the CT.

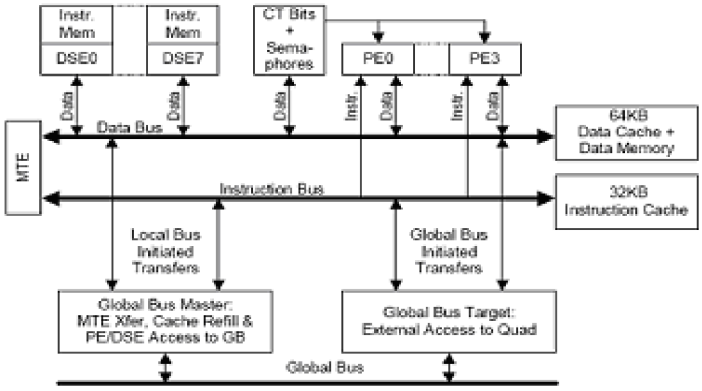


Figure 3.2 Global Bus Architecture

3.1.4 PE Architecture

Each Quad in the 3SOC architecture consists of processing elements and local memory. The processing elements are of three types: a general-purpose Processing Element (PE), a Digital Signal Engine (DSE) and a Memory Transfer Engine (MTE). Each Quad has four PEs, one MTE and eight DSEs. PEs provides set up and management of the operations of the DSEs and MTE. The eight DSEs are an independent Quad asset and any one or more of the PEs can control all of them. The MTE is associated with the Quad.

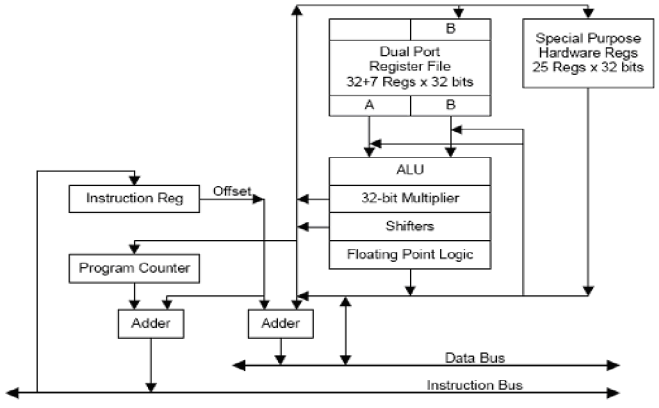


Figure 3.3 PE Architecture

Any PE or other Quad device can initiate a transfer by the MTE. The MTE can have up to four transfer streams going at once. The general-purpose Processing Element (PE) is a 32-bit general-purpose processor with interrupts. It is a 32-bit machine with a bank of 32 general purpose registers, a bank of 32 special-purpose registers, an ALU, a 32-bit multiplier and single precision (32-bit) floating point logic. It uses Harvard architecture, with separate data and instruction memories.

The PE instruction set processes both fixed point and 32 bit floating-point data, and it accommodates 8-bit, 16-bit and 32 bit data formats. It also has instructions for processing of bit string data, such as found in Huffman encoding. Most instructions have an op code and 2 operands specified by an A-field and a B-field.

3.1.5 DSE Architecture

The Digital Signal Engine (DSE) is a coprocessor that provides high performance processing of DSP functions such as multiply and accumulate. Figure 3.4 shows a block diagram of the DSE. The DSE consists of the following:

- A 128 word by 32-bit dual port data memory (DPDM)
- A 32-bit integer ALU with support for packed integer SIMD instructions
- A floating point unit with a 24-bit multiplier, 4 floating point accumulator registers, and a FMAC ALU
- A packed integer MAC unit (PIMAC) with 16 8-bit multipliers, 4 16-bit multipliers, 1 32-bit multiplier and 24 matching packed integer accumulator registers
- External memory read and write FIFOs
- A bit field access unit consisting of 3 registers: Field access control, Field Data and Field access as shown in figure 3.4
- A Program Control Unit (PCU) for controlling data Transfers in the data memory.

The PCU moves and combines data in the DSE dual port data memory (DPDM). It also controls transfers between the DPDM and special data processing units such as the floating-point accumulators, the field access register and the external memory read and write FIFOs

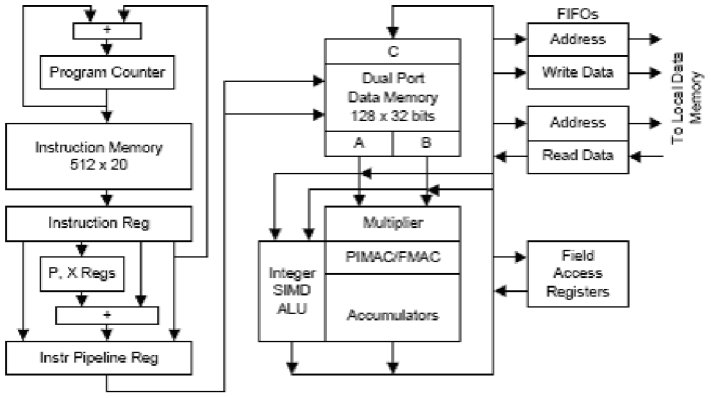


Figure 3.4 DSE Architecture

The PCU consists of the following.

- A program counter (PC) and a dedicated 512 word instruction memory.
- An instruction register, instruction decode unit and an instruction pipeline register.
- Four pointer registers (P registers) for addressing the data memory.
- An index (X) register for indexed addressing of the data memory.
- Three program control registers: PSW, loop counter and JSR save.

The PCU generates addresses for the DPDM and control signals for the integer ALU, floating point and PIMAC accumulators, field access registers and external memory FIFOs. The PCU reads and executes 20-bit instructions from a dedicated DSE instruction memory. Each DSE instruction defines a movement or combination of data in the DPDM or it defines a program control operation. The dedicated instruction memory allows the DSE to execute instructions at one clock tick per instruction.

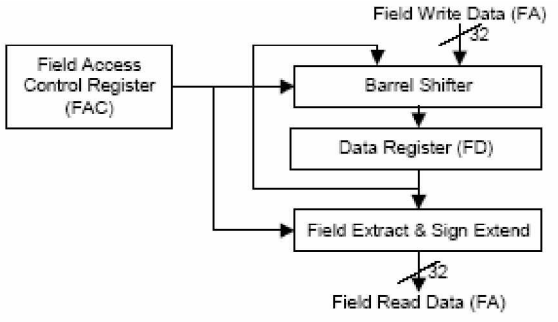


Figure 3.5 Field Access Unit

The PCU uses the immediate contents of the A and B fields or the contents of four pointer registers (P0-P3) to generate DPDM addresses. The pointer registers allow access to any part of memory and include address auto increment capability. An index register (X) allows indexed address to any bank of registers.

3.1.6 MTE Architecture

The Memory and I/O Transfer Engine (MTE) provides each Quad with four Memory Transfer Controllers (MTCs). Each MTC moves a block of byte data from a source address to a destination address. The MTE uses a single set of hardware to implement four MTC processors in a time sliced, round robin fashion. An arbiter in the MTE selects each processor in turn. Each MTC processor does one task, such as transferring one GBus atom of data (1, 2, or 4 Octets), then relinquishes control to the next processor. The MTE is also used in the I/O Quad for I/O data transfers. The MTE provides DMA data transfer in standard Quads and provides I/O data transfer and control in the I/O Quad.

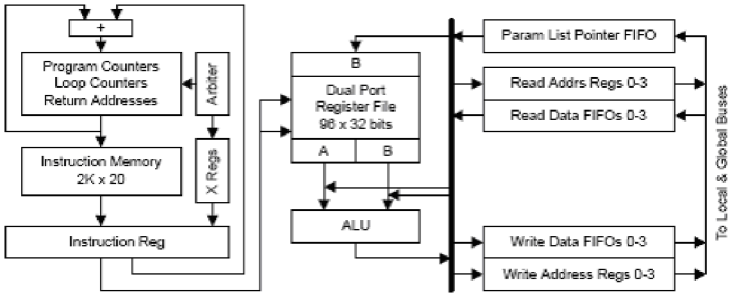


Figure 3.6 MTE Architecture

The MTE is a multithreaded processor with four program counters and a 96-word dual-port register file used as six banks of 16 registers. Also, each MTC has its own memory read and write FIFOs. The MTE arbiter selects the active MTC processor by selecting its program counter and register bank.

3.1.7 Global Functions

Global functions concern the status and control of the CRA as a whole. Global functions include:

- Global semaphores

- Global Registers: Global Wake-up Interrupt (GWI) + Global Stop/Resume
- Clock System: Clock Phase Locked Loop + Clock Generators
- Global Timing Engine (GTE)
- Global Performance Monitors
- Power Management

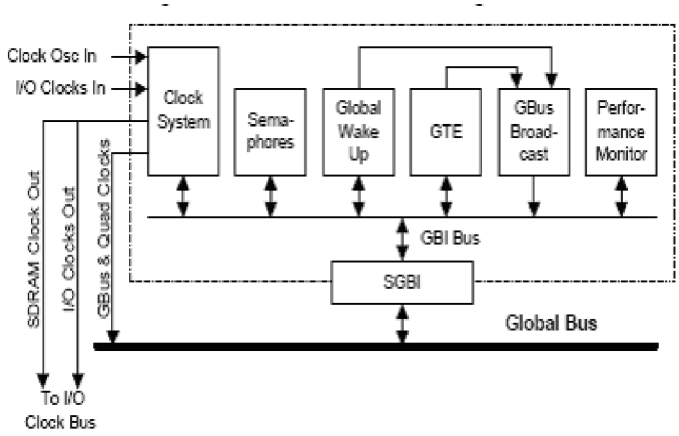


Figure 3.7 Global Bus

3.1.8 PIO Architecture

The 3400 chip's I/O Quad has 2 PEs, 2 MTEs, local instruction memory and data memory, 1 Bus Interface Unit (BIU) and 16 pin groups. It can support multiple I/O devices, and up to 4 DMA devices. Many devices, such as RS-232, SVGA, SCSI, PCI, 1394, etc. can be implemented in firmware, using unmodified chip hardware.

PIO devices use a PE and pins, and possibly a serial unit (SIO) configured to operate in PIO mode. The PE program directly reads the pins of the pin groups, and directly writes to them, changing their levels (bit-banging). It directly reads and writes SIO control and data registers.

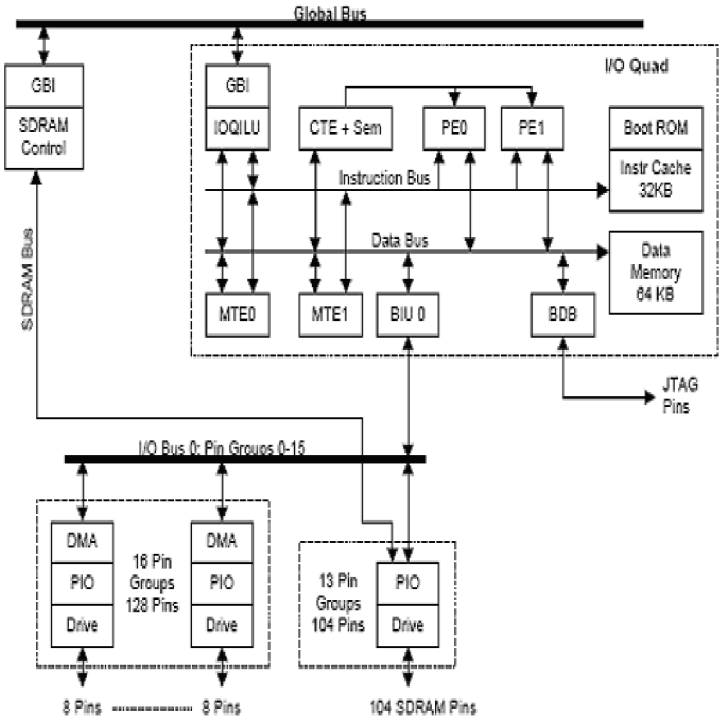


Figure 3.8 PIO Architecture

DMA devices use the PSM, which is the heart of a pin group, and which controls all the other hardware in the pin group. In addition to the PSM the pin group contains timers/counters, a serial unit configured to operate in DMA mode, FIFOs, an address generator, a parity generator, external comparators and pins (8 pins per group). Each DMA device may use three programmable hardware layers. The three hardware layers of a DMA device are:

- Processing Element (PE).
- Optional MTE, and a channel controlling 4 FIFOs (Command, status, data input, data output) in a BIU.
- One or more pin groups, each containing a Pin State Machine and 4 FIFOs.

The firmware configures and controls the hardware in each layer in accordance with the device's operational requirements and dynamically processes pin I/O data to implement device functionality.

3.1.9 Design Process of DMA / PIO device

First a DMA device designer specifies the number and function of the pin groups, and then he uses PIOconfig to set up the components of each of the device's pin groups. When the configuration is complete, PIOconfig automatically generates a C file. The C file is compiled into the PE program controlling the device. This C code calls functions in the Cradle-supplied PIOconfig library, which configure device hardware. PIOconfig is also used when designing PIO devices.

