DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DESIGN AND IMPLEMENTATION OF TV SET SOFTWARE AND HARDWARE TO SOLVE TECHNICAL PROBLEMS OF THE SET USING USER INTERFACE MENU

by

Serdar KILINÇARPAT

November, 2006
İZMİR

DESIGN AND IMPLEMENTATION OF TV SET SOFTWARE AND HARDWARE TO SOLVE TECHNICAL PROBLEMS OF THE SET USING USER INTERFACE MENU

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of
Dokuz Eylül University
In Partial Fulfillment of the Requirements for
the Degree of Master of Science in Computer Engineering

by
Serdar KILINÇARPAT

November, 2006
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"DESIGN AND IMPLEMENTATION OF TV SET SOFTWARE AND HARDWARE TO SOLVE TECHNICAL PROBLEMS OF THE SET USING USER INTERFACE MENU"** completed by **Serdar KILINÇARPAT** under supervision of **Ins. Dr. M. Kemal ŞİŞ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

_____

Ins. Dr. M.Kemal ŞİŞ

_____

Supervisor

_____	_____
Asst. Prof. Dr. Adil ALPKOÇAK	Prof. Dr. Haldun KARACA
_____	_____
Committee Member	Committee Member

_____

Prof. Dr. Cahit HELVACI
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to thank my supervisor " Ins. Dr .Malik Kemal ŞİŞ" for the encouragement, support and valuable solutions he produced.

I also appreciate the encouragement of the entire family members (the teaching staff) of Computer Engineering Dep. / Dokuz Eylul University.

Lastly, I gratefully would like to thank my fiancé, my family and especially my friend Hakan BULU and Semih UTKU for the encouragement, support and care.

Serdar KILINÇARPAT

DESIGN AND IMPLEMENTATION OF
TV SET SOFTWARE AND HARDWARE TO SOLVE TECHNICAL PROBLEMS
OF THE SET USING USER INTERFACE MENU

ABSTRACT

Using a Graphic User Interface(GUI) we can communicate with the TV Set electronic board through infrared remote controller. Writing to and reading from the registers of IC's on the electronic board of the TV set, we can do the remote diagnostic of technical problems, we can try possible solutions and improve software, which is embedded in the microcontroller of the board. This thesis fulfills this goal.

GUI layer is a brunch of the complex software structure of embedded TV-set. What we have written for this thesis is actually an addition to this software, which is around 0,5 kB in binary, executable format. The whole software is 128 kB and resides in the rom area of SDA555 microcontroller. Our implementation covers another user menu,which we have created for the purpose of thesis and connections to the main software, which makes the menu work compatible with the main software.

This project is mainly aimed to be used at TV television sets. This is not a restriction, but we recommend it being used in this market with a strict selftrust, since this area is our profession in industry, as an embedded software development engineer. We thereby think that this project can be used at any consumer electronic device, produced on line in high volumes.

Keywords : Graphic User Interface(GUI) , Remote Diagnostic , TV Set

KULLANICI ARAYÜZ MENÜSÜ KULLANILARAK TELEVİZYONUN TEKNİK PROBLEMLERİNİN ÇÖZÜMÜ İÇİN YAZILIM VE DONANIM DİZAYN VE İMPLEMENTASYONU

ÖZ

Kullanıcı arayüzü grafiklerini(GUI) kullanarak ve bir infrared uzaktan kumanda vasıtasıyla televizyon elektronik kartıyla iletişim sağlanabilir. Kart üstündeki register'lara okuma ve yazma yapılarak teknik problemlerin uzaktan teşhisini sağlayabilir, olası çözümleri deneyebilir ve kart üstündeki mikroişlemcinin içindeki yazılımı geliştirebiliriz. Bu tez bu amacı gütmektedir.

GUI katmanı, Tv-set'inin kompleks yazılım yapısının bir dalıdır. Bu tezde gerçekleştirdiğimiz şey, esasında bu yapıya 0,5 kB civarında işletilebilir, binary formatında bir ek yapmaktır. Yazılımın tamamı 128 kB dır ve SDA555 mikroişlemcisinin rom alanında durmaktadır. İmplementasyonunumuz, tezin amacı doğrultusunda yarattığımız bir menüyü ve bunun ana yazılımla uyumlu çalışması için gereken bağlantılarını kapsamaktadır.

Bu tez/proje esas olarak televizyon setleri için kullanılmayı amaçlıyor. Ancak bunu bir kısıtlama olarak algılamamak gerekir. Endüstride, gömülü yazılım geliştirme mühendisi olarak, uzmanlığımız TV setleri olduğu için, bu alanda kullanılmasını daha büyük bir özgüvenle tavsiye edebiliriz. Bu bağlamda, bu tezin, yüksek hacimle üretimi yapılan herhangi bir tüketici elektroniği ürününde kullanılabileceğini düşünüyoruz.

Anahtar Kelimeler : Grafik Kullanıcı Arayüzü , Uzaktan Teşhis , TV Seti

CONTENTS

Page

CHAPTER ONE
INTRODUCTION

1.1 Introduction

Consumer electronics production is a though business. It has a very low profit margin, around 3%-10% depending on the type of product and the demand to it. The profitability of the product is closely affected by the technical competitiveness of the product to the other models of other brands in the market.

The more you add technical options and possibilities for the user of the product, the more price you can get in marketing the electonic product.

But, as the technical complexity of the product grows, the number of consumer complaints are increased. Because, in this market, products are produced according to orders of consumers. If an electronic device producer company can make a deal with it's consumers and sells a new brand product, sales department requests from R&D department, that the product has a project, a technical group to work on it and realize it in a specific, certain time interval. The product should be ready in a few months. This time interval could be from 2 months up to 6 months in today's conditions.

R&D development interval is a quite complex process. Many people can take part at different steps of the schedule. Many problems are encountered by the software and hardware development guys. Most of them are solved before the production date. But, sometimes time is not enough and not all of the problems can get solved in that restricted time.

At that moment, managers decide which problems to solve and which problems to postpone to other versions of productions of the electronic good/project. This is done by

giving priority flags to all of the problems. Some problems are “must” problems, meaning that, they should be solved before the first lot of product shipped to the customer. These must problems are nightmares of the developers, because sometimes guys should work overnight to overcome these problems. But, if you think that the solving process of these problems are the most annoying moments of the engineer.You might be wrong.

The most annoying moment are experienced by the bugs/problems found by the customer after the products are shipped. This situation is experienced many times thorugh the year. Most conventional way to solve these situations is to send one or two engineers to the customer together with the customer contact person to the region where the problem is seen. And this costs thousands of dollars and time to producer company. Because, sometimes engineers have the visa for the country, and taking visa adds about one week to the solution time of the problem. And this means not only time but another thousands of dollars, since the production of the product is suspended.

We are slowly coming to the motivation of thesis, the aim and use of it. Since such situations are easily encountered many times, this thesis has been created for the solution of technical problems found by the consumer. Flying to the customer should not be necessary. With the help of this thesis taking technical feedback form the customer is possible, regardless of his/her being a technical person. Problems can be solved by using the product itself. The customer can be guided to use the product in a different way, something like a small testing device and more than that, something like a simulator, as described in Figure 1.1.

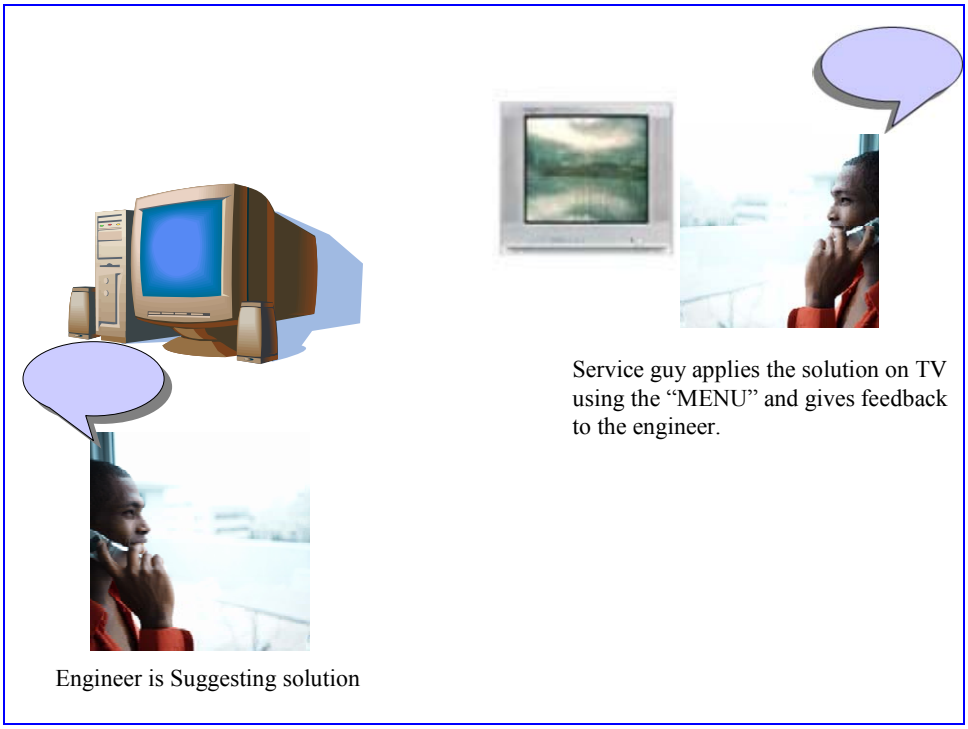

Figure 1.1 The engineer and the technical/nontechnical guy on customer side are arguing upon the problem and the engineer guides the guy, so that the problem diagnostic is done.

We aimed to write a software routine, hidden in the software of the product, to help us find out what the reason of the encountered bug is and even how to solve it. You might find that the routine is easy to implement. But it is working together with the original software of the tv set and it has to fit to a restirced area in the rom of microcontroller. Besides, such kind of software has not been done before and will probably find various uses in the electronics market by many companies for many other reasons.

The goal of this software peace is to drive the chips, working together with I2C interface, by the service-guy of the electronic product, by simply using the remote controller of the device,Figure 1.2, and a grapic user interface, written by the software enginneer as a service routine and accessed by some hidden code.

While realizing this project, SDA555 microcontroller chips has been used. It has an internal memory like 128 KB. But this project should not necesssarily be applied at this specific chip and at its' internal memory version. It can also be succesfully be applied at

external memory version of SDA555 (Micronas chip, formerly known as Infineon) and it can also be applied at any chip, driven by embedded software, running under an OS or not. (such as ST or Philips concepts.)

We have used an existing TV set software for the base of our implementation of remote diagnostic software. Using Keil Compiler and Linker, we did add our software to the structure of existing software. The new software has been uploaded to the rom area of SDA555 microcontroller. Software has been run and seen that there is a new menu available with the TV set. Using this menu we are able to access registers of all IC's one by one. With this access we can easily investigate the state of the board and the software, without using probes, oscilloscopes etc.

The uploaded software(flash code) can also be updated by RS232 protocol (can alternatively be named as UART) at some versions of microcontroller IC's such as ST's 5518. Upload process is done by the hyperterminal property of any PC. The well-known 5518s are used at satellite receivers, which is another good sample product, which can be used at the scope of the project.

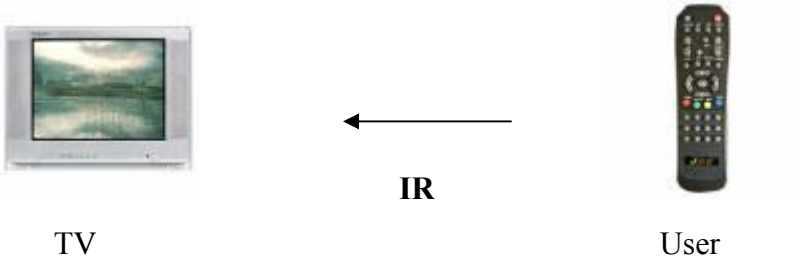

Figure 1.2 Simple communication of the user with the TV-Set.

CHAPTER TWO
DESIGN BASICS

2.1 Introduction

In the design of our hardware and software platform, we took the benefit of the already existing hardware and software base of the tv set.

The main part of the platform is the I2C bus. Let us briefly go over what I2C is, how it is used and the sample code to handle and manage it .

2.1.1 Abbreviation of Inter-IC

Philips Semiconductors is the developer of the I2C bus. In early 80's, Philips aimed to establish a communication line between the microcontroller and other ICs on the electronic board of a TV Set.

Integrated circuits on the electronic board of an embedded system are connected to each other and the microcontroller by the parallel address and data bus of the microcontroller. This type of wiring results in a huge mess of connections. On the PCB(Printed Circuit Board), there might not be such a wide area sometimes. Even if there is, this mess might affect the EMC tests in negative way. This situation is not acceptable for mass production companies, such as TV-set, VCR and set top box producers.

Philips Labs in Eindhoven (The Netherlands) started a research to overcome these problems resulted in a 2-wire communication bus called the I2C bus. I2C is an abbreviation

of Inter-IC bus. Its name also explains its purpose: to provide a communication link between Integrated Circuits.

The I2C bus is a standard in the electronics industry and has been accepted by many major chip producers, such as Micronas, ST, Intel. I2C is mainly used in audio and video products.

2.1.2 The I2C Bus Communication Protocol

The I2C bus has a total of 3 wires. One wire is a ground line and the other wires are active wires. The active wires, called SDA and SCL, are both bi-directional. SDA is the Serial DAta line, and SCL is the Serial CLock line.

Every device connected to the bus has its own unique address, no matter whether it is an MCU, LCD driver, rom, or ram. Each of these chips can act as a receiver and/or transmitter, depending on the functionality. Obviously, an LCD driver is only a receiver, while a memory or I/O chip can be both transmitter and receiver.

The I2C bus is a multi-master bus. This means that more than one IC is authorized to initiate a data transfer. The I2C protocol specification says that the IC that starts a data transfer on the bus is called the Bus Master. Consequently, at that time, all the other ICs are regarded to be Bus Slaves.

Since bus masters are generally microcontrollers, let's take a look at a general 'inter-IC communication' on the bus. Lets consider the following setup in Figure 2.1 and observe how the MCU sends data to one of its slaves .

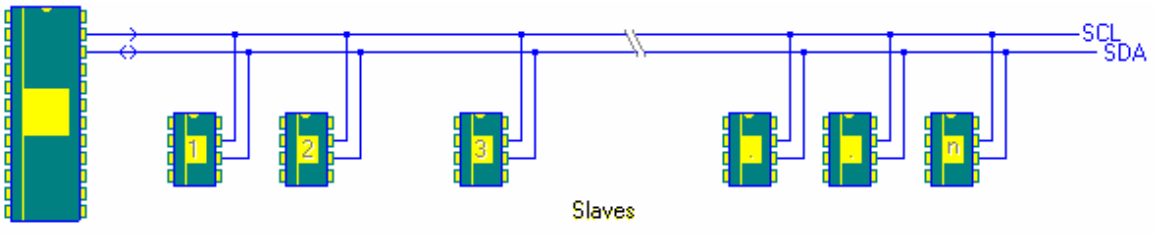

Figure 2.1 Master IC on the left connected to slave IC's, with I2C data and clock lines, shown as in figure above.

First, the MCU sends a START condition. This means 'Attention' signal to all of the connected ICs. All ICs on the bus will be ready for incoming data.

Then the MCU sends the ADRESS of the device it wants to communicate with, together with an indication whether the operation is a Read or Write operation. After receiving the address, all IC's will compare it with their own address. If it doesn't match, they have to wait until the bus is released by the stop condition, before accessing to the bus. If the address matches, however, the responding IC will send a response called the ACKNOWLEDGE signal.

As the MCU receives the acknowledge signal, it will start transmitting DATA. After transmitting the DATA, the MCU will send the STOP condition. This is a signal that the bus has been released and that all other connected ICs are again available for another transmission to start any moment.

As we have seen simply I2C bus wiring does manage the data transfer between compatible IC's. In this thesis, the circuitry does use the 8-bit controller SDA555 to handle I2C bus data management as a master and many slaves like audio and video IC's, which have SDA and SCL pinnings.

CHAPTER THREE
HARDWARE COMPONENTS & DESIGN

3.1 Introduction

Though we did not need to make a detailed hardware design in this Project, it is obvious that we need some definite hardware setup and connections for realizing hardware components run compatible with each other and the update of the flash software be done successfully.

In our project, the block diagram looks like the Figure 3.1. Mainly speaking, most of the modules are microchips connected by I2C protocol, which makes the reachable area of this thesis wider in the concept of product.

In Figure 3.1, you can see connections are made by arrows in one or both directions. These connections are multi-purpose. They are either video/audio carrying lines (Information is carried both analog and digital, but as the projects in the market proceed in time, connections get more and more digital for a faster and efficient process of data) or I2C connections or come other synchronization and voltage connections. But, most of these arrows stand for I2C connection.

Other than the I2C connections, you can find the basic hardware design steps of this project to have a feeling of how a semi analog and semi digital TV signal processing concept looks like.