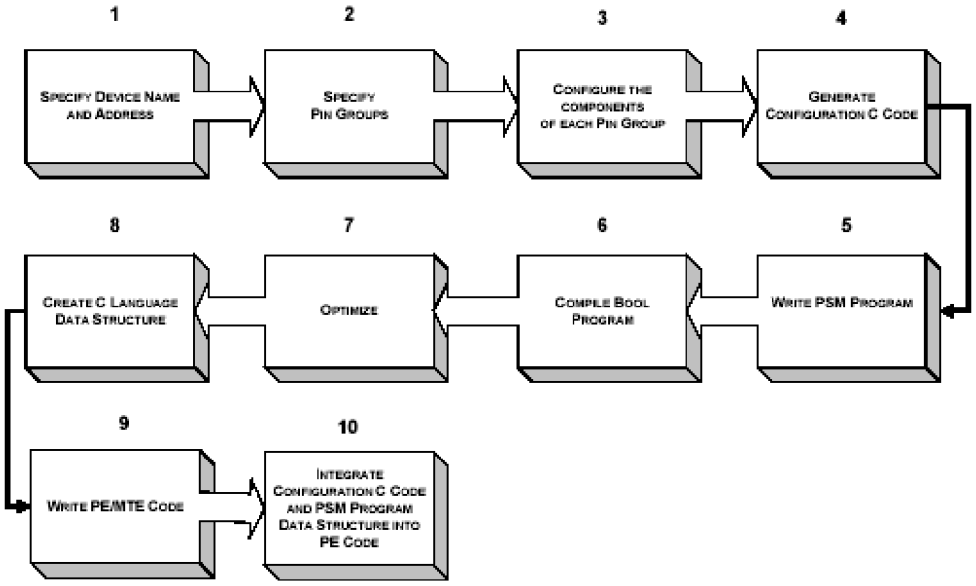


Figure 3.9 DMA/PIO device design process

In the next stage, the DMA device designer writes the PSM program(s) for those pin groups of the device that will have an active PSM. This is done using any language and tool set capable of generating standard .pla files. Cradle recommends that designers use BOOL, ESPRESSO (both public domain development tools), and PLA2UMS, a Cradle-developed utility. The designer starts with a skeleton BOOL source file named pins.b in the ..\src\pio directory. This file contains:

- The names of all input, output, and Flip-Flop-output lines of the PLA and aliases associated with each line. Because each input line is fed by the input-side multiplexers from one of several possible inputs, each can have more than one name.
- Names for encoding output-side commands from the PLA to the timer/counters, data FIFOs, and serial mode control lines.
- A writestd(...) line specifying all the PLA outputs that will be generated by the compiler.

After compiling the BOOL program, the user runs it through the ESPRESSO program. This program takes advantage of the fact that outputs can be inverted, and that both a signal and its inverse are available at the input to the PLA AND gates. It minimizes the number of product terms needed in the compiled program.

The output of the ESPRESSO tool is fed to the PLA2UMS utility, which creates a C language data structure representing the PSM program. The data structure should be compiled into the PE program controlling the device. The data in the structure is processed at run-time by a Cradle-supplied library function that loads the program into PSM.

3.1.10 Clock Configuration at Pin Group

A pin group has three (3) clock signals:

• I/O Clock

• Input Sample Clock

• Output Sample Clock

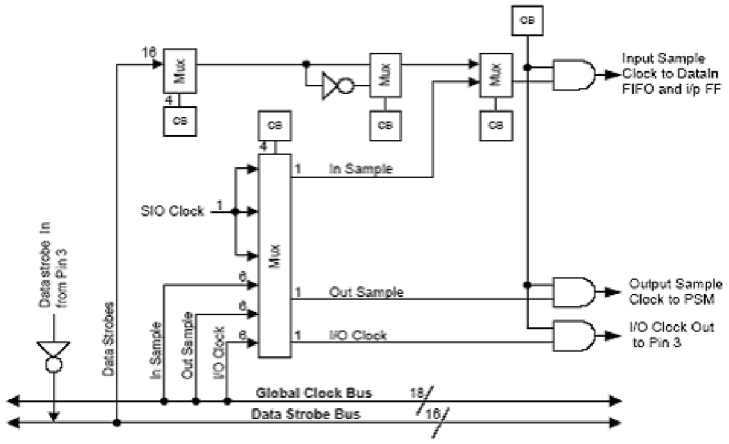


Figure 3.10 Clock Configuration

These clocks can be derived from one of the following:

- A stand-alone internal clock generator circuit, fed by a high-frequency PLL clock. There are 6 such generators in the chip.
- The SIO clock when it is subdivided from the quad clock
- External clocks (0-5). Each of these clocks can either feed one of the Delay Locked Loops or one of the internal clock generators. The batch simulator can feed in a VCD-file originated or C++ program-originated clock signal into these pins.

Each internal clock generator can be used either stand-alone, or can be fed with a clock from an external clock pin. As mentioned, there is also a Delay Locked Loop associated with each external clock pin and internal clock generator circuit. This DLL is not simulated, since the simulator does not simulate clock skew, setup times and hold times. It is good practice to use the DLL when running on hardware whenever the clock rate is >25 MHz.

When the clock source is configured to be Clock Generator, a dedicated hardware circuit generates the 3 clocks internally. The user specifies the period, which specifies the frequency of all 3 clocks. For each clock, the user then specifies a mark length (in units of PLL clock), thus choosing a duty cycle for the waveform. For the Input Sample and Output Sample clocks, the user can specify a stagger, that is, a phase lag behind the I/O Clock (in units of ½ a PLL clock).

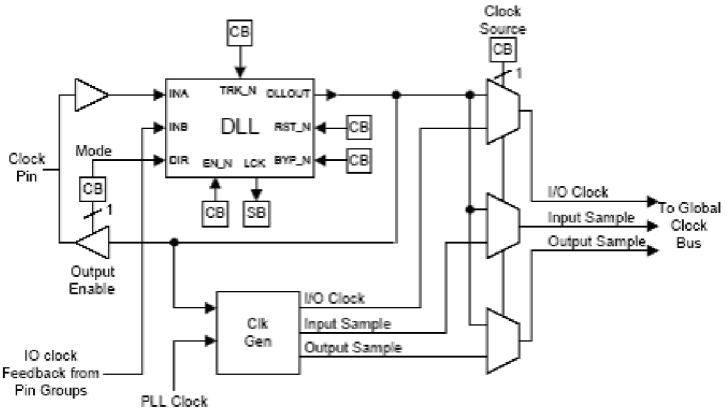


Figure 3.11 External Clock Configuration

As well as going to the pin groups, the IO clock generated by the Internal Clock Generator can also be fed out through an external clock pin. This output can be raw or, if the DLL (Delay Lock Loop) is enabled, will be de-skewed so that the clock edges at the external clock pin are aligned with the IO clock edges at the pin groups.

When the Clock source is configured to be external, the IO clock is fed from outside into an external clock pin, and the user can optionally either feed it through a Delay Lock Loop or feed it through the internal clock generator.

When the external clock is fed through the DLL, all 3 clocks fed to the pin groups are the same, and are fed from the output of the DLL. The DLL de-skews the clocks so that the clock

edges at the registers in the pin circuits are aligned with the clock edges at the external clock pin.

When the external clock is fed through the Internal Clock Generator, the hardware circuit generates the 3 clocks internally. The IO Clock is synchronized to the external clock to within 1 PLL clock pulse, and the Output Sample and Input Sample clocks are offset from the IO clock as determined by the programmed values of mark and stagger. The programming resolution of the offsets is ½ a PLL clock pulse. The clocks of a pin group are inactive until the pin group is completely configured and its PSM program completely loaded. The PE activates these by setting a bit in a configuration register.

3.1.11 Timer Counter Configuration Block

Each timer can be configured to operate in either timer or counter modes. In **timer mode**, the down-counter decrements every Output Sample Clock edge. In **counter mode**, the down counter decrements every Output Sample Clock edge in which the PSM is outputting a decrement commands to the counter. The PSM can send the following commands to a timer:

- NOP
- Decrement Counter
- Load from RAM0
- Load from RAM1
- Load from RAM2
- Load from RAM3
- Reload

A timer operates in either one-shot or continuous mode. When the timer reaches 0, the timer=0 input to the PSM goes to 1. In **one-shot** mode, the PSM must issue a load or reload command to the timer in order to reload the down counter with a time-constant. In **continuous** mode, the circuit automatically reloads the down counter with a time-constant from the latch. The **load** command transfers the time-constant from RAM to the latch as well as to the down counter. The **reload** command transfers the latch contents to the down counter.

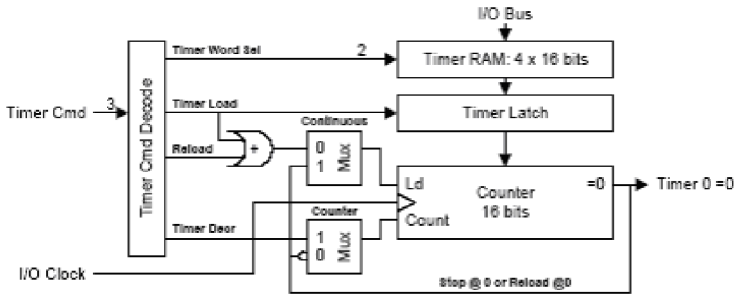


Figure 3.12 Timer Counter Configuration

3.1.12 SIO Configuration Architecture

A pin group has one (1) serial unit. The Serial I/O circuit provides synchronized shifting of input and output data and can be controlled by a PE, an MTE, or the pin group's PSM. The controlling unit can:

- Determine the busy/not-busy status of the unit by reading the SIO bit counter (PE/MTE) or by sampling the RTS line (PSM).
- Initiate a shift operation by writing to the SIO control register (PE/MTE) or by setting to 1 the OKtoSend line (PSM).
- Reset the unit by setting the ResetCounter bit in the SIO control register (PE/MTE) or by setting to 1 the SIOflush line (PSM).

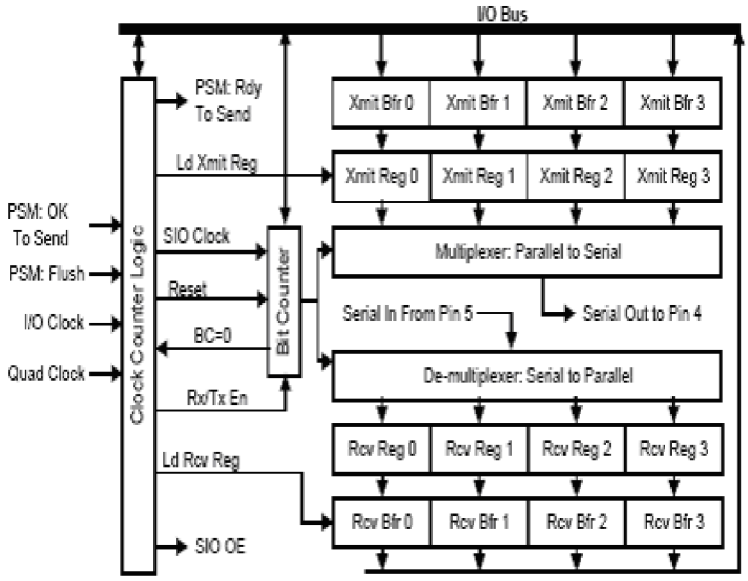


Figure 3.13 SIO Configuration Architecture

The unit can operate in either single-buffered or double-buffered mode. The data buffers are all 32 bits wide and can be programmed to handle 1 to 32 bits per word. Data is output on Pin 4 and is input on Pin 5. The shift clock can be the same as the pin group's Output Sample clock, or can be divided down from the quad clock. The shift clock can be output on Pin 6.

The SIO either runs continuously (in which case double-buffering is used) or stops after each word. The data can be inverted, as can the shift clock, so that shifting occurs on the trailing edge. A user can shift either the MSBit or LSBit first.

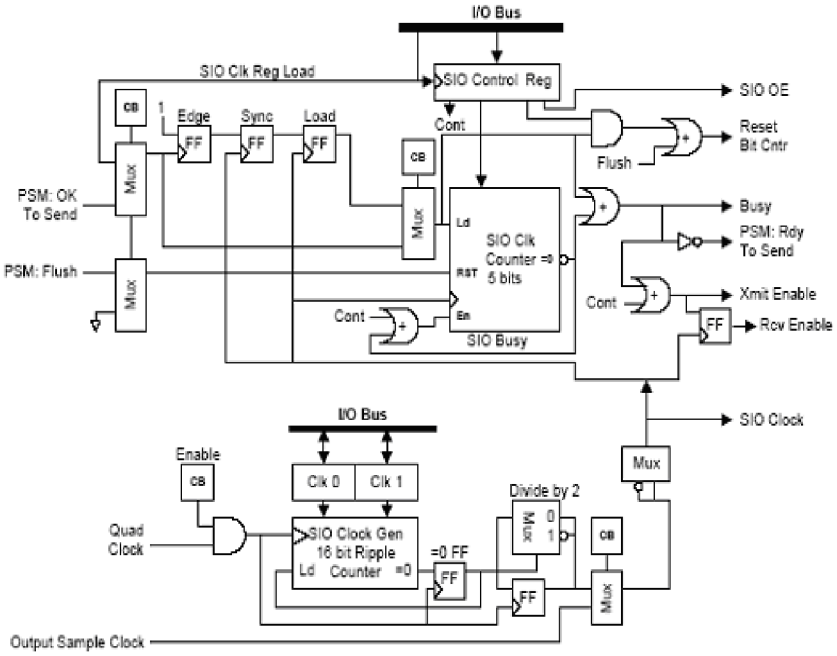


Figure 3.14 SIO Clock Counter

3.1.13 PIO Interrupt Block Diagram

A pin group has one interrupt-generating module. The interrupt-generating module can interrupt a PE by writing to a local semaphore that has been primed previously by the PE reading it. The module can also wake up an MTC by writing to its PLP FIFO. The programmer writes the address of the semaphore/PLP-FIFO into the pin group's Interrupt Address register at run time.