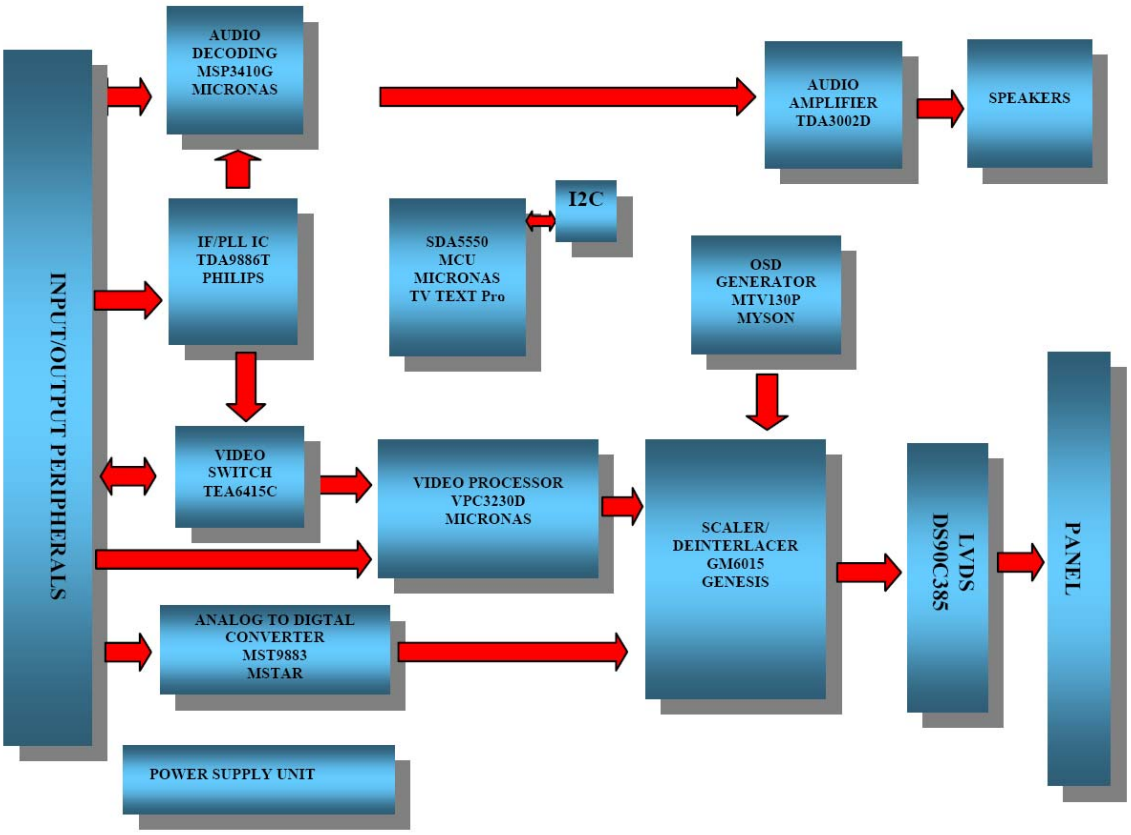

Figure 3.1 Master IC (SDA5550)in the center connected to slave IC's, with I2C data and clock lines, shown as in figure above.

3.2 Hardware Blocks

Hardware blocks can simply be introduced by dividing them into two parts, one of which is called front-end and the next is called as back-end. Meaning of these terms are, that input signals are taken and processed at front-end, and the signals are processed and output at the back-end part.

- **Front End**: Tuner IF, Video Matrix Switch, Video Processor, TV Text Pro, Audio Processor, Audio Amplifier, Analog to Digital Converter

- **Back End**: Scaler/Deinterlacer, On Screen Display, LVDS transmitter

3.2.1 Front End

As mentioned earlier, in this project, the mainboard consists of two major blocks. The first block is Analog front-end. This block performs demodulation of RF, CVBS, Audio, RGB, SVHS, YPBPR input selection and processing. TV tuner is an asymmetrical IF output type and is PLL controlled. For multi-standard reception, the video and sound filters are switch able SAW filters and controlled by SAW_SW output from TDA9886. After the SAW filter block, IF signal is applied to TDA9886 IF inputs (VIFIN[1,2] and SIF[1,2]). TDA9886 feeds the VPC3230D video processor with base band composite video signal. TDA9886 is multi-standard (PAL, SECAM and NTSC) vision, sound IF signal PLL demodulator for positive and negative modulation including sound AM and FM processing. These processes are summarized in Figure 3.2.

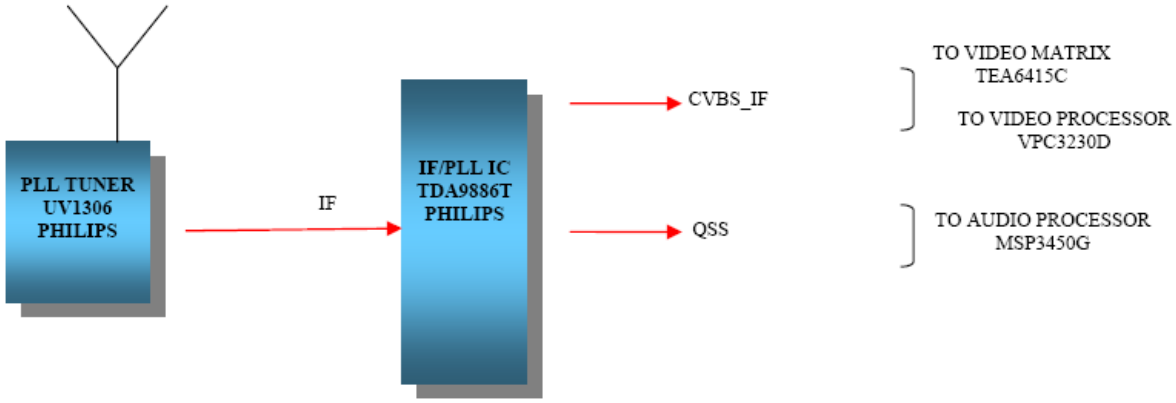

Figure 3.2 Tuner and IF Processor.

The video matrix switch TEA6415C, Figure 3.3, performs input selection. The video matrix switch TEA6415C switches 8 video input sources on 6 outputs. Matrix switch handles with video input signals: SC1_CVBS, SC2_CVBS, CVBS, S-VIDEO and output signals: CVBS_OUT1, CVBS_OUT2 and YC. VPC3230D, Figure 3.4, handles video processing, such as color standard detection and decoding, picture alignment (brightness,

contrast, color etc). It feeds the SDA5550 TV Text Pro with the selected CVBS output. SDA5550 is an 8-bit controller based on an 8051 core, which performs embedded teletext, and TV controller functions. SDA5550 performs teletext decoding and feedback the VPC3230D with analog RGB text and fast blanking signals.

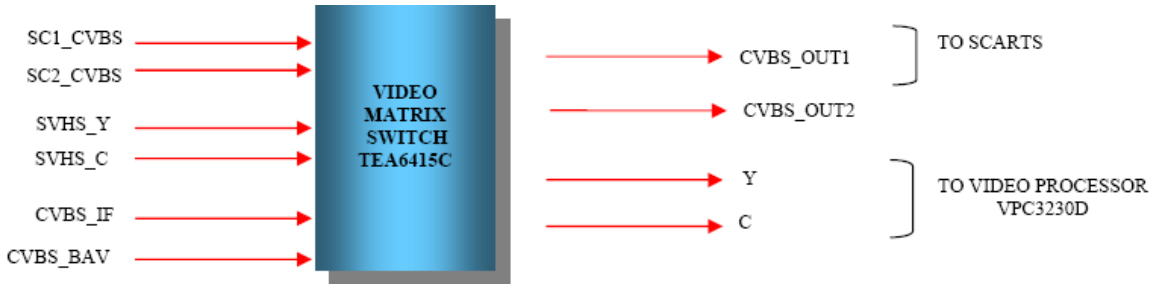

Figure 3.3 Video Matrix

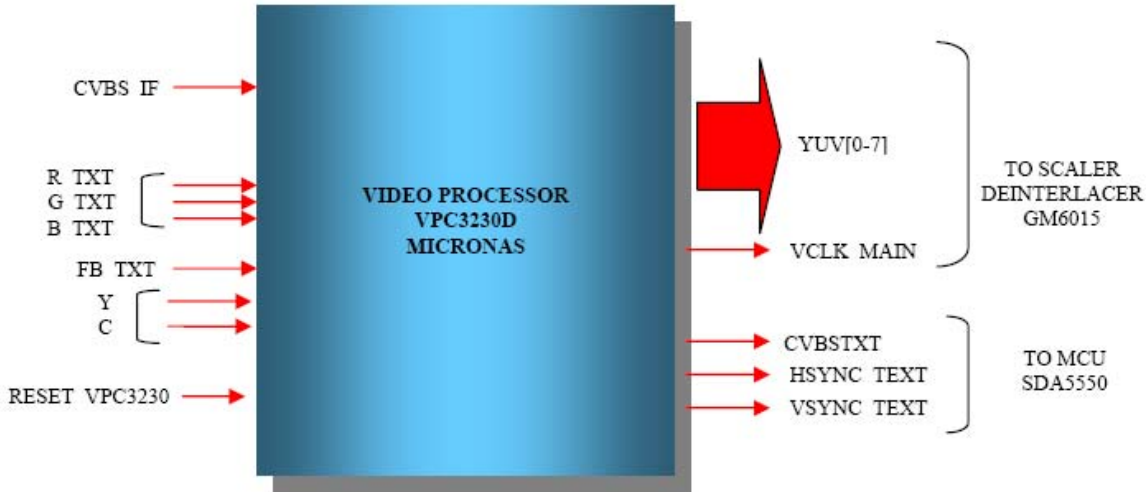

Figure 3.4 Video Processor.