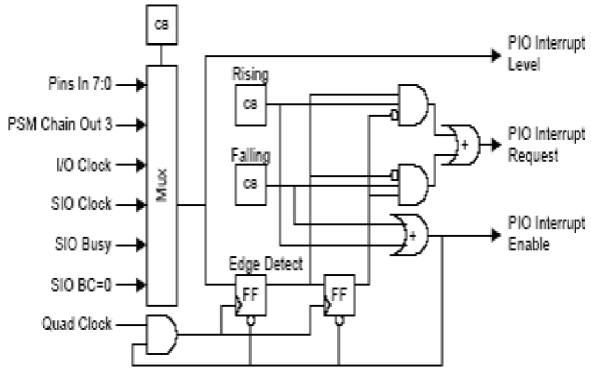


Figure 3.15 PIO Interrupt Block

3.1.14 PSM Architecture

Each pin group has **one** Pin State machine. The PSM outputs control all other components of the pin group. In particular, the PSM has outputs for:

- **State:** If all zero, state register does not change (that is, all-zero is equivalent to "stay in current state").
- **Status:** Fed to the status FIFO; if not all-zero, clocks data into the FIFO.
- **Output pins and OE of pin group:** Eight OE patterns, held in RAM.
- **Control/Data mux control:** Eight (8) controls for output pin source, held in RAM.
- **Data FIFO control**: Includes three (3) commands–Data Out FIFO, Address Counter, Serial mode)
- **Chain**: This is useful for pattern detection; it's often used for user-defined purposes.
- **Get next command from Command FIFO**
- **SIO control**
- **Timers control**

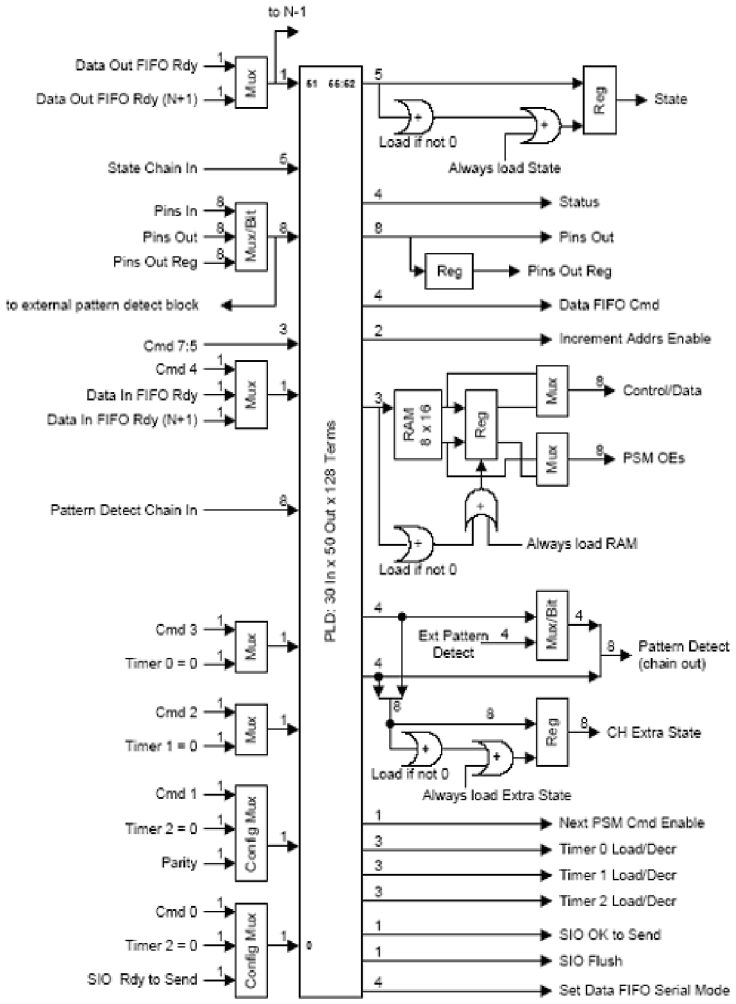


Figure 3.16 PSM Architecture

PSM configuration consists of setting the input MUXs to the desired inputs. Because a PSM has only 30 inputs, only a subset of the allowable inputs is actually connected at any one time. Typically, this configuration is fixed and depends on the device design.

Some registers on the output of the PSM will be loaded only if the output value from the PSM is non-zero. The value will be maintained unchanged if the PSM outputs zero. This saves product terms in the PSM program, allowing the user to program "leave unchanged" with zero cost. However the designer can override the feature, and cause the registers to be loaded even when the PSM outputs zero. PIOconfig provides overrides for the 3 circuits in question – State, RAM and Extra State.

3.15 BIU FIFO Configuration Architecture

Each device uses **four** BIU FIFOs, one each for:

• Command

• Status

• Data In

• Data Out

These FIFOs are 64-bits wide at the data bus side, where access is by PE or MTE. Through the I/O Bus, the FIFOs connect to corresponding FIFOs of the pin groups that comprise the device. A valid device address is 0-3 for BIU 0. Pin groups 0-15 are connected to BIU 0. Data can be transferred automatically to and/or from SDRAM by the BIU−without processor intervention by configuring the FIFO to operate in DMA mode. In this case, the source/destination address must be set up by the device initialization code. The auto-increment option must also be configured for the FIFO.

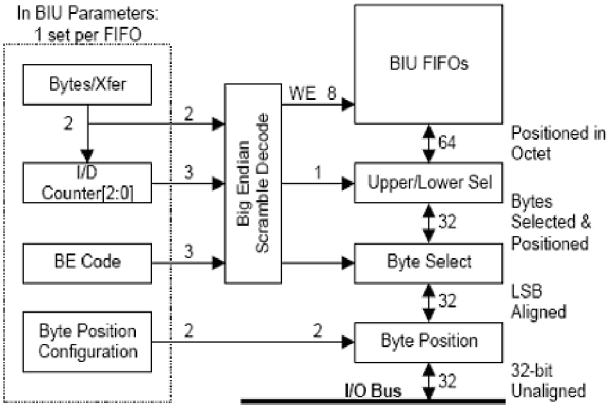


Figure 3.17 BIU FIFO Architecture

3.2 H.261

3.2.1 Introduction

H.261 is an ITU-T H-series standard applicable to videophone or video conferencing. Video coding algorithm defined, in here, is supposed to be able to operate in real-time with limited delay. Video transmission is at p X 64 Kbps (where p is in the range of 1-30) over digital transmission lines.

3.2.1 The Technology

H.261 uses DCT as a main data compression tool in compressing moving images in 'Spatial' and 'Temporal' domains. The transform coefficients are coded by a variable length coders (VLC) e.g. Huffman coders. The motion compensated temporal prediction errors are mapped in DCT domain (to remove spatial co-relation) followed by quantization and VLC as shown below:

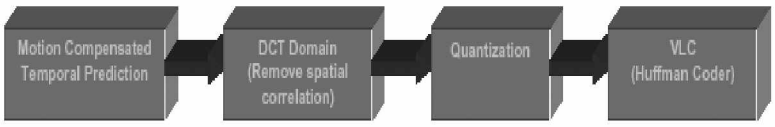


Figure 3.18 Basic Flow of H.261

The over all structure of the H.261 is a hybrid structure in 'Spatial' and 'Temporal' domain. H.261 is similar to MPEG-1 from a compression point of view. Typical frame specifications for the H.261 are as:

- Image Format: CIF (NTSC & PAL)
- Resolution: 352 X 288 (for QCIF resolution is 176 X 144)
- Y: CB: CR: 4:2:0
- Scan Type: Progressive
- MB/sec: 4.6 (for QCIF rate is 1.1 MB/sec)
- Pictures/Sec: 29.97 Hz

H.261 support following video formats:

Video Format	Luminance Image Resolution	Chrominance Image Resolution	Bit-Rate(Mbps) (if 30 fps and uncompressed)	H.261 Support
QCIF	176 x144	88 x 72	9.1	Required
CIF	352 x 288	176 x 144	36.5	Optional

3.2.2 Techniques used for H.261 codec

1. Two dimensional (2-D) 8 X 8 DCT to remove intra-frame correlation
2. Zig-zag order to scan the transform coefficients
3. Run Length coding for zero-valued coefficients after Quantization.

4. Motion estimation is applied to video sequence to improve the prediction between successive frames (more involved in MPEG).
5. Transmission rates control in the range of p X 64 Kbps
6. Error resilience including synchronization and concealment technique required in transmission code, to cover up channel errors.
7. Common Intermediate format (CIF) for a single solution to different video formats (NTSC / PAL or SECAM)

3.2.3 H.261 data structure

H.261 data structure comprises of four layers as defined below:

1. Picture Layer
2. Group of Blocks (GOB) Layer
3. Macro block Layer (MB)
4. Fundamental (8 X 8) block layer

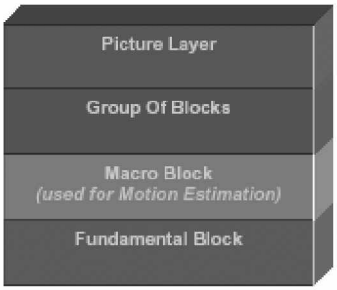


Figure 3.19 H.261 Data structure

All the layers are multiplexed for transmission in series, with each layer having its corresponding header information bits. Hierarchical structure is used such that the number of a hierarchical synchronization can avoid the loss of a whole frame. Certain regions that have uncorrectable errors can be duplicated or interpolated.

H.261 Picture syntax:

Each layer for H.261 is composed of header information and data of the following layer as shown below:

PSC	TR	PTYPE	PEI	PSPARE	PEI	GOB Data

Figure 3.20 Picture syntax

Group of Blocks:

Each picture is divided into groups of blocks (GOBs). A group of blocks (GOB) comprises one twelfth of the CIF or one third of the QCIF picture areas (see Figure 6). A GOB relates to 176 pels by 48 lines of Y and the spatially corresponding 88 pels by 24 lines of each of CB and CR. Data for each group of blocks consists of a GOB header followed by data for macroblocks. The structure is shown in Figure3.21. Each GOB header is transmitted once between picture start codes in the CIF or QCIF sequence numbered in Figure6, even if no macroblock data is present in that GOB.

1	**2**
3	**4**
5	**6**
7	**8**
9	**10**
11	**12**

Figure 3.21 Group of Block

Macroblock layer

Each GOB is divided into 33 macroblocks as shown in Figure 8. A macroblock relates to 16 pels by 16 lines of Y and the spatially corresponding 8 pels by 8 lines of each of CB and CR. Data for a macroblock consists of an MB header followed by data for blocks. MQUANT, MVD and CBP are present when indicated by MTYPE.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
12	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
23	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**

Figure 3.22 Macro Block Layer

MBA	MTYPE	MQUANT	MVD	CBP	BLOCK DATA

Figure 3.23 Macro Block Structure

Block Diagram:

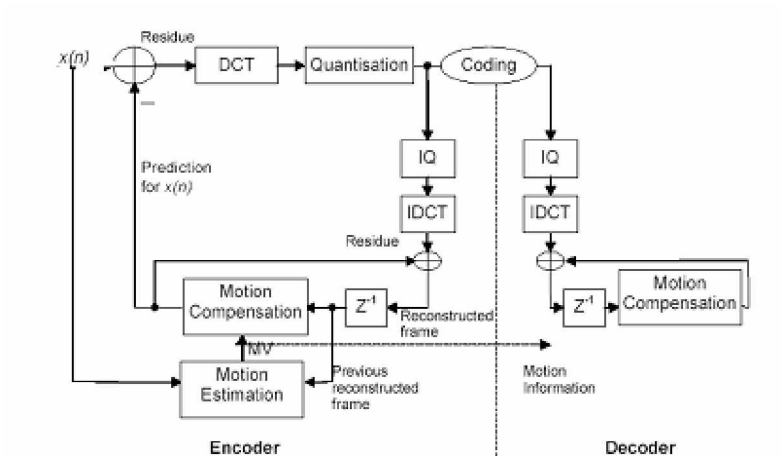


Figure 3.24 H.261 Block Diagram

A very brief description of the necessary parts of H.261are given in order to explain the design. Figure 1 & 2 explains basic blocks of H.261 Codec. H.261supports I & P frames with sequence of IPPPIPPP. Quantization uses a constant step size defined by either GQUANT in GOB header or MQUANT in Macroblock header.

3.2.4 Various Parts of Algorithm

1. Motion Estimation and compensation
2. Discrete Cosine Transform
3. Quantization
4. Variable Length Coding

Interframe predictive coding is used to eliminate the large amount of temporal and spatial redundancy that exists in video sequences and helps in compressing them. In conventional predictive coding the difference between the current frame and the predicted frame (based on the previous frame) is coded and transmitted.

Motion Estimation:

Motion Estimation is the process of matching the current MBs to be coded with a similar MB from the previous frame. Displacement is represented by a vector which is called motion vector. The main goal of motion estimation is the calculation of this difference which is

quantized and sent as a motion vector. MPEG uses both spatial and temporal correlation to find the motion vectors. Spatial correlation is assumed that a particular pixel is highly correlated with its neighboring pixels and can be predicted using them. Temporal correlation is video frames that show similarities between consecutive frames with very slight differences due to motion, sufficient to send enough info to extrapolate from the reference frame. Successive pictures in a motion video sequence tend to be highly correlated. Motion Estimation is used to identify and compress temporal redundancies between these correlated consecutive picture frames. The amount of motion is encapsulated in the motion vector. Forward motion vectors refer to correlation with previous pictures. Backward motion vectors refer to correlation with future frames. In h.261 only forward motion estimation is done. This is done prior to DCT to achieve the temporal correlation between frames.

Motion Estimation is in most cases based on a search scheme, which tries to find the best matching position of a 16x16 MB of the current frame with a 16x16 block within a predetermined, or adaptive search range in the previous frame. The Figure 3.25 describes the basic principles of BMA, which is most popular in motion estimation in standardized video compression schemes. $I_K(x,y)$ is defined as the pixel intensity (luminance or Y component) at location (x,y) in the k-th frame, and $I_{K-1}(x,y)$ is the pixel intensity at location (x,y) at the k-1th frame. For block-matching motion estimation in the search area of the size $R^2 = R_x \times R_y$ pixel of the reference frame and $I_K(x,y)$ belongs to the current frame.