After video processing, VPC3230D transfers the processed video signal to GM6015 scaler chip in BT656 format. YPbPr and VGA inputs are multiplexed before entering the MST9883C by using PI5V330, Figure 3.5, multiplexer. MST9883C analog-to-digital converter performs YPbPr/RGB component video signal analog-to-digital conversion. The Hsync and Vsync signals are directly fed to the ADC, which has both separate Hsync/Vsync pins and sync on green (SOG) pin. The 4:4:4 RGB digital outputs with separate horizontal and vertical syncs and data clock is fed to the GM6015 scaler IC.

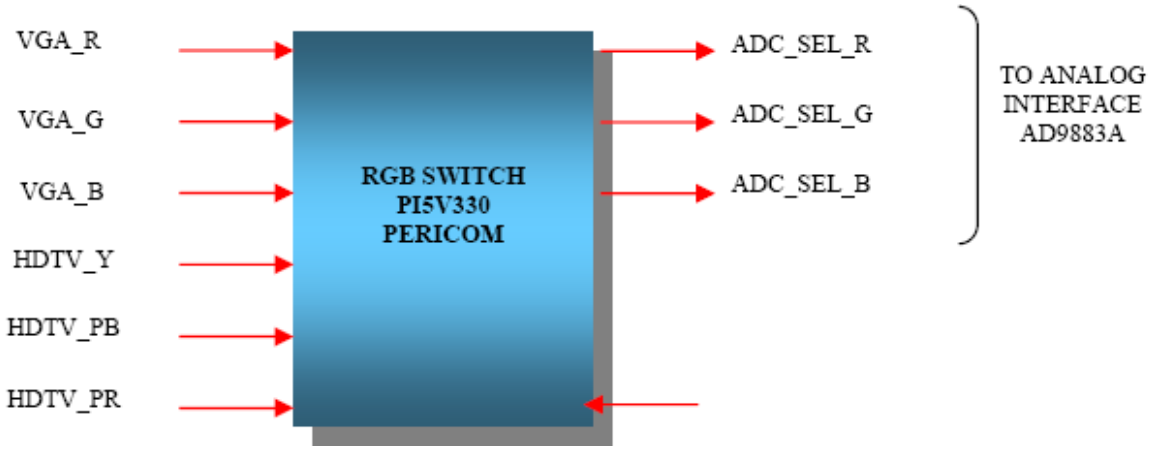

Figure 3.5 RGB Switch.

Sound signal from TDA9886 IF IC in QSS format and line-in right-left sound signals are fed to the MSP3410G audio processor. MSP3410G supports tone control, volume control, AVL, surround effect etc. and supplies amplifier, headphone and CVBS & audio line outputs. There is also an optional audio delay IC to audio outputs. The board employs MAX9714 and TDA1308T to drive speaker and headphone outputs respectively. All these connections can be seen in Figure 3.6.

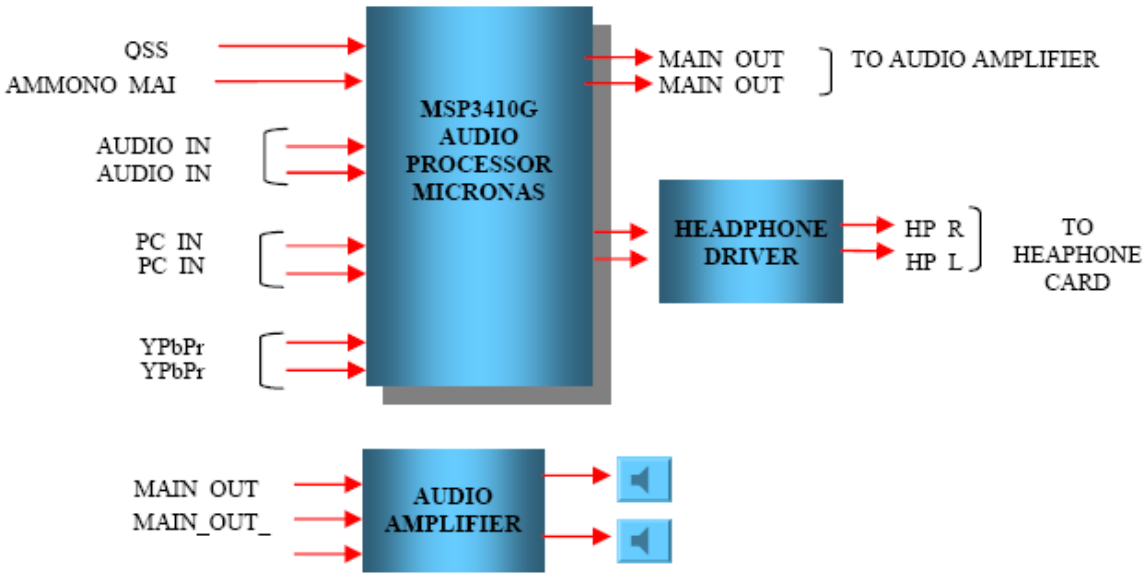

Figure 3.6 Audio Processing.

3.2.2 Back End

The Back End section is handled by GM6015 chip. The highly integrated GM6015 includes dual digital video inputs for MAIN and PIP display, dual bi-level/tri-level sync separators, pixel based motion adaptive deinterlacing, 2:2/3:2 inverse pull down film processing, diagonal processing, arbitrary shrink/zoom scaling on both MAIN and PIP channels. It can handle 480/576i, 480/576p, 720p and 1080i HD, Dual NTSC/PAL/SECAM CVBS and YC, VGA, SVGA, XGA PC graphics. There is also on-screen display (OSD) user interface option. The MTV130 OSD IC feeds GM6015 with RGB OSD signals. RCA YPbPr input and VGA RGB input are switched through PI5V330 and fed to MST9883C. After analog-to-digital conversion, the digitized 4:4:4 RGB signals are fed to the port A input of GM6015 and data in BT.656 format coming from the VPC3230D is fed to port B. The signal level of GM6015 outputs is in TTL format. The LCD panel may be fed either with TTL signals or by adding DS90C385 LVDS transmitter. Connections can be seen in Figure 3.7.