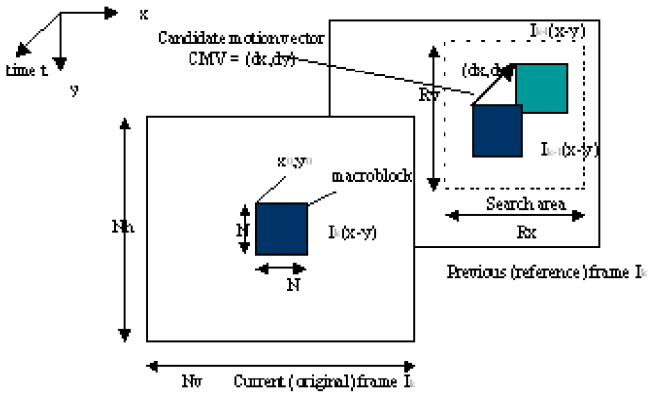


Figure 3.25 Block matching between current and previous frames

There are a number of criteria to evaluate the "goodness" of a match and some of them are:

1. Cross Correlation Function
2. Pel Difference Classification (PDC)

3. Mean Absolute Difference
4. Sum of Absolute Difference.

As the architecture have in-built SAD, which gives the best match we in this project has used the SAD. There are various algorithms using different search criteria for getting the best match but as the h.261 supports the 2-dimensional logarithmic search algorithm we are using the same and discussed it over here.

Two Dimensional Logarithmic Search (TDL)

This algorithm is more accurate, especially when the search window is large. The algorithm may be described as:

Step 1: Pick an initial step size. Look at the block at the Centro the search are and the four blocks at a distance of s from this on the X and Y axes. (the five positions form a + sign)

Step 2: If the position of best match is at the centre, halve the step size. If however, one of the other four points is the best match, then it becomes the centre and step 1 is repeated.

Step 3: When the step size becomes 1, all the nine blocks around the centre are chosen for the search and the best among them is picked as the required block.

A particular path for the convergence of the algorithm is shown in the following figure 3.26.

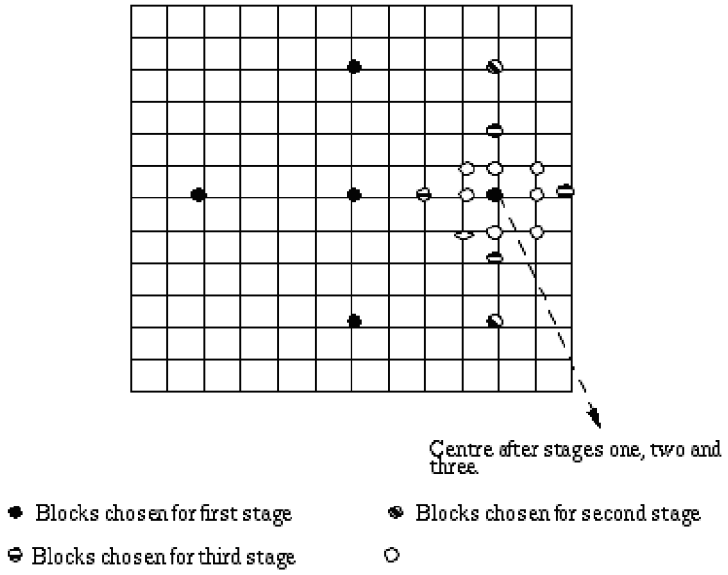


Figure 3.26 Example path for convergence of 2-D Logarithmic Search

A lot of variations of this algorithm exist and they differ mainly in the way in which the step size is changed. Some people argue that the step size should be halved at every stage. Some people believe that the step size should also be halved if an edge of the search space is reached. However, this last idea has been found to fail sometimes.

3.2.5 DCT

This block extract the spatial redundancy , it first calculate 1 dimensional DCT along the ROW and then after taking transpose of the result , performs the row DCT again which now is the COLUMN DCT. The transfer function of the inverse transform is given by:

The *N*x*N* two-dimensional DCT is defined as:

$$F(u,v) = \frac{2}{N} C(u) C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

$$C(u), \quad C(v) = \begin{cases} \frac{1}{\sqrt{2}} \text{for } u,v = 0 \\ 1 \text{ otherwise} \end{cases}$$

The inverse DCT (IDCT) is defined as:

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u) C(v) F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

where *x,y* are spatial co-ordinates in the image block
u,v are co-ordinates in the DCT coefficient block

Quantization

Quantization is used to reduce the number of possible values to be transmitted, reducing the required number of bits. The DCT coefficients are quantized as a lossy compression step. There are two types of quantization available. Both are essentially a division of the coefficient by a quantization step size. The first method uses one of two available quantization matrices to modify the quantization step size depending on the spatial frequency of the coefficient. The second method uses the same quantization step size for all coefficients.

The number of quantizers is 1 for the INTRA dc coefficient and 31 for all other coefficients. Within a macro block the same quantizer is used for all coefficients except the INTRA dc one. The decision levels are not defined. The INTRA dc coefficient is nominally the transform value linearly quantized with a step size of 8 and no dead-zone. Each of the other 31 quantizers is also nominally linear but with a central dead-zone around zero and with a step size of an even value in the range 2 to 62.

Variable Length Coding:

The serialization and coding of the quantized DCT coefficients exploits the likely clustering of energy into the low-frequency coefficients and the frequent occurrence of zero-value coefficients. The block is scanned in a diagonal zigzag pattern starting at the DC coefficient to produce a list of quantized coefficient values, ordered according to the scan pattern.

The list of values produced by scanning is entropy coded using a variable-length code (VLC). Each VLC code word denotes a run of zeros followed by a non-zero coefficient of a particular level. VLC coding recognizes that short runs of zeros are more likely than long ones and small coefficients are more likely than large ones. The VLC allocates code words which have different lengths depending upon the probability with which they are expected to occur. To enable the decoder to distinguish where one code ends and the next begins, the VLC has the property that no complete code is a prefix of any other. It uses the standard table to do the Huffman coding to the results of the DCT and for this:

- The first step is to group the values into runs of (zero or more) zeros followed by a non-zero value.
- The second step is to generate the variable length code words corresponding to each group (a run of zeros followed by a non-zero value) and the EOB marker

3.3 Audio Compression

With the necessity of audio on a wireless link of 76kbps,the audio pcm data has been compressed using the basic principles involved in the G711 standard.

The G-711 standard is the most common compression algorithm for worldwide telecommunication networks, and is found on T1, E1 and ISDN lines. Basically, the G-711

standard compresses a 13-bit A-law or 14-bit μ-law linear PCM sample to an 8 bit logarithmic representation. The logarithmic partitioning of the input values for encoding using A-law PCM. The diagram depicts the basic idea behind the technique. The encoding of negative values is performed using a similar structure whereby the smaller the magnitude of the negative value, the smaller the segment number. The μ-law is similar and the conversion to and from it is incorporated within the standard. Both techniques allow only 8-bits to be used per a sample with a resulting speech quality almost indistinguishable from the original signal at a significantly reduced data rate of 64 kbits/s. This resulting PCM signal provides a quality benchmark for all other telephone compression algorithms. At the receiving end the data is then converted back to the linear scale (13-bits for A-law and 14-bits for μ-law).

In this project basic A-law audio compression technique is used where input Bit Values for A-law Compression appear as:

Bit 15 14 13 12 … 3 2 1 0

Value S X11 X10 X9 … X0 -- -- --

Which consists of:

S - Sign bit

X - 13-bit magnitude

"--" represents don't care

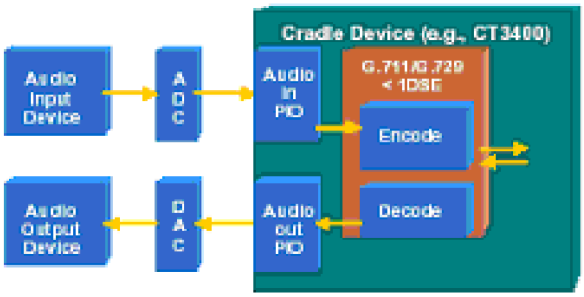


Figure 3.27 Audio codec on Cradle

Output (written over input array) consists of integers where each integer contains the encoded value (in bits 0-7). The even numbered bits are inverted during a-law encoding:

Output Bit Values for A-law Compression

Bit 15 … 7 6 5 4 3 2 1 0

Value -- … P S2 S1 S0 Q3 Q2 Q1 Q0

Which consists of:

P - Polarity bit (sign)

S - 3-bit segment number

Q - 4-bit quantization number

"--" Represents don't care

Chapter-4

System Design

4.1 Video Surveillance System:

This is a complete Remote Video surveillance system, which is designed to monitor the activity at a remote place by sitting at a base station. This comprises of one-way video at 1 Mbps and Audio and data together at 76 Kbps, which is 2 ways. In this project the video, audio and RS232 drivers used were provided by cradle and were modified according to system requirement. This chapter deals with the design specifications and the memory and resources allocation in the chip. It further explains the encoder and decoder mapping in the chip.

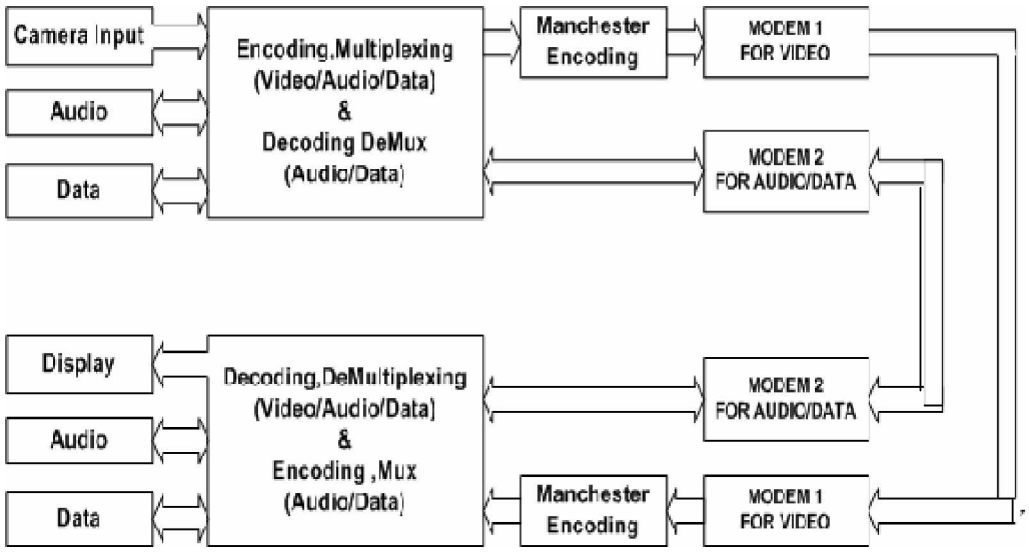


Figure 4.1 Video Surveillance System

Channel 1: One way Video at 1 Mbps (Asynchronous link)

Channel 2: Two way Audio and data multiplexed at 76 kbps.

Modem 1: For video

Modem 2: Synchronized Modem for multiplexed audio and data both ways

Design Description:

This describes the implementation of Video surveillance system on CRADLE. Sections following this include specifications, resources used and current limitations of the encoder and the decoder.

First step in system design is to capture the data from the camera and give it to the cradle chip where we will be performing the compression/decompression of the image and then sending it to the transmission media. The basic flow is given above in the block diagram.

In this project an analog camera is used to capture the video and then this is applied to the video board of the cradle which consists of **TVP5145 decoder** chip, that convert RGB or composite video to the YUV format, so this chip act as the Interface no. 1. After getting the YUV format from the decoder chip we store the data in the SDRAM provided with the cradle and do the necessary processing i.e. compression/decompression (discussed later). Once the processing is over send the data using the appropriate protocol (HDLC over here) to the transmission media.

At the receiving end receive the data and give it to the cradle and decompress it over here and then again send it to the interface which at this time is **SAA7148 encoder** which collects the image in the YUV format and then convert it to the composite video or s-video. The video-in and the video-out device drivers have been implemented using the IO quad block of the chip and using the necessary PE and MTE.

4.2 COMPRESSION ALGORITHM

4.2.1 Encoder

Encoder Specifications

Maximum frame size: 352x288 (CIF)
Frame format: 4:2:0 (YUV)
Frame rate: 5-25
Bit rate: 1 Mbps
Search range: +/- 7
MB Types supported: 16x16

Entropy coding: Huffman

Reference frame: I

Encoder resource requirement

On chip resources:

1 PEs

3 DSEs

10176 Bytes of PL

Off chip resources:

DRAM => 4 * FRAME SIZE

Encoder limitations:

Only CIF 420 Support

Block Diagram:

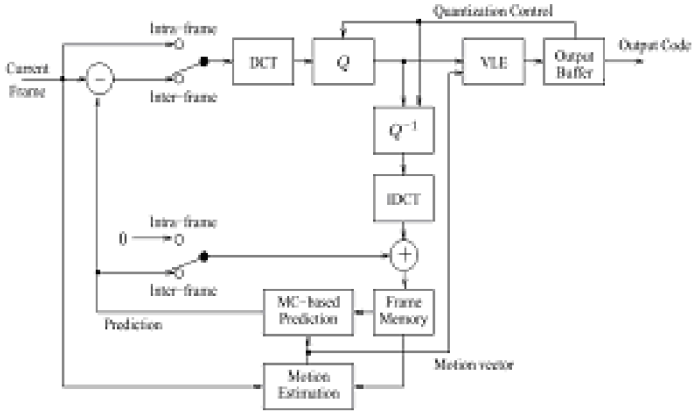


Figure 4.2 Encoder Block Diagram

Encoder design

The encoder is implemented using 3 DSEs and 1 PE, with I and P frame support. DCT DSE requires 252 instructions and VLC DSE requires 465 instructions. The computations are done using DSEs and program control and data transfer functions are performed by PEs with the help of MTCs. One PE controls the function of 3 DSEs like loading DPDMs, writing Picture and GOB header and starting of DSEs. Loading of DSE programs is carried by the PE with the help of MTCs. Separate programs are loaded on to DSEs for DCT and VLC.