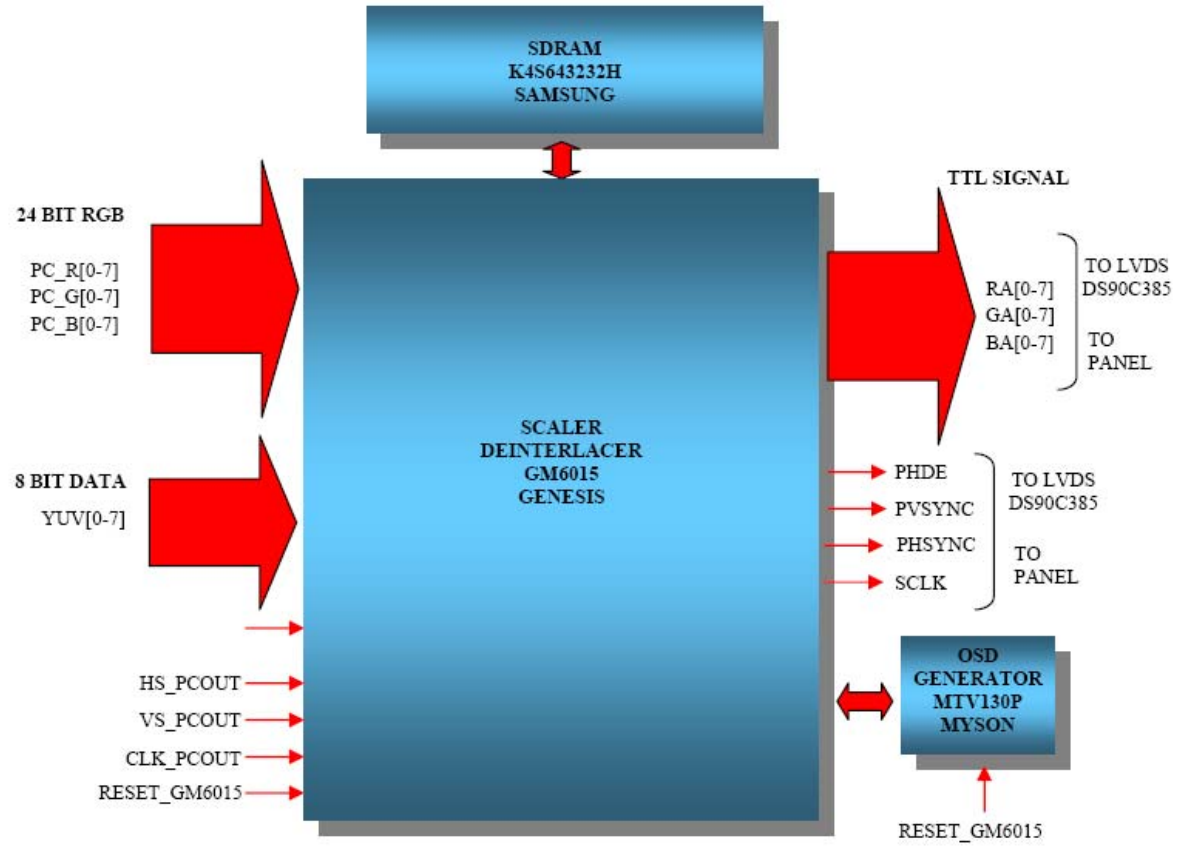

Figure 3.7 Scaler Deinterlacer.

The video output from GM6015 is digital RGB bus format is intended to interface with TTL compatible display devices. As GM6015 does not have integrated LVDS transmitter, TTL control signals from GM6015 are also fed to DS90C385 LVDS IC to produce LVDS output for LVDS compatible LCDs, Figure 3.8.

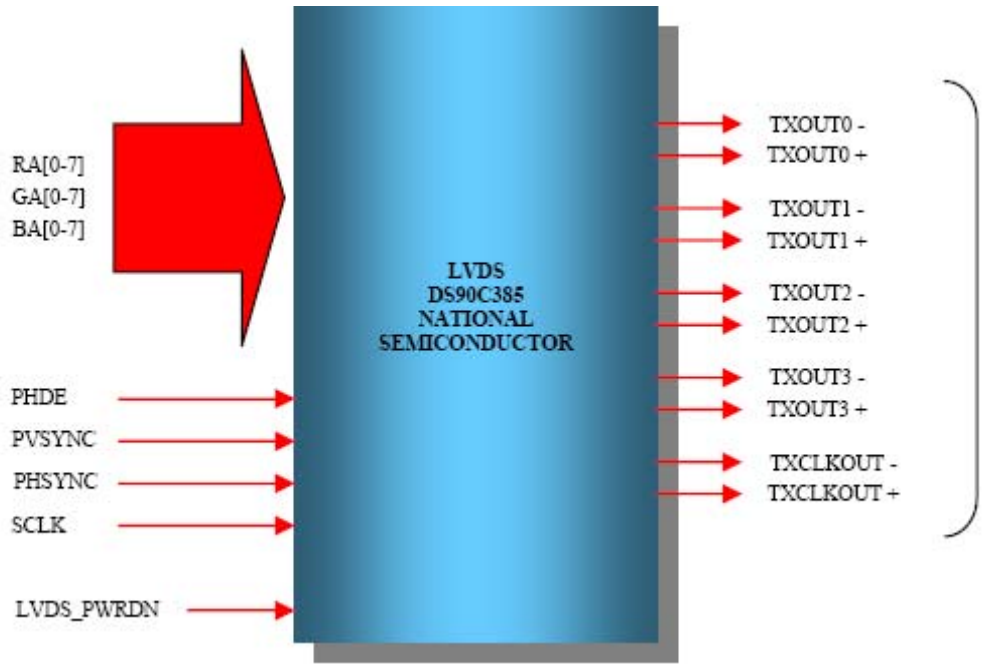

Figure 3.8 LVDS Transmitter.

3.3 Integrated Circuits

Let us briefly explain the IC's used in the project, their aims in the circuitry and what they are capable of.

3.3.1 SDA55XX(SDA5550) – Microcontroller

The SDA55XX is a single chip teletext decoder for decoding World System Teletext data as well as Video Programming System (VPS), Program Delivery Control (PDC), and Wide Screen Signalling (WSS) data used for PAL plus transmissions (Line 23). The device also supports Closed caption acquisition and decoding. The device provides an integrated general-purpose, fully 8051-compatible Microcontroller with television specific hardware features. Microcontroller has been enhanced to provide powerful features such as memory banking, data pointers, and additional interrupts etc.

The on-chip display unit for displaying Level 1.5 teletext data can also be used for customer defined on screen displays. Internal XRAM consists of up to16 Kbytes. Device has an internal ROM of up to 128 KBytes. ROMless versions can access up to 1 MByte of external RAM and ROM. The SDA 55XX supports a wide range of standards including PAL, NTSC and contains a digital slicer for VPS, WSS, PDC, TTX and Closed Caption, an accelerating acquisition hardware module, a display generator for Level 1.5 TTX data and powerful On screen Display capabilities based on parallel attributes, and Pixel oriented characters (DRCS).

The 8-bit Microcontroller runs at 360 ns. cycle time (min.). Controller with dedicated hardware does most of the internal TTX acquisition processing, transfers data to/from external memory interface and receives/ transmits data via I2C-firmware user-interface. The slicer combined with dedicated hardware stores TTX data in a VBI buffer of 1 Kilobyte. The Microcontroller firmware performs all the acquisition tasks (hamming and parity-checks, page search and evaluation of header control bits) once per field. Additionally, the firmware can provide high-end Teletext features like Packet-26-handling, FLOF, TOP and list-pages. The interface to user software is optimized for minimal overhead. SDA 55XX is realized in 0.25 micron technology with 2.5 V supply voltage and 3.3 V I/O (TTL compatible).

3.3.2 TPA30033D2 – Audio Amplifier

3.3.2.1 General description

The TPA3003D2 is a 3-W (per channel) efficient, Class-D audio amplifier for driving bridged-tied stereo speakers. The TPA3003D2 can drive stereo speakers as low as 8 Ω. The high efficiency of the TPA3003D2 eliminates the need for external heatsinks when playing music. Stereo speaker volume is controlled with a dc voltage applied to the volume control terminal offering a range of gain from –40 dB to 36 dB.

3.3.2.2 Features

- 3-W/Ch Into an 8-Ω Load From 12-V Supply
- Efficient, Class-D Operation Eliminates Heatsinks and Reduces Power Supply Requirements
- 32-Step DC Volume Control From −40 dB to 36 dB
- Third Generation Modulation Techniques − Replaces Large LC Filter With Small Low-Cost Ferrite Bead Filter
- Thermal and Short-Circuit Protection

3.3.3 TDA9885/86

3.3.3.1 General description

The TDA9885 is an alignment-free single standard (without positive modulation) vision and sound IF signal PLL. The TDA9886 is an alignment-free multistandard (PAL, SECAM and NTSC) vision and sound IF signal PLL demodulator for positive and negative modulation including sound AM and FM processing. Both devices can be used for TV, VTR, PC and set-top box applications.

3.3.3.2 Features

- 5 V supply voltage
- Gain controlled wide-band Vision Intermediate Frequency (VIF) amplifier (AC-coupled)
- Multistandard true synchronous demodulation with active carrier regeneration (very linear demodulation, good intermodulation figures, reduced harmonics, excellent pulse response)
- Gated phase detector for L/L accent standard
- Fully integrated VIF Voltage Controlled Oscillator (VCO), alignment-free;

frequencies switchable for all negative and positive modulated standards via I2C-bus

- Digital acquisition help, VIF frequencies of 33.4, 33.9, 38.0, 38.9, 45.75 and 58.75 MHz
- 4 MHz reference frequency input [signal from Phase-Locked Loop (PLL) tuning system] or operating as crystal oscillator
- VIF Automatic Gain Control (AGC) detector for gain control, operating as peak sync detector for negative modulated signals and as a peak white detector for positive modulated signals
- Precise fully digital Automatic Frequency Control (AFC) detector with 4-bit digital-to-analog converter; AFC bits via I2C -bus readable
- TakeOver Point (TOP) adjustable via I2C-bus or alternatively with potentiometer
- Fully integrated sound carrier trap for 4.5, 5.5, 6.0 and 6.5 MHz, controlled by FM-PLL oscillator
- Sound IF (SIF) input for single reference Quasi Split Sound (QSS) mode (PLL controlled)
- SIF AGC for gain controlled SIF amplifier; single reference QSS mixer able to operate in high performance single reference QSS mode and in intercarrier mode, switchable via I2C-bus
- AM demodulator without extra reference circuit
- Alignment-free selective FM-PLL demodulator with high linearity and low noise
- I2C-bus control for all functions
- I2C-bus transceiver with pin programmable Module Address (MAD).