Pseudo Code

The pseudo code for PE which controls the DCT and VLC DSEs is as follows:

```
INITIALIZE THE MBPOOL TO ZEROS USING CONSTANT FILL
INITIALIZE THE MBPOOL STRUCTURES AND INITIALIZE THEM AS FREE
LOAD DSE DPDMs AND DSE CODE FOR DCT
LOAD DSE DPDMs AND DSE CODE FOR VLC
LOAD DSE DPDMs AND DSE CODE FOR FEEDER
INITIALIZE THE VBV STURCTURE
DUMMY DONE FOR FIRST WAIT ON VLD DSE
READ ENCPARAM STRUCT AND FILL PICTURELAYER STRUCT ACCORDINGLY
INITIALISE VLC DSE COMMAND STRUCTURE
INITIALISE DCT OUTPUT BUFFERS
INITIALISE RECEIVE BUFFERS STATUS FREE
INITIALISE FEEDER OUTPUT BUFFERS FREE
START DSEs FOR DCT, VLC AND FEEDER

ENCODE_I_FRAME()
{
WRITE PICTURE HEADER
LOAD FRAME LEVEL DPDMS ON VLC DSE
--FOR ALL GOBs IN A FRAME
    {
     DOUBLE BUFFERING FOR MBPOOL
    IF MBPOOL IS FREE
     {
    TRIGGER TRANSFER FOR 6 BLOCKS OF MB FROM SD TO PL
    WRITE GOB HEADER
    ENCODE GOB
    {
    //FOR ALL MBs
            {
             IF MBPOOL IS FREE
             {
                    WAIT FOR THE PREVIOUS TRANSFER
                    CHECK IF ONE GOB HAS FINISHED,
                    IF YES THEN BREAK
                    TRIGGER DCT DSE
                    TRIGGER NEXT TRANSFER OF 6 BLOCKS
                     OF MB FROM SD TO PL
```

```
                    }
                    LOOK FOR FREE MB
                    }
                }
            }
            }
}
```

Each macro block is processed in following steps:

I frame processing:

1. Fetch input macro block from DRAM (PE): In this step one buffer is being filled by the MTC while other buffer is used by the next stage of processing Intra prediction.
2. Intra prediction (DSE): The following steps are performed in sequential manner for each 8x8 sub block with in macro block. These steps are executed on two DSEs one for luma processing and one for chroma processing. The chroma processing is same as luma processing except prediction.

```
For all sub blocks
{
    Transform and Quantisation (DSE)
    VLC (DSE)
}
```

3. Write reconstructed data to DRAM (PE): In this step one buffer is being filled by the DSEs with reconstructed macroblock while other buffer is being transferred to DRAM.
4. VLC (DSE): During VLC processing buffer is switched when one buffer is full. PE will transfer the encoded data into DRAM with the help of MTCs.
5. These steps are repeated for every macroblock until the I frame reconstruction and encoding is complete.

Variable Length Encoding

Variable length coding technique was used for entropy coding. In H.261 macroblock addresses, macro block types, motion vectors, coded block patterns & transform coefficients are coded using VLC.

Motion estimation

Motion Estimation (ME) is one of the computational intensive operations of a video codec. ME for video coding have taken an entirely different direction with the emergence of zonal algorithms. These algorithms made it possible to significantly reduce, if not almost eliminate, the tremendous computational overhead of motion estimation that was incurred in a video encoding system, while having little if any loss in quality.

Encoder API

Following *EncParamStruct* structure must be initialized to valid parameters to configure encoder. This structure pointer is input to the *Init_encoder()* function.

```
typedef struct EncParamStruct
{
    int PictureFormat; // input frame
    int FrameRate;
   int QUpdateFrequency;        // CIF
}EncParamStruct;
```

init_encoder() : Initializes the encoder resources including DSEs, local memory and configuration parameters as per the *EncParamStruct* structure values.

The target frame rate is input to encoder for rate control calculations. The typical usage is as follows: *EncParamStruct.frame_rate = 30;*

API Functions

After initialization encoder will be ready to receive commands. The current implemented commands are START, STOP, STATUS. These commands are processed by *Encode_command()* function. The API consists of the following functions:

Void Encode_command (START)

This function is used to start the Encoder. *EncParamStruct* must be initialized before starting the Encoder.

void Encode_command(STOP)

This function is used to stop the Encoder.

void Encode_command(STATUS)

This function is used to get the status of the Encoder. The status is returned by the function in *EncStatusStruct* structure.

```
typedef struct EncStatusStruct
{
    int QP;          // Current QP
    int rate;        // xxxx: current bit rate in kbps
    int frame_rate;          // xxx: current frame rate
}EncStatusStruct;
```

4.2.2 Decoder Design

Decoder Specifications

Maximum frame size: 352x288 (CIF)
Frame format: 4:2:0 (YUV)
Frame rate: 5-25
Bit rate: 2 Mbps
MB Types supported: 16x16
Entropy coding: Huffman
Reference frame: I

Decoder resource requirement

On chip resources:

1 PE
2 DSEs
12 KB

Off chip resources:

DRAM: 2*Frame size.

Decoder limitations

Only CIF 420 Support

Block Diagram:

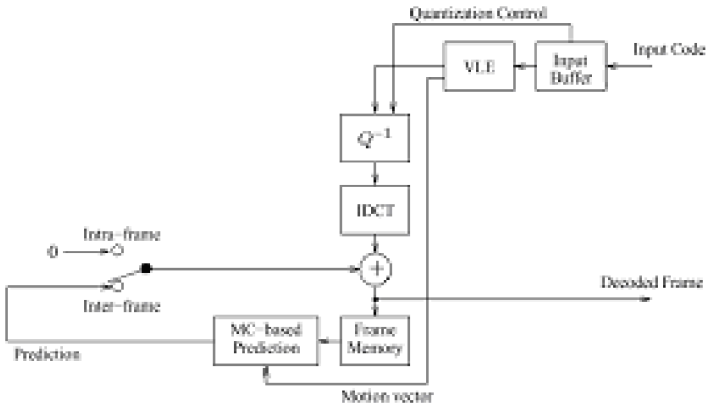


Figure 4.3 Decoder Block Diagram

Decoder design

The decoder is implemented using 2 DSEs and 1 PE, with I frame support. IDCT DSE requires 304 instructions and VLD DSE requires 280 instructions.

The implementation is in modular form, where each processor performs predefined tasks, PE does main header parsing and triggering of MTE required for transfer of packets from DRAM to Local memory. DSE 4 is dedicated for Variable Length Decoding (VLD) at macroblock level, which internally first decodes macroblock, then dequantize it. DSE 5 calculate IDCT of the macroblock and transfer the decoded block to DRAM.

Decoder can be modeled as a sequence of processes, which does, decoding process at macroblock by macroblock. PE triggers MTE to bring encoded stream from DRAM to local memory one packet at a time.

I frame Decoding:

1. PE decodes H261 Header, Picture Header, and GOB Header. PE also takes care of transfer of Next RTP Packet in case of end of Current RTP Packet.
2. PE initializes both DSEs, and gives the starting of the stream from the point DSE4 decodes the MB Header. DSE4 also does de-quantization of the decoded coefficients.

3. After DSE4 finishes one MB, then it triggers for DSE5 for Inverse DCT. There is pipeline of three MB, will be maintained between these two DSEs. If DSE4 is decoding present MB, then DSE5 will do processing on previous decoded MB.
4. DSE5 transfer fully decoded macroblock to DRAM.

Following section list pseudo code of the decoder control program, which is executed on PE.

Pseudo Code

Load Code & DPDMs for variable length decoding on DSE4.
Load Code & DPDMs for Inverse DCT on DSE5.
Load Code & DPDMs for renderring on DSE6.
Initialize The MBPOOL To Zeros Using Constant Fill
Initialize The MBPOOL Structure and Initialize Them As free
Start All three DSEs.

Read the Payload Address, Total number of packets.
Trigger a transfer of packet to Local memory from DRAM;
Wait for transfer complete;

For all frames
{ Decode Picture Header.
// For all Group of Blocks
{ Decode GOB Header.
//For All Macro block
{ Decode MB Header.
Fill the DPDM of DSE4 for Variable length decoding.
Give Start Command to DSE5.
Wait until it completes.
}
}
}
Following section list pseudo code of the decoder control program, which is executed on DSE 4.
While(1)

```
{
  Wait for start command from PE.
  If received
  {
      Do variable length decoding of the ecoded stream.
      Fill the Variable length decoding result in MBPOOL
      buffer.
      Fill DPDMs of IDCT DSE i.e. Address Of MBPOOL.
      Start the IDCT Dse.
   }
}
```

Following section list pseudo code of the decoder control program which is executed on DSE5.

```
While(1)
{           Wait for start command from VLD DSE.
            If received
            {  Do inverse DCT of Macroblock.
                Transfer the result in DRAM.
            }
}
```

Psuedo code for Inverse DCT

1) *Read MBPOOL_STRUCT structure and collect the needed information*

```
typedef struct MBPOOL_STRUCT
{
   int MBStatus;
   int CurFrameptr;
   int RefFrameptr;
   short Xoffset;
   short Yoffset;
   unsigned short imagewidth;
   unsigned short imageheight;
   int MBStructureAddr;
} MBPOOL;
```

2) Read MBAstructure

```
typedef struct MB_STRUCT
{
    int MBA;
    int MBAStuffing;
    int MBQuant;
    int MBType;
    int CBP;
    int MVDH;
    int MVDV;
} MB;
```

3) Find IDCT by calling dse_idct function.

```
Dse_idct()
{
  Calculate Row IDCT
  Save the temporary result in DPDMs
  Take transpose
  Calculate Column IDCT
}
```

Decoder API

API Functions

void Init_Decoder(void) : Initializes the decoder resources including DSEs, local memory. The current implemented commands are START, STOP, STATUS. These commands are processed by Decode_command() function. The API consists of the following functions:

void Decode_command(START)

This function is used to start the Decoder.

void Decode_command(STOP)

This function is used to stop the Decoder.

4.3 TRANSMISSION AND RECEPTION DEVICES

On chip resources:

1 PE for transmission and 1 PE for reception

1 MTE for transmission and 1 MTE for reception

Off chip resources:

Manchester Encoder and Decoder

RF Modems

Protocol implemented:

HDLC

Mode of communication

Serial at 1Mbps, asynchronous for video, 76 kbps synchronous for Audio and data.

Structure for Transmission

```
typedef struct data_out_buffer_st
{
        int  buffer_status; //FULL :1 empty: 0
        int dataout_buffer_addr;
        struct data_out_buffer_st      *next_buffer_addr;
        int packet_count;
}data_out_buffer_st;
```

Pseudo Code for Transmitter

Step1: Check status of the Buffer . If status is BUFFER_BUSY
Go to step 1, else go to step 2.

Step2: Set Number of Valid Packets, go to step 3

Step3: Start_CRC, Tx_Start_Of_Packet and set buffer status as BUFFER_BUSY, go to step4.

Step4: Tx_Data and Calculate CRC go to step 5.

Step5: Tx_checksum & Transmit End_of_Packet go to step 6

Step6: Decrement no_of_packets goto step 7.

Step7: Check if (no_of_packets>0) go to step3 else step 8.

Step8: Switch to next tx_buffer and go to step1.

Structure for Reception

```
typedef struct data_in_buffer_st
{
        int  buffer_status; //FULL :1 empty: 0
        int datain_buffer_addr;
        struct data_in_buffer_st        *next_buffer_addr;
        int packet_count;
}data_in_buffer_st;
```

Pseudo code for Reception:

Step1: Check the Buffer status , If EMPTY go to step 2, else go to Step1.

Step2: Receive Start_of_Frame
Receive Data_alignment_bits
Receive data_bits
Calculate CRC on the received data_bits

Step3: Check Rx_Check_sum = = Tx_Check_sum
If yes goto step 4, else goto step 1.

Step4: Write Packet Count, Change status to FULL, jump to next Next_buffer_addr go to step 1.

4.4 Audio Codec Design

Specifications:

Audio output at 64kbps.

Algorithm used:

G711 Audio compression Algorithm.

Resource allocation:

1 PE - used for the process control

1 DSE - used for the A-law Companding

1 MTE - used for the audio data streaming into the SDRAM
From the PIO SIO Devices.

Memory allocation:

- In this scenario three buffers of 60 samples of audio data have been allotted in SDRAM.
- Audio –In device collects the data and stores in these buffers. i.e. int audio_in_buffers [3][240];
- A buffer of 240 pcm audio samples is allocated in the local memory i.e. int audio_in_buffer_pl[240];
- A buffer of 60 compressed samples is allocated in the local memory i.e. char en_audio_out_pl[60];
- The compressed data from the local memory is transferred into the SDRAM where three buffers have been allocated i.e. Char en_audio_out_sd[3][60];

Similarly, buffers as mentioned below are allocated for the encoded input data to be decoded and driven out through the Audio-Out device.

Char en_audio_in_pl[60];

Char en_audio_in_sd[3][60];

Int audio_out_buffer_pl[240];

Int audio_out_buffers[3][240];

Pseudo Code for the PE:

Step1: initialize the buffers and the flags of the buffers of all the PE's goto step 2.

Step2: check the current audio in buffer flag to be free and goto step3 else goto step 2

Step3: check the buffer flag of the private local memory audio_in_buffer_pl. If FREE goto step4 else goto step3

Step 4: transfer data from SDRAM buffer to Private Local Memory for encoding the data and set the encoder flag to encode the data and goto step5.

Step 5: wait for the DSE to decode the data. If completed goto step 6 else goto step5.

Step 6: provide command for encoding the data in audio_in_buffer_pl goto step 7.

Step 7: Transfer the decoded data in audio_out_buffer_pl, into the audio_out_buffer i.e from Local Memory to SDRAM and switch the audio_out_buffer. Goto step 8

Step 8: check untill flag status of the en_audio_in_sd indicates that the SDRAM is ready with the data, then transfer data from SDRAM to Private Local Memory i.e. en_audio_in_pl and go to step9

Step 9: wait for the DSE to encode the data. If completed go to step 10 else goto step 9

Step 10: provide command for the DSE to decode goto step 11

Step 11: transfer data from en_audio_out_pl to en_audio_out_sd buffers i.e. from Private Local memory to SDRAM. Goto step 12

Step 12: switch the audio_in_buffer and goto 2

Pseudo code for DSE:

Encoder:

Step1: check the sign of the input data and set the sign bit "S" of the output data
Step 2: verify the segment number and write the segment number in three bits M2 M1 M0
Step3: set the quantized value in that segment and write in the last four bits i.e Q3 Q2 Q1 Q0
Now the encoded data appears as
S M2 M1 M0 Q3 Q2 Q1 Q0

Decoder:

Step1: check the sign of the input data and set the sign bit "S" of the output data
Step 2: verify the segment number and set the base value to be added to the linearly dequantized value
Step3: set the dequantized value in that segment and add the base value representing data in 16 bits

Chapter-5

Results

SUMMERIZED RESULTS:

These are the number of cycles taken to perform the specific tasks of the algorithm, performed over one **Macro Block** at a clock rate 230 MHz. Detailed results are attached in the Appendix C. And Results in the form of snapshots are available in the CD attached with this thesis.

Frame rate: 17 fps (Maximum)

Encoder Section:

1.For DCT: 839 cycles.

2.For DCT, IDCT and Quantization: 36289 cycles.

3.For Variable length coding: 12630cycles.

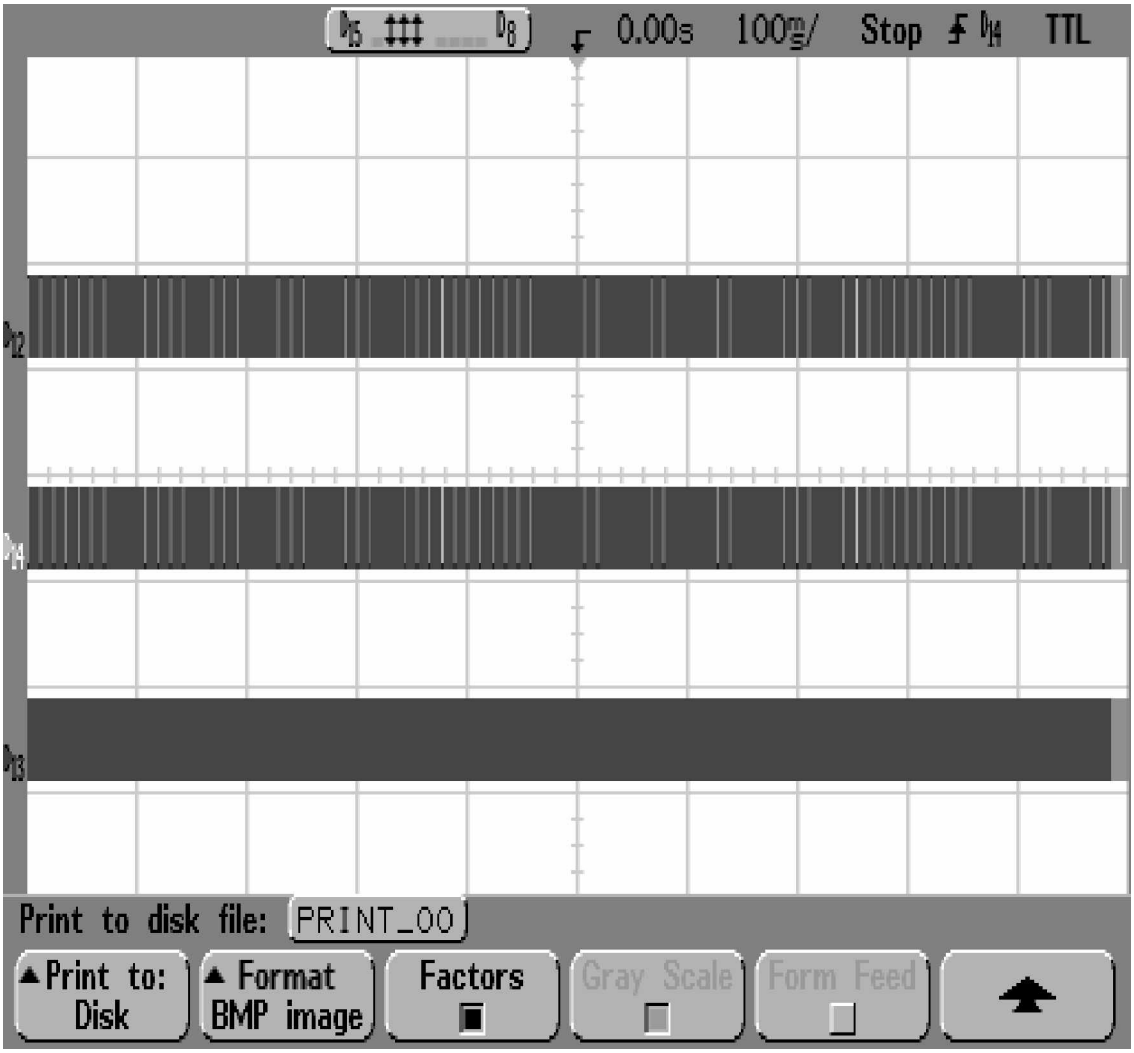
Decoder Section:

1.For IDCT: 919 cycles

2.For Variable Length decoding Inverse Quantization and Inverse zigzag scan : 14149 cycles.

Audio is transmitted and received at 64 kbs.

Snap Shot 1 from the logic analyzer

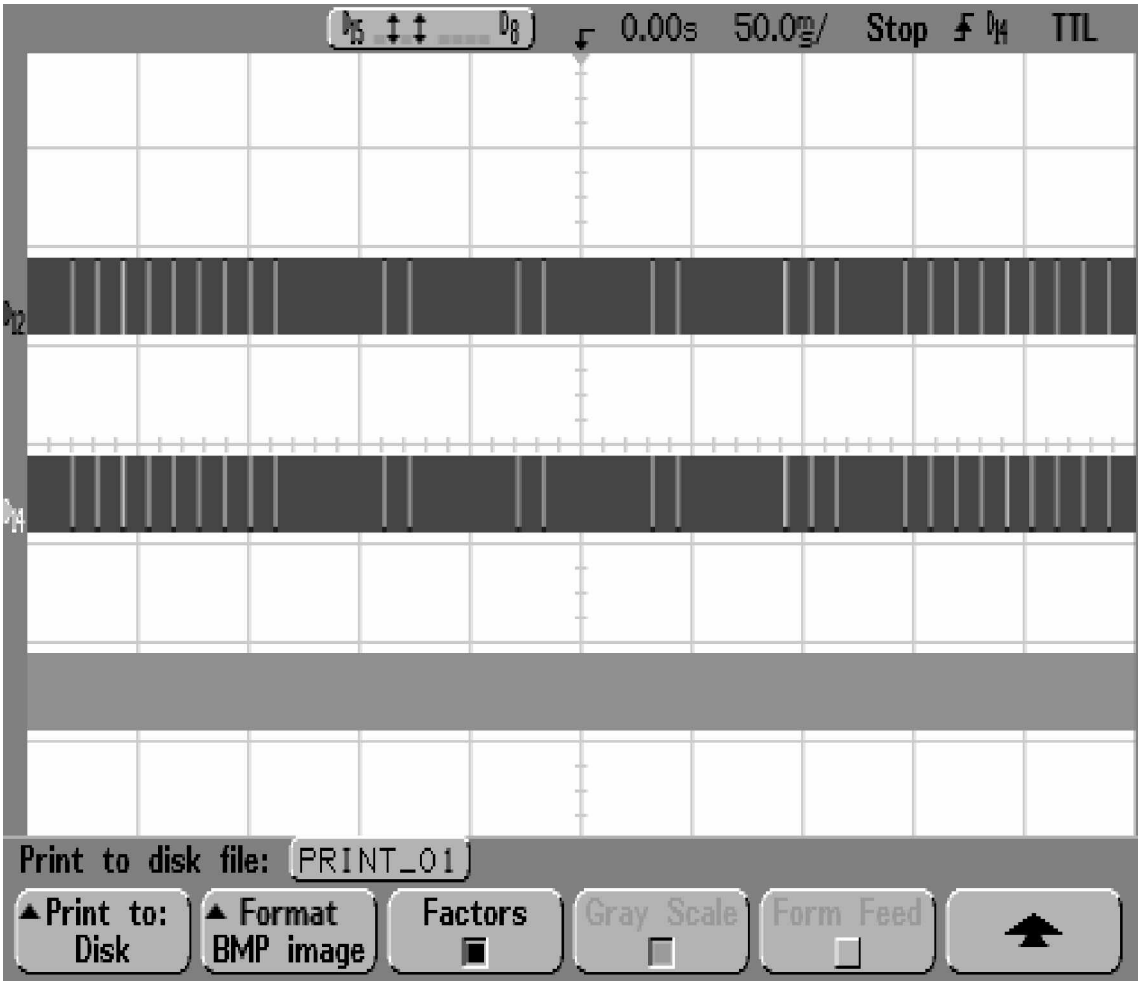


D12 shoving the data format at Transmitter pin

D14 showing the received data after Manchester encoding and decoding at the receiver input

D13 Clock-out at transmitter end.

Snap Shot 2 from the logic analyzer



Chapter-6

Conclusion & Future Scope

This project has been completed and the system can be used for remote video surveillance system with the specifications mentioned above. Further improvements can be done by optimizing the code to give 30 fps. This particular system is working with CIF format and giving 17fps now. As per need this can be easily modified to give QCIF or D1 format. D1 can be achieved by scaling the CIF to 640x480.

The serial communication is set up using the HDLC protocol and for clock recovery Manchester encoding and decoding devices are used externally.

FUTURE-SCOPE:

For making the system more intelligent like object identification, more interactivity at the user end one can use it for basic understanding of the system and can further go for h.264 kind of algorithm. Similarly regarding the Audio in this project G.711 standard is implemented, but to get more compression G.729 can be used.

In the future more and more features and interactivity can be provided with lesser and lesser bit rate. Newer and enhanced algorithms can be used for implementation of the standard.

References

[1] ITU-T recommended H.261 standard, "Video Codec for Audiovisual services at px 64 kbits/s" June 1994.

[2] J.L.Mitchell, W.B.Pennebaker, C.E.Fogg and D.J.LeGall, MPEG Video Compression Standard", in Digital Multimedia Standards Series, Chapman & Hall, New York, NY, 1997.

[3] Amit Gulati and George Campbell, " Efficient mapping of the H.264 encoding algorithm onto Multiprocessor DSPs"

[4] Christopher Loeffler, Adriaan Liebenberg, and George S. Moschytz, "Practical Fast 1D DCT Algorithms With 11 Multiplications" IEEE transaction on signal processing, Vol.9, January 24,1977, pp. 988-991.

[5] Ephraim Feig and Shmael Winograd,"Fast Algorithms for the Discrete Cosine Transform" IEEE transaction on signal processing, Vol 40, NO.9 September 1992, pp. 2174-2193.

[6] Jumehwa Song and Boon-Lock Yeo, "A Fast Algorithm for DCT-Domain Inverse Motion Compensation Based on Shared Information in a Macro block" IEEE transaction on circuits and systems for video technology, Vol.10, No.5, August 2000, pp.767-775.

[7] Wen-Hsiung Chen, C. Harrison Smith, And S. C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform" IEEE trasaction on communications, Vol. Com. 25, No.9., September 1997, p. 1004 - 1009

[8] N.I.Cho and S.U.Lee, "Fast Algorithm and Implementation of 2D Discrete Cosine transform", IEEE transaction on Circuits and Systems, Vol.38, No.3, March 1991, pp297-305.

[9] T..George Campbell, " Multimedia signal processing usinga parallel MDSP architecture".

[10] Oliver Sohm, " Variable-Length Decoding on the TMS320C6000 DSP platform" application report, SPRA805 – June 2002.

[11] Liang-Gee Chen, Wai-Ting Chen, Yeu –Shen Jehng Tzi Chuieh, "An Efficient Parallel Motion Estimation Algorithm for Digital Image Processing", IEEE transactions on Circuits and systems on Video Technology, Vol.1, No.4, Dec.1991, pp378-384.

[12] Lai-Man Po, Wing-Chung Ma, "A novel Four-Step Search Algorithm for fast block matching", IEEE transactions on Circuits and Systems on Video Technology, Vol.6, No.3, Jun.1996, pp313-317.

[13] R.Li, B.Zeng, M.L.Liu, "A new three-step search algorithm for block motion estimation" IEEE transactions on Circuits and Systems on Video Technology, Vol.4, No.4, Aug.1994, pp438-442.

[14] Shan Zhu, Kai-Kuang Ma, "A new Diamond search algorithm for fast block matching", IEEE transactions on circuit and systems on video technology, Vol.9, No.2, Feb.2000, pp287-290

[15] L.Chiariglione, "MPEG and Multimedia communications", IEEE Trans. CSVT, Vol.7, No.1, Feb.1997.

[16] Deepak Turaga, Mohamed Alkanhal, "Mid-Term project on Search Algorithms for Block-Matching in Motion Estimation".

[17] Chaiytan sukhadia, " Documentation on Basic Concepts of MPEG-4 Motion Estimation".

[18] Documentation of the signal compression and error resilience training program at Nirma University .

[19] D.J. Le Gall, "The MPEG Video Compression Algorithm",Signal Processing: Image Communication 4, No. 2, 1992,pp. 129-140

[20] J.L.Mitchell, W.B.Pennebaker, C.E.Fogg and D.J.LeGall, "MPEG Video Compression Standard", in Digital Multimedia Standards Series, Chapman & Hall, New York, NY, 1997.