3.3.4 TDA1308

3.3.4.1 General description

The TDA1308 is an integrated class AB stereo headphone driver contained in an SO8 or a DIP8 plastic package. The device is fabricated in a 1 mm CMOS process and has been primarily developed for portable digital audio applications.

3.3.4.2 Features

- Wide temperature range
- No switch ON/OFF clicks
- Excellent power supply ripple rejection
- Low power consumption
- Short-circuit resistant
- High performance
- High signal-to-noise ratio
- High slew rate
- Low distortion
- Large output voltage swing.

3.3.5 PI5V330

3.3.5.1 General description

Pericom Semiconductor's PI5V series of mixed signal video circuits are produced in the Company's advanced CMOS low-power technology, achieving industry leading performance. The PI5V330 is a true bidirectional Quad 2-channel multiplexer/demultiplexer that is recommended for both RGB and composite video

switching applications. The VideoSwitch™ can be driven from a current output RAMDAC or voltage output composite video source. Low ON-resistance and wide bandwidth make it ideal for video and other applications. Also this device has exceptionally high current capability which is far greater than most analog switches offered today. A single 5V supply is all that is required for operation. The PI5V330 offers a high-performance, low-cost solution to switch between video sources. The application section describes the PI5V330 replacing the HC4053 multiplier and buffer/amplifier.

3.3.5.2 Features

- High-performance, low-cost solution to switch between video sources
- Wide bandwidth: 200 MHz
- Low ON-resistance: 3W
- Low crosstalk at 10 MHz: –58 dB
- Ultra-low quiescent power (0.1 μA typical)
- Single supply operation: +5.0V
- Fast switching: 10 ns
- High-current output: 100 mA

3.3.6 GM6015

3.3.6.1 General description

The Genesis Microchip 6015RD1 LCD TV reference board is a complete display processor for LCD, PDP and LCOS based televisions. The reference board demonstrates the processing capabilities of the Genesis Microchip gm6015 television controller IC. The gm6015 IC is a full-featured, dual-channel video processor with Genesis industry leading Crystal Ciema Plus™ video scan conversion. The 6015RD1 board inputs analog YPbPr/RGB, NTSC/PAL/SECAM CVBS/YC, UHF/VHF and outputs digital RGB to an

XGA LCD panel. A convenient on-screen display system provides easy control of the board's processing capabilities.

The design kit is complete with hardware and software. Software includes G-Probe debug software, GWizard register calculator and G-TV application source code. The 6015RD1 is a related reference board that outputs analog YpbPr/RGB.

3.3.6.2 Features

- Dual channel, gm6015 based LCD TV system
- Industry leading Crystal Cinema Plus video scan conversion
- Inputs:

  i. Component analog YPbPr/RGB

  ii. 480/576I, 480/576P, 720P and 1080I HD

  iii. Dual NTSC/PAL/SECAM CVBS and YC

  iv. VGA, SVGA, XGA PC graphics

  v. Separate, composite or sync on Y/G

  vi. UHF/VHF RF (NTSC)
- Default output with XGA LCD interface PCB:

  i. Component analog YpbPr/RGB
- Other outputs:

  ii. 8/16/20/24-bit 4:2:2/4:4:4 digital YCbCr/RGB

  iii. 480/576I, 480/576P, 720P and 1080I HD

  iv. VGA, SVGA, XGA PC graphics

  v. Separate, composite or sync on Y/G
- On-screen display (OSD) user interface with automated self running demonstration
- Small form factor PCB

3.3.7 AD9883A

3.3.7.1 General description

The AD9883A is a complete 8-bit, 140 MSPS monolithic analog interface optimized for capturing RGB graphics signals from personal computers and workstations. Its 140 MSPS encode rate capability and full power analog bandwidth of 300 MHz supports resolutions up to SXGA (1280 _ 1024 at 75 Hz). The AD9883A includes a 140 MHz triple ADC with internal 1.25 V reference, a PLL, and programmable gain, offset, and clamp control. The user provides only a 3.3 V power supply, analog input, and Hsync and COAST signals. Three-state CMOS outputs may be powered from 2.5 V to 3.3 V.

The AD9883A's on-chip PLL generates a pixel clock from the Hsync input. Pixel clock output frequencies range from 12 MHz to 140 MHz. PLL clock jitter is 500 ps p-p typical at 140 MSPS. When the COAST signal is presented, the PLL maintains its output frequency in the absence of Hsync. A sampling phase adjustment is provided. Data, Hsync, and clock output phase relationships are maintained.

The AD9883A also offers full sync processing for composite sync and sync-on-green applications. A clamp signal is generated internally or may be provided by the user through the CLAMP input pin. This interface is fully programmable via a 2-wire serial interface. Fabricated in an advanced CMOS process, the AD9883A is provided in a space-saving 80-lead LQFP surface-mount plastic package and is specified over the 0C to 70C temperature range.

3.3.7.2 Features

- 140 MSPS Maximum Conversion Rate
- 300 MHz Analog Bandwidth
- 0.5 V to 1.0 V Analog Input Range
- 500 ps p-p PLL Clock Jitter at 110 MSPS

- 3.3 V Power Supply
- Full Sync Processing
- Sync Detect for “ Plugging ”
- Midscale Clamping
- Power-Down Mode
- Low Power:500 mW Typical
- 4:2:2 Output Format Mode

3.3.8 MC141585

3.3.8.1 General description

This is a high performance HCMOS device designed to interface with a micro controller unit to allow colored symbols or characters to be displayed onto a LCD monitor. Because of the large number of fonts, 512 fonts including 496 standard fonts and 16 multi-color fonts, LMOSD2-16 is suitable to be adopted for the multi-language monitor application especially. It minimizes the MCU’s burden through its built-in RAM. By storing a full screen of data and control information, this device has a capability to carry out ‘screenrefresh’ without any MCU supervision. Programmable hatch pattern generator is added for individual pixel inspection.

Since there is no clearance between characters, special graphics oriented characters can be generated by combining two or more character blocks. The full OSD menu is formed of 15 rows x 30 columns which can by freely positioned on anywhere of the monitor screen by changing vertical or horizontal delay.

Special functions such as character background color, blinking, bordering or shadowing, four-level windows with programmable size, row double height and double width,

programmable vertical height of character and row-to-row spacing, and full-screen erasing and Fade-In/Fade-Out are also incorporated. There are 8 color selections for any individual character display with row intensity attribute and window intensity attribute to expand the color mixture on OSD menu.

3.3.8.2 Features

- Totally 512 Fonts Including 496 Standard Fonts and 16 Multi-Color Fonts.
- 10x18 or 12x18 Font Matrix Selection
- Maximum Pixel CLK of 80MHz
- Maximum input resolution of 1580 dots/line (PIXin/HSYNC ratio)
- Wide Operating Frequency: max. 150KHz for Monitor
- Fully Programmable Character Array of 15 Rows by 30 Columns
- 8-Color Selection for Characters with Color Intensity Attribute on Each Row
- 7-Color Selection for Characters background
- True 16-Color Selection for Windows
- Shadowing on Windows with Programmable Shadow Width/Height/Color
- Fancy Fade-In/Fade-Out Effects
- Programmable Height of Character to Meet Multi-Sync Requirement
- Row To Row Spacing Control to Avoid Expansion Distortion
- Four Programmable Windows with Overlapping Capability
- Character Bordering or Shadowing
- Character/Symbol Blinking Function
- Programmable Vertical and Horizontal Positioning for Display Center
- M_BUS (IIC) Interface with Address $7A

3.3.9 MSP34X0G (MSP3400G) –Multistandard Sound Processor Family

3.3.9.1 General description

The MSP 34x0G family of single-chip Multistandard Sound Processors covers the sound processing of all analog TV-Standards worldwide, as well as the NICAM digital sound standards. The full TV sound processing, starting with analog sound IF signal-in, down to processed analog AF-out, is performed on a single chip.