[21] B. G. Haskell, A. Puri and A. N. Netravali, "Digital Video: An Introduction to MPEG – 2", ISBN: 0-412-08411-2, Chapman & Hall, 1997.

[22] R.Schäfer and T.Sikora, " Digital Video Coding Standards and Their Role in Video Communications", Proceedings of the IEEE, Vol.83, No.6, June 1995.

[23] T.Sikora, " The MPEG - 4 Video Standard Verification Model", IEEE Trans. CSVT, Vol.7, No.1, Feb.1997. (HTML-Version / PS-Version)

[24] T.Sikora, " MPEG - 4 Very Low Bit Rate Video", Proc. IEEE ISCAS Conference, Hongkong, June 1997. (HTML Version / PS-Version)

[25] R.Gonzalez and R.Woods, "Digital Image Processing", 2nd Ed., Pearson Education, 2002. ISBN 81-7808-629-8.

[26] I. Richardson, "H.264 and MPEG-4 Video Compression – video coding for next generation multimedia", John Willey, 2003, ISBN 0-470-84837-5.

URL Links

[27] www.cradle.com
[28] www.jpeg.org
[29] www.mpeg.org
[30] http://www.m4if.org/resources/profiles/index.html
[31] www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm
[32] http://www.dspexperts.com/dsp/projects/projects.html
[33] www.vcodex.com/resources.html
[34] www.efg2.com/Lab/Library/ImageProcessing/Algorithmshtm#GeneralAlgorithms
[35] www.ece.utexas.edu/~bevans/courses/ee381k/projects/fall98/chen-lu-wang/presentation/

APPENDIX A: List of tables in standard h261

TABLE 1/H.261

VLC table for macroblock addressing

MBA	Code	MBA	Code
1	1	17	0000 0101 10
2	011	18	0000 0101 01
3	010	19	0000 0101 00
4	0011	20	0000 0100 11
5	0010	21	0000 0100 10
6	0001 1	22	0000 0100 011
7	0001 0	23	0000 0100 010
8	0000 111	24	0000 0100 001
9	0000 110	25	0000 0100 000
10	0000 1011	26	0000 0011 111
11	0000 1010	27	0000 0011 110
12	0000 1001	28	0000 0011 101
13	0000 1000	29	0000 0011 100
14	0000 0111	30	0000 0011 011
15	0000 0110	31	0000 0011 010
16	0000 0101 11	32	0000 0011 001
		33	0000 0011 000
		MBA stuffing	0000 0001 111
		Start code	0000 0000 0000 0001

Run is a 6 bit fixed length code

Run	Code
0	0000 00
1	0000 01
2	0000 10
ξ	ξ
ξ	ξ
63	1111 11

Level is an 8 bit fixed length code

Level	Code
–128	FORBIDDEN
–127	1000 0001
ξ	ξ
–2	1111 1110
–1	1111 1111
0	FORBIDDEN
1	0000 0001
2	0000 0010
ξ	ξ
127	0111 1111

VLC table for TCOEFF

Run	Level	Code
EOB		10
0	1	1s[a] If first coefficient in block
0	1	11s Not first coefficient in block
0	2	0100 s
0	3	0010 1s
0	4	0000 110s
0	5	0010 0110 s
0	6	0010 0001 s
0	7	0000 0010 10s
0	8	0000 0001 1101 s
0	9	0000 0001 1000 s
0	10	0000 0001 0011 s
0	11	0000 0001 0000 s
0	12	0000 0000 1101 0s
0	13	0000 0000 1100 1s
0	14	0000 0000 1100 0s
0	15	0000 0000 1011 1s
1	1	011s
1	2	0001 10s
1	3	0010 0101 s
1	4	0000 0011 00s
1	5	0000 0001 1011 s
1	6	0000 0000 1011 0s
1	7	0000 0000 1010 1s
2	1	0101 s
2	2	0000 100s
2	3	0000 0010 11s
2	4	0000 0001 0100 s
2	5	0000 0000 1010 0s
3	1	0011 1s
3	2	0010 0100 s
3	3	0000 0001 1100 s
3	4	0000 0000 1001 1s
4	1	0011 0s
4	2	0000 0011 11s
4	3	0000 0001 0010 s
5	1	0001 11s
5	2	0000 0010 01s
5	3	0000 0000 1001 0s
6	1	0001 01s
6	2	0000 0001 1110 s
7	1	0001 00s
7	2	0000 0001 0101 s
8	1	0000 111s
8	2	0000 0001 0001 s
9	1	0000 101s
9	2	0000 0000 1000 1s
10	1	0010 0111 s
10	2	0000 0000 1000 0s
11	1	0010 0011 s
12	1	0010 0010 s
13	1	0010 0000 s
14	1	0000 0011 10s
15	1	0000 0011 01s
16	1	0000 0010 00s
17	1	0000 0001 1111 s
18	1	0000 0001 1010 s
19	1	0000 0001 1001 s
20	1	0000 0001 0111 s
21	1	0000 0001 0110 s
22	1	0000 0000 1111 1s
23	1	0000 0000 1111 0s
24	1	0000 0000 1110 1s
25	1	0000 0000 1110 0s
26	1	0000 0000 1101 1s
Escape		0000 01

a) Never used in INTRA macroblocks.

Appendix:B Snap Shots

ORIGINAL IMAGE

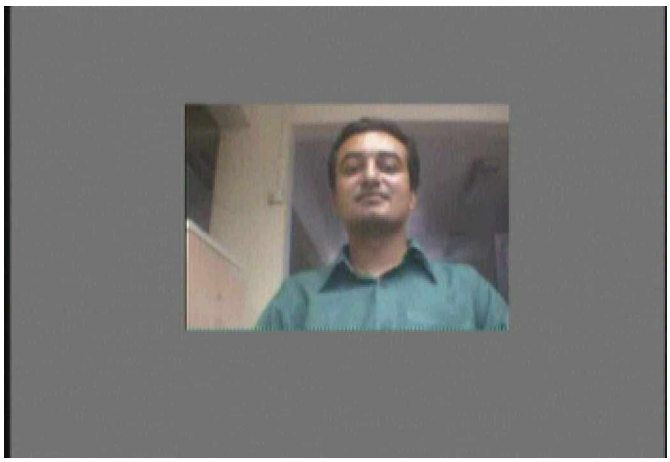
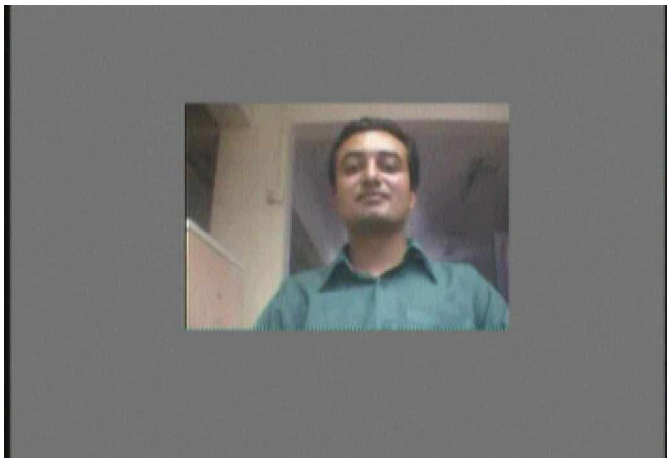


IMAGE AFTER DCT AND IDCT



ORIGINAL IMAGE

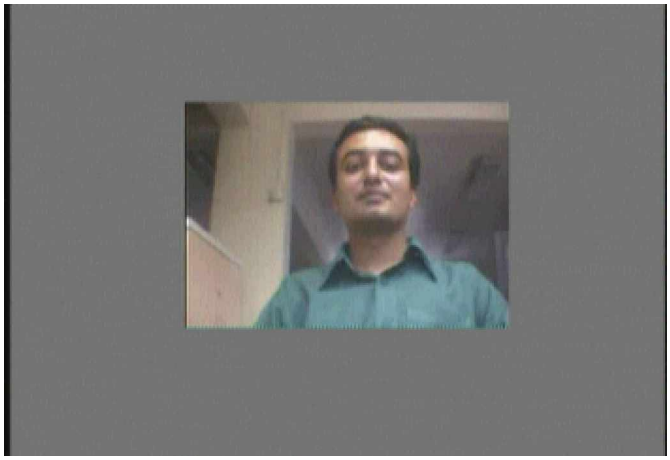
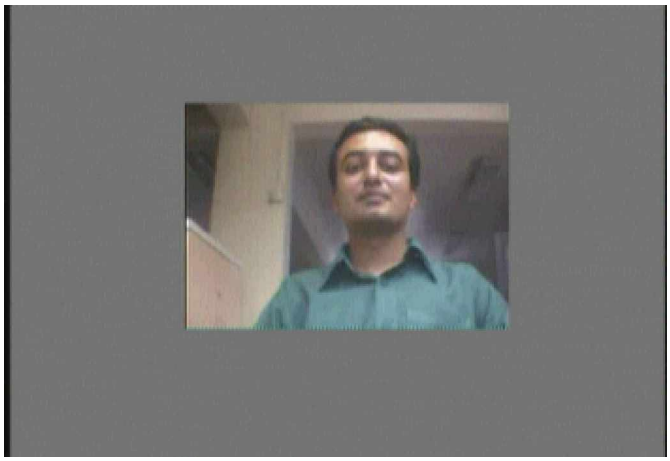