Figure 3.9 shows a simplified functional block diagram of the MSP 34x0G. This new generation of TV sound processing ICs now includes versions for processing the multichannel television sound (MTS) signal conforming to the standard recommended by the Broadcast Television Systems Committee (BTSC). The DBX noise reduction, or alternatively, MICRONAS Noise Reduction (MNR) is performed alignment free. Other processed standards are the Japanese FM-FM multiplex standard (EIA-J) and the FM Stereo Radio standard.

Current ICs have to perform adjustment procedures in order to achieve good stereo separation for BTSC and EIA-J. The MSP 34x0G has optimum stereo performance without any adjustments. All MSP 34x0G versions are pin and software downward compatible to the MSP 34x0D. The MSP 34x0G further simplifies controlling software. Standard selection requires a single I²C transmission only. The MSP 34x0G has built-in automatic functions: The IC is able to detect the actual sound standard automatically (Automatic Standard Detection). Furthermore, pilot levels and identification signals can be evaluated internally with subsequent switching between mono/stereo/bilingual; no I²C interaction is necessary (Automatic Sound Selection).

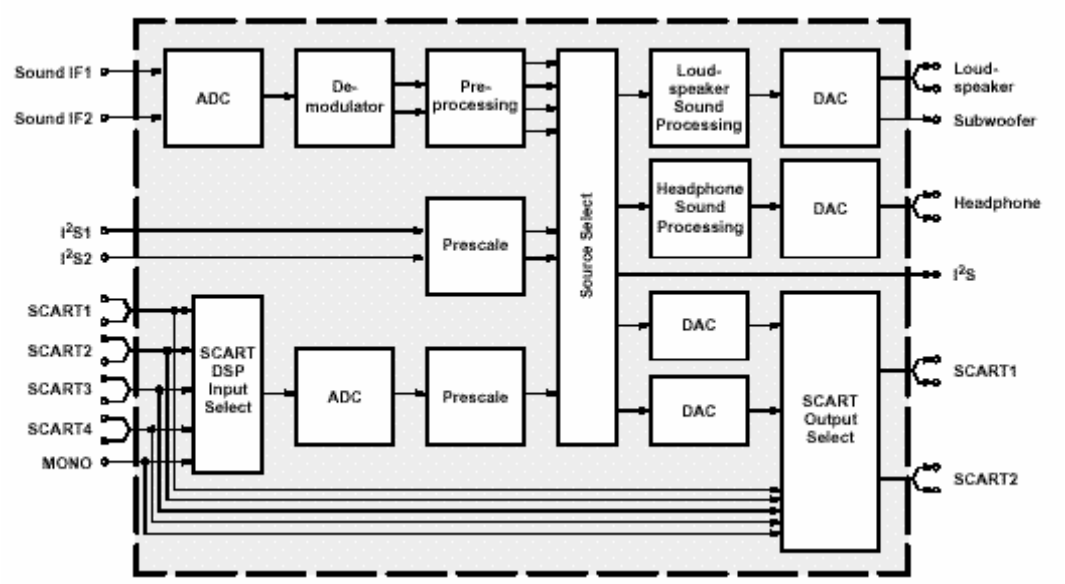

Figure 3.9 The inside modules of MSP3410G and the input and outputs.

3.3.9.1.1 Source Select. I_2S bus interface consists of five pins:

1. I2S_DA_IN1, I2S_DA_IN2: For input, four channels (two channels per line, 2*16 bits) per sampling cycle (32 kHz) are transmitted.
2. I2S_DA_OUT: For output, two channels (2*16 bits) per sampling cycle (32 kHz) are transmitted.
3. I2S_CL: Gives the timing for the transmission of I_2S serial data (1.024 MHz).
4. I2S_WS: The I2S_WS word strobe line defines the left and right sample.

3.3.9.2 Features

- Standard Selection with single I_2C transmission
- Automatic Standard Detection of terrestrial TV standards
- Automatic Sound Selection (mono/stereo/bilingual), new registers MODUS, STATUS
- Two selectable sound IF (SIF) inputs
- Automatic Carrier Mute function

- Interrupt output programmable (indicating status change)
- Loudspeaker / Headphone channel with volume, balance, bass, treble, loudness
- AVC: Automatic Volume Correction
- Subwoofer output, programmable low-pass and complementary high-pass filter 26
- 5-band graphic equalizer for loudspeaker channel
- Spatial effect for loudspeaker channel
- Four Stereo SCART (line) inputs, one Mono input; two Stereo SCART outputs
- Complete SCART in/out switching matrix
- Two I^2S inputs; one I^2S output
- Dolby Pro Logic with DPL 351xA coprocessor
- All analog FM-Stereo A2 and satellite standards; AM-SECAM L standard
- Simultaneous demodulation of (very) high-deviation FM-Mono and NICAM
- Adaptive deemphasis for satellite (Wegener-Panda, acc. to ASTRA specification)
- ASTRA Digital Radio (ADR) together with DRP 3510A
- All NICAM standards
- Korean FM-Stereo A2 standard

3.3.10 VPC3230 – Video Processor

3.3.10.1 General description

The VPC 323xD/324xD is a high quality, single-chip video front-end, which is targeted for 4:3 and 16:9, 50/60 and 100/120 Hz TV sets. It can be combined with other members of the DIGIT3000 IC family (such as DDP 33x0A/B, TPU 3040) and/or it can be used with 3rd-party products.

3.3.10.2 Features

- high-performance adaptive 4H comb filter Y/C separator with adjustable vertical peaking
- multi-standard color decoder PAL/NTSC/SECAM including all substandard
- four CVBS, one S-VHS input, one CVBS output
- two RGB/YCBCR component inputs, one Fast Blank (FB) input
- integrated high-quality A/D converters and associated clamp and AGC circuits

CHAPTER FOUR
SOFTWARE DESIGN

4.1 Introduction

In this chapter we are going to explain the software program we wrote and burned into the SDA5550 microcontroller, which has been used in the television set production. The software will not run under an OS, will be kept in the internal 128 KB flash memory of the Micronas IC SDA5550. Micronas IC is 8-bit 8051 core microcontroller.

During the compilation process Keil compiler has been used. Let us briefly introduce that compiler which is widely used with 8051, 251, USB and 166 microcontroller families.

The Keil C51ANSI C compiler lets you create C programs for the 8051 microcontroller while retaining the efficiency and speed of handoptimized assembly. Extensions incorporated into the C51 compiler give you full access to all CPU resources and derivatives. C51 is fully integrated into the µVision2 IDE that combines Compiler, Assembler, Real-Time OS, project manager, and debugger in a single, intelligent environment as shown in figure 4.1.

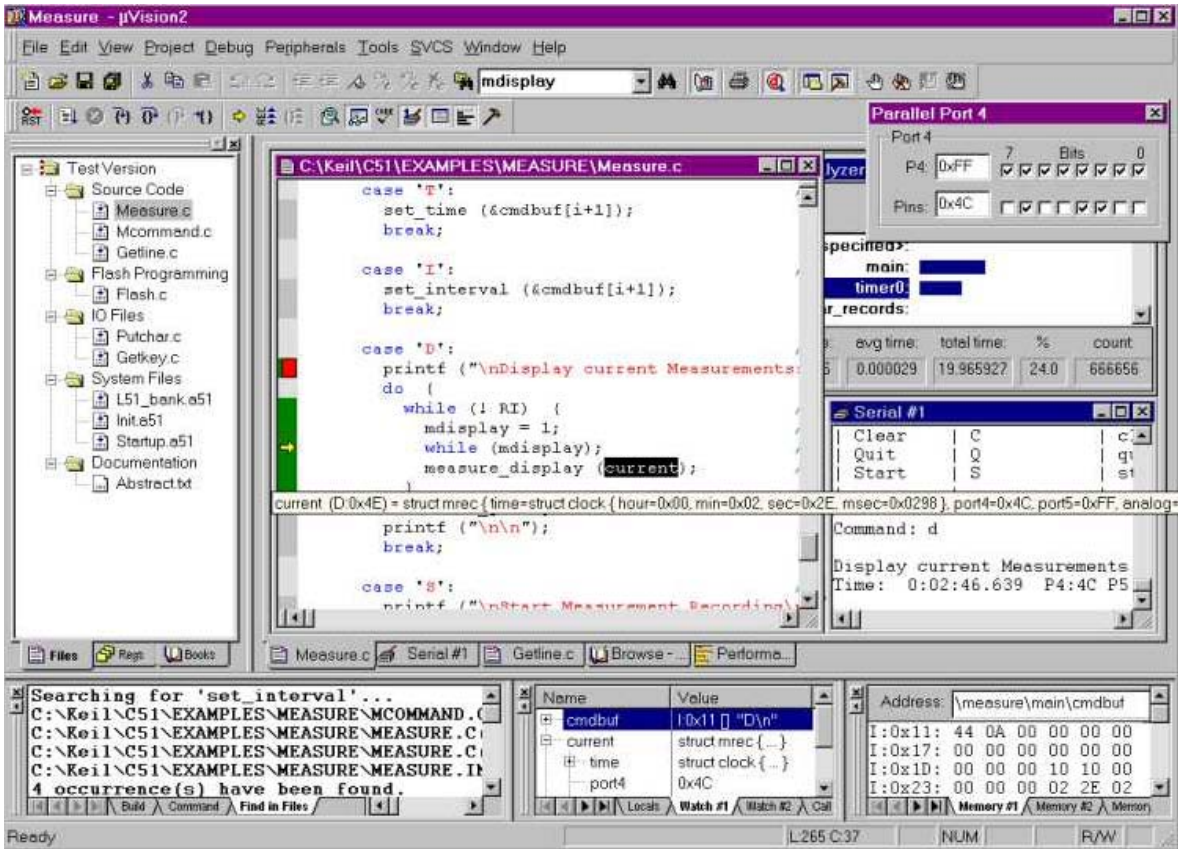

Figure 4.1 The µVision2 IDE is a tool of Keil compiler. In this figure, we see the project management and debugging facilities of KEIL.