IMAGE AFTER ME AND MC



Appendix C Profiler Results

```
                          CRADLE STATISTICS
                     Generated on 2005/04/16 10:33:33

Cradle Architecture Version: 3400
Simulator Version:           4.0.011
Total Simulation Time:       26325614 cycles

*** PROCESSOR STATISTICS ***

QUAD  0 ============================================================
            Active        DLY        WFI  Read-Wait    I-Cache    D-Cache
            Cycles     Cycles     Cycles     Cycles     Misses     Misses
         ---------- ---------- ---------- ---------- ---------- ----------
PE 0      26325612      26065          0    9273700        729          0
PE 1             0          0          0          0          0          0
PE 2             0          0          0          0          0          0
PE 3             0          0          0          0          0          0
            Active      Delay     Stalls    D-Cache
            Cycles     Cycles     Cycles     Misses
         ---------- ---------- ---------- ----------
DSE 0            0          0          0          0
DSE 1            0          0          0          0
DSE 2            0          0          0          0
DSE 3            0          0          0          0
DSE 4      3968062    2258880     990313          0
DSE 5      3968142          0    1547182          0
DSE 6      3943337          0    3730690          0
DSE 7            0          0          0          0
            Active  Read-Wait    D-Cache
            Cycles    Cycles      Misses
         ---------- ---------- ----------
MTE 0,0      16308       3496          0
MTE 0,1      51555       9742          0
MTE 0,2      23063       4594          0
MTE 0,3      50906       8346          0

IO QUAD ============================================================
            Active        DLY        WFI  Read-Wait    I-Cache    D-Cache
            Cycles     Cycles     Cycles     Cycles     Misses     Misses
         ---------- ---------- ---------- ---------- ---------- ----------
PE 0       7381603          0          0    5778200        569          0
PE 1       4024250          0          0    1845100        200          0
            Active  Read-Wait    D-Cache
            Cycles    Cycles      Misses
         ---------- ---------- ----------
MTE 0,0          0          0          0
MTE 0,1          0          0          0
MTE 0,2          0          0          0
MTE 0,3          0          0          0
MTE 1,0          0          0          0
MTE 1,1          0          0          0
MTE 1,2          0          0          0
MTE 1,3          0          0          0
```

```
*** LOCAL BUS STATISTICS ***

QUAD  0 ================================================
              Local Bus Read         Local Bus Write
              Trans      Bytes       Trans      Bytes
          ---------- ----------  ---------- ----------
PE 0          209141     441565      508942    1979787
PE 1               0          0           0          0
PE 2               0          0           0          0
PE 3               0          0           0          0
DSE 0              0          0           0          0
DSE 1              0          0           0          0
DSE 2              0          0           0          0
DSE 3              0          0           0          0
DSE 4         141496     566276         225        682
DSE 5         178806     703720        3362      13448
DSE 6          77320     154640           0          0
DSE 7              0          0           0          0
MTE 0,0          336       1600        3566      14264
MTE 0,1         2384      16848        1833       7332
MTE 0,2          540       2944        2951      11804
MTE 0,3         2396      17440        1810       7240
External           0          0           0          0
          ---------- ----------  ---------- ----------
Totals        612419    1905033      522689    2034557
Local Bus utilization: 4.312%

IO QUAD ========================================================
              Local Bus Read         Local Bus Write
              Trans      Bytes       Trans      Bytes
          ---------- ----------  ---------- ----------
PE 0          115593     210325       46308     166861
PE 1           36950      67924       15067      55639
MTE 0,0            0          0           0          0
MTE 0,1            0          0           0          0
MTE 0,2            0          0           0          0
MTE 0,3            0          0           0          0
MTE 1,0           17         68        6978       7080
MTE 1,1            0          0           0          0
MTE 1,2            0          0           0          0
MTE 1,3            0          0           0          0
External           0          0           0          0
          ---------- ----------  ---------- ----------
Totals        152560     278317       68353     229580
Local Bus utilization: 0.839%

*** DRAM STATISTICS ***

QUAD  0 ================================================================
                    DRAM-Read                          DRAM-Write
               Data                      Page       Data                      Page
              Trans      Bytes         Misses      Trans      Bytes         Misses
          (no. of 4
        octet packets)
          ---------- ---------- ---------- ---------- ---------- ----------
PE 0          94330     402847      22994     494454    1943371      15966
PE 1              0          0          0          0          0          0
PE 2              0          0          0          0          0          0
PE 3              0          0          0          0          0          0
DSE 0             0          0          0          0          0          0
DSE 1             0          0          0          0          0          0
```

```
DSE 2            0          0          0          0          0          0
DSE 3            0          0          0          0          0          0
DSE 4            0          0          0          0          0          0
DSE 5         3437      13748       1333          0          0          0
DSE 6        38660     154640       1599          0          0          0
DSE 7            0          0          0          0          0          0
MTE 0,0         34       1088          0          0          0          0
MTE 0,1        135       2224         42       1248       4992        193
MTE 0,2         43       1376         43         96        384         15
MTE 0,3        108       1728         36       1344       5376        202

IO QUAD ============================================================
                    DRAM-Read                       DRAM-Write
              Data                      Page       Data                      Page
             Trans      Bytes      Misses      Trans      Bytes      Misses
          ---------- ---------- ---------- ---------- ---------- ----------
PE 0         58350     228495      24810      44391     163995      15916
PE 1         18651      74258       1244      14372      54205        151
MTE 0,0          0          0          0          0          0          0
MTE 0,1          0          0          0          0          0          0
MTE 0,2          0          0          0          0          0          0
MTE 0,3          0          0          0          0          0          0
MTE 1,0          0          0          0          0          0          0
MTE 1,1          0          0          0          0          0          0
MTE 1,2          0          0          0          0          0          0
MTE 1,3          0          0          0          0          0          0
          ---------- ---------- ---------- ---------- ---------- ----------
Totals      213748     880404      52101     555905    2172323      32443

DRAM read delay:  100 cycles
DRAM banks:       8
DRAM line size:   2048 bytes
DRAM utilization: 12.658% (access+miss)/(total_cycles)



Total Subroutine call(s) for PE#0= 59 in Quad #0.
Total Subroutine call(s) for PE#1= 41 in Quad #1.
Total Subroutine call(s) for PE#0= 64 in Quad #1.
Total call to DseDecodeMB() subroutine at address 0x40003418 is 1 time(s)
                        Cumulative cycles 482.
                        Exclusive cycles 482
Total call to DseDecodeMB() subroutine at address 0x40001000 is 1 time(s)
                        Cumulative cycles 13619790.
                        Exclusive cycles 13019387
Total call to initLoadTable() subroutine at address 0x400013f8 is 1 time(s)
                        Cumulative cycles 31291.
                        Exclusive cycles 12627
Total call to read_control_word() subroutine at address 0x4000a0a0 is 3
time(s)
                        Cumulative cycles 4756.
                        Exclusive cycles 1308
Total call to read_control_word() subroutine at address 0x40009f88 is 29
time(s)
                        Cumulative cycles 32217.
                        Exclusive cycles 32217
Total call to quad_get_index() subroutine at address 0x40009f70 is 1
time(s)
                        Cumulative cycles 10102.
                        Exclusive cycles 478
Total call to quad_get_index() subroutine at address 0x4000c3b0 is 1
time(s)
                        Cumulative cycles 9624.
                        Exclusive cycles 1680
```

```
Total call to quad_get_num_r_quads() subroutine at address 0x4000c320 is 16
time(s)
                        Cumulative cycles 17944.
                        Exclusive cycles 16006
Total call to get_base_address() subroutine at address 0x4000ad18 is 19
time(s)
                        Cumulative cycles 34246.
                        Exclusive cycles 13250
Total call to quad_in_pio_quad() subroutine at address 0x40009f08 is 7
time(s)
                        Cumulative cycles 16226.
                        Exclusive cycles 4427
Total call to pe_getsreg() subroutine at address 0x400059a0 is 1 time(s)
                        Cumulative cycles 1422.
                        Exclusive cycles 1422
Total call to get_pe_quad_base_addr() subroutine at address 0x40009c08 is 3
time(s)
                        Cumulative cycles 9311.
                        Exclusive cycles 1682
Total call to get_pe_quad_base_addr() subroutine at address 0x4000ad50 is 3
time(s)
                        Cumulative cycles 7629.
                        Exclusive cycles 4468
Total call to get_base_address() subroutine at address 0x40009b90 is 3
time(s)
                        Cumulative cycles 7915.
                        Exclusive cycles 1782
Total call to semaphore_global_alloc() subroutine at address 0x40009c28 is
2 time(s)
                        Cumulative cycles 12202.
                        Exclusive cycles 1670
Total call to GlobalSemaphoreAlloc() subroutine at address 0x4000adc0 is 2
time(s)
                        Cumulative cycles 8476.
                        Exclusive cycles 5920
Total call to write_control_word() subroutine at address 0x4000a738 is 8
time(s)
                        Cumulative cycles 8544.
                        Exclusive cycles 2440
Total call to write_control_word() subroutine at address 0x4000c510 is 8
time(s)
                        Cumulative cycles 6104.
                        Exclusive cycles 6104
Total call to _MTE_load_default_mte_code_on_quad() subroutine at address
0x40009e98 is 1 time(s)
                        Cumulative cycles 523258.
                        Exclusive cycles 492
Total call to MTE_load_default_mte_code_on_quad() subroutine at address
0x4000c088 is 1 time(s)
                        Cumulative cycles 522766.
                        Exclusive cycles 2682
Total call to MTE_load_mte_code_on_quad() subroutine at address 0x4000c190
is 1 time(s)
                        Cumulative cycles 515761.
                        Exclusive cycles 493348
Total call to get_quad_base_addr() subroutine at address 0x4000cad8 is 7
time(s)
                        Cumulative cycles 24926.
                        Exclusive cycles 6112
Total call to main() subroutine at address 0x40005848 is 1 time(s)
                        Cumulative cycles 8682603.
                        Exclusive cycles 979
Total call to start_program_num() subroutine at address 0x4000a580 is 2
time(s)
                        Cumulative cycles 8681624.
```

```
                        Exclusive cycles 6348
Total call to launch_program_num() subroutine at address 0x4000b2f0 is 2
time(s)                 Cumulative cycles 8673620.
                        Exclusive cycles 8573631
Total call to lock_load_table_semaphore() subroutine at address 0x4000bd78
is 2 time(s)
                        Cumulative cycles 9475.
                        Exclusive cycles 1933
Total call to semaphore_lock() subroutine at address 0x4000a328 is 2
time(s)
                        Cumulative cycles 1108.
                        Exclusive cycles 1108
Total call to launch_allocate_memory() subroutine at address 0x4000af88 is
2 time(s)
                        Cumulative cycles 74160.
                        Exclusive cycles 20427
Total call to malloc() subroutine at address 0x4000ccf0 is 4 time(s)
                        Cumulative cycles 31540.
                        Exclusive cycles 2512
                        Exclusive cycles 9909
Total call to __malloc_unlock() subroutine at address 0x4000d498 is 4
time(s)
                        Cumulative cycles 48.
                        Exclusive cycles 48
Total call to semaphore_unlock() subroutine at address 0x4000a368 is 2
time(s)
                        Cumulative cycles 720.
                        Exclusive cycles 720
Total call to get_pe0_quad_base_reg() subroutine at address 0x4000a448 is 1
time(s)
                        Cumulative cycles 1656.
                        Exclusive cycles 1656
Total call to video_decoder_init() subroutine at address 0x40005710 is 1
time(s)
                        Cumulative cycles 3133785.
                        Exclusive cycles 3054139
Total call to init_decoder() subroutine at address 0x40004d98 is 1 time(s)
                        Cumulative cycles 55038.
                        Exclusive cycles 4033
Total call to LoadIDCT_InitDseDpdms() subroutine at address 0x40005448 is 1
time(s)
                        Cumulative cycles 2086.
                        Exclusive cycles 2086
Total call to MTE_DSE_pload_any() subroutine at address 0x40009c88 is 3
time(s)
                        Cumulative cycles 66092.
                        Exclusive cycles 7718
Total call to MTE_Start() subroutine at address 0x40009d88 is 3 time(s)
                        Cumulative cycles 11061.
                        Exclusive cycles 4407
Total call to MTE_wait() subroutine at address 0x40009e40 is 3 time(s)
                        Cumulative cycles 40359.
                        Exclusive cycles 40359
Total call to LoadVLD_InitDseDpdms() subroutine at address 0x40005308 is 1
time(s)
                        Cumulative cycles 3648.
                        Exclusive cycles 3648
Total call to start_dse() subroutine at address 0x40005298 is 1 time(s)
                        Cumulative cycles 611.
                        Exclusive cycles 611
Total call to init_renderer() subroutine at address 0x40005520 is 1 time(s)
                        Cumulative cycles 24608.
                        Exclusive cycles 622
Total call to LoadDSEDPDMs() subroutine at address 0x40005598 is 1 time(s)
                        Cumulative cycles 2155.
```

```
                        Exclusive cycles 2155
Total call to StartRendererDse() subroutine at address 0x400056e8 is 1
time(s)                 Cumulative cycles 399.
                        Exclusive cycles 399
Total call to decode_frame() subroutine at address 0x40004f00 is 1 time(s)
                        Cumulative cycles 8276.
                        Exclusive cycles 8276
Total call to decode_GOB() subroutine at address 0x400043e0 is 1 time(s)
                        Cumulative cycles 744.
                        Exclusive cycles 744
Total call to decode_macroblock() subroutine at address 0x40004470 is 1
time(s)
                        Cumulative cycles 94228.
                        Exclusive cycles 7759
Total call to Decode() subroutine at address 0x40004c98 is 9 time(s)
                        Cumulative cycles 82789.
                        Exclusive cycles 82789
Total call to LoadVLD_DPDMs() subroutine at address 0x400053f8 is 3 time(s)
                        Cumulative cycles 3680.
                        Exclusive cycles 3680
Quad 1 E0
Total call to pio_device_video_out_init() subroutine at address 0x400041b8
is 1 time(s)
                        Cumulative cycles 1628.
                        Exclusive cycles 271
Total call to video_out_shared_st_init() subroutine at address 0x40003c68
is 1 time(s)
                        Cumulative cycles 1357.
                        Exclusive cycles 1357
Total call to video_out_load_init_device() subroutine at address 0x40004060
is 1 time(s)
                        Cumulative cycles 157.
                        Exclusive cycles 157
Total call to video_out_PG_DMA_init() subroutine at address 0x40003a28 is 1
time(s)
                        Cumulative cycles 599595.
                        Exclusive cycles 292
Total call to videoOut_init() subroutine at address 0x400041f0 is 1 time(s)
                        Cumulative cycles 599303.
                        Exclusive cycles 2721
Total call to InitializeConfigLib() subroutine at address 0x40009c80 is 1
time(s)
                        Cumulative cycles 115.
                        Exclusive cycles 115
Total call to ConfigurePG() subroutine at address 0x400064f0 is 2 time(s)
                        Cumulative cycles 26650.
                        Exclusive cycles 22348
Total call to _get_base_address() subroutine at address 0x40009b90 is 3
time(s)
                        Cumulative cycles 6474.
                        Exclusive cycles 1382
Total call to ConfigurePGExtendedPSM() subroutine at address 0x400075a0 is
2 time(s)
                        Cumulative cycles 288942.
                        Exclusive cycles 35634
Total call to ReadPGConfigReg() subroutine at address 0x4000a0c0 is 25
time(s)
                        Cumulative cycles 161100.
                        Exclusive cycles 61275
Total call to GeneratePIOaddress() subroutine at address 0x4000ac20 is 1406
time(s)
                        Cumulative cycles 2870022.
                        Exclusive cycles 2870022
Total call to WritePGConfigReg() subroutine at address 0x4000a758 is 656
time(s)
```

```
                        Cumulative cycles 4339306.
                        Exclusive cycles 1658922


Total call to ConfigurePGV3Config() subroutine at address 0x40009230 is 2
time(s)
                        Cumulative cycles 174771.
                        Exclusive cycles 30627
Total call to ConfigureUMS2PGConfig() subroutine at address 0x400096c8 is 2
time(s)
                        Cumulative cycles 106104.
                        Exclusive cycles 14418
Total call to videoOut_config() subroutine at address 0x40004380 is 1
time(s)
                        Cumulative cycles 166.
                        Exclusive cycles 166
Total call to ConfigureDMAdevice() subroutine at address 0x40005e88 is 1
time(s)
                        Cumulative cycles 444213.
                        Exclusive cycles 8743
Total call to ConfigurePGAllPins() subroutine at address 0x400066a0 is 1
time(s)
                        Cumulative cycles 130783.
                        Exclusive cycles 64936
Total call to WritePGReg() subroutine at address 0x4000a848 is 43 time(s)
                        Cumulative cycles 175309.
                        Exclusive cycles 87417
Total call to ReadPGReg() subroutine at address 0x4000a1b0 is 1 time(s)
                        Cumulative cycles 4432.
                        Exclusive cycles 2511
Total call to ConfigurePGClock() subroutine at address 0x40006c68 is 1
time(s)
                        Cumulative cycles 18040.
                        Exclusive cycles 9253
Total call to ConfigurePGTimer() subroutine at address 0x40008f70 is 1
time(s)
                        Cumulative cycles 196785.
                        Exclusive cycles 43902
Total call to ConfigurePGSerialIO() subroutine at address 0x40008cc0 is 1
time(s)
                        Cumulative cycles 39151.
                        Exclusive cycles 9900
Total call to ConfigurePGCommandStatusFIFO() subroutine at address
0x400071b8 is 1 time(s)
                        Cumulative cycles 17201.
                        Exclusive cycles 4202
Total call to ConfigurePGInterrupt() subroutine at address 0x400077b8 is 1
time(s)
                        Cumulative cycles 7211.
                        Exclusive cycles 3148
Total call to ConfigurePGDataFIFO() subroutine at address 0x40007308 is 1
time(s)
                        Cumulative cycles 26299.
                        Exclusive cycles 6685
Total call to ConfigurePGPSM() subroutine at address 0x40007888 is 1
time(s)
                        Cumulative cycles 5128132.
                        Exclusive cycles 1229399
```