.

Memory Selectors

Selector	Address Space
data	128 Bytes directly addressable on-chip RAM, fastest access; default space in SMALL Memory Model.
bdata	16 Bytes bit-addressable RAM; mixed bit and byte access.
idata	256 Bytes indirectly addressable on-chip RAM.
pdata	256 Bytes *paged* external RAM; default in COMPACT Model.
xdata	64 KB external RAM; default in LARGE Memory Model.
code	64 KB program memory.

Figure 4.2 While defining variables to refer to memory addresses, selectors are used.

Some sample declarations of variables as memory selectors are as follows:

- unsigned char data cProgramme, cProgrammePointer;

- unsigned int data iPointerAddr;
- unsigned char data cProgrammeBuffer[4] = {0x1,0,0,0};
- unsigned char data cNameBuffer[2] = {CHR_MINUS,CHR_MINUS};

These declarations are needed in the design of software menus. Variables should be chosen from memory selectors as addressed in Figure 4.2.

CHAPTER FIVE
IMPLEMENTATION AND RESULTS

5.1 Introduction

In principle, special menu implementations should be out of reach of the end-user. End-user means the owner of the product.

Since the problem can be observed at any sample of the product, special menu should be avaialable in every sample's software. To prevent any user from reaching these menus, special chipher should be entered to for enterance.

We have chosen a special number as the chipher, such as highschool number. While the "Main menu" is being displayed on the screen, technical guy should enter digits "1", "6", "5", "9" recpectively by pressing the digit button on RC (remote controller). Let us investigate how this procedure is implemented by the Checkpasswords() procedure:

```
unsigned char code acEntryServiceSequence[] =
{DIGIT1,DIGIT6,DIGIT5,DIGIT9};
/*************************************************************
*
*    Procedure name:    CheckPasswords
*
*    Function:          Checks incoming ir-commands,
*                       if they fit to password sequence
*
*    Input-parameters:  iIrCommand: last received ir-command
*
*    Output-parameters: --
*
*    Return-value:      0: no valid password
```

```
*                       1: service mode entered
*
*    Global Variables:  gcPasswordServiceIndex
*
*************************************************************/

unsigned char CheckPasswords(unsigned int iIrCommand)
{

  unsigned char cPwdOk = 0;

  if(  (gstMenu.cId == MENU_MAIN)
  {
      if(iIrCommand == DIGIT4)
        gcPasswordServiceIndex = 0;
      if(acEntryServiceSequence[gcPasswordServiceIndex]
        == iIrCommand)
      {
        if(++gcPasswordServiceIndex == 4)
        {
            MenuOff();
            MenuOn(MENU_SERVICE_SELECT);
            gcPasswordServiceIndex = 0;
            return 1;
        }
      }
      if(iIrCommand == DIGIT3)
        gcPasswordServiceIndex = 0;

  }
```

```
  return 0;
}
```

In that circumstance, the menu shown below, Figure 5.1, will be displayed on the screen. You can see that there are 5 active lines in the menu. By pressing "▲/▼" buttons on RC user can scroll between those lines (I2CAddr, Regaddr, Value, Write, Read).

By pressing the "◄/►" buttons on RC, user can either realize "read" and "write" function, or address and data values can be entered on the first, second and third lines.

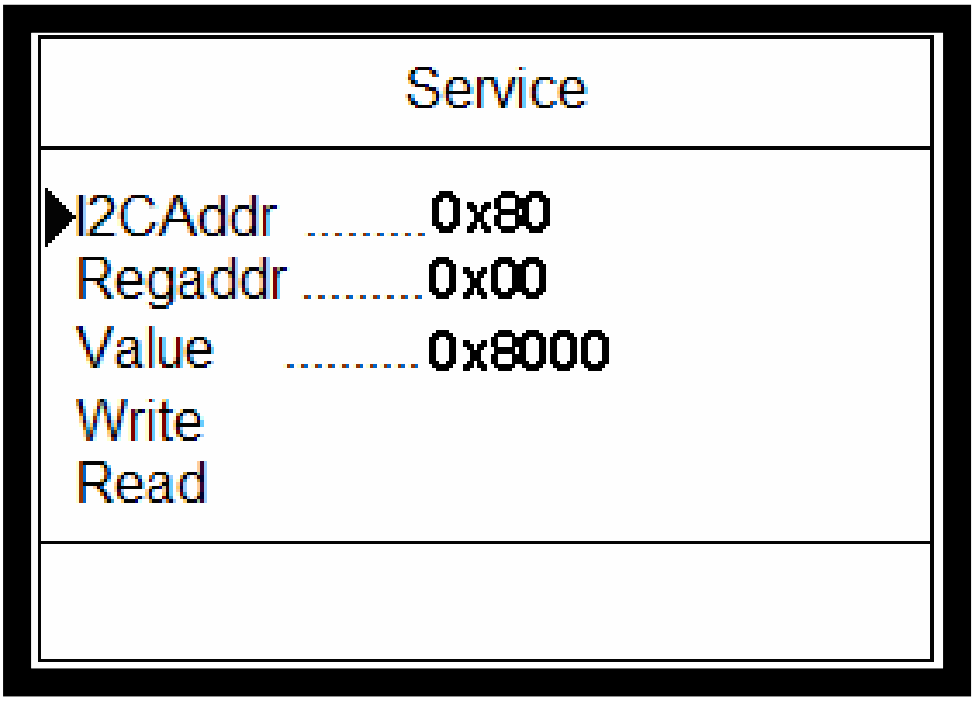

Figure 5.1 Service Menu

Entering the values as in Figure 5.1 and choosing the "Write" line, is equal to calling the MspSwReset funciton be called, as implemented below. This sample is an example of how we can directly access the registers of MSP3410G. With the help of this application, we will be able to make a software reset to MSP3410G IC, so that default settings will be

uploaded to registers from the beginning. This way, we will check if our software settings are correct or not.

```
// Msp device address
#define MSPADDRESS   0x80        // MSP 3410D Device address

// Subaddresses of MSP Read/Write registers
#define CTRLADDR     0x00        // Control register adress
(software reset)

/************************************************************
*
*    Procedure name:    MspSwReset
*
*    Function:          resets MSP 3410 D by software
*
*    Input-parameters:  --
*
*    Output-parameters: --
*
*    Return-value:      --
*
*    Global Variables:  --
*
************************************************************/

void MspSwReset()
{
  unsigned int iMspCtrlData;

  iMspCtrlData = 0x8000;        // set 'reset bit'
```

```
  I2CWrite(MSPADDRESS,CTRLADDR,(unsigned
          char *)(&iMspCtrlData),2); // reset msp statically
  iMspCtrlData = 0x0000;        // clear 'reset bit'
  I2CWrite(MSPADDRESS,CTRLADDR,(unsigned
          char*)(&iMspCtrlData),2); // clear reset
}
```

We can now look at the write and read funcitons. Following functions ReadRegister() and SetRegister() are used to read from and write to the registers in any IC on the board of the TV Set.

```
/*************************************************************
*    Procedure name:    ReadRegister
*
*    Function:  Gets value from the current register .
*
*    Input-parameters:  unsigned int
*                       uiICAddr,uiRegisterAddr
*                       range : 01 .. 65535
*    Output-parameters: unsigned int ui
*
*    Return-value:
*
*    Global variables:
*************************************************************/

unsigned int ReadRegister(unsigned int uiI2CAddr,
                          unsigned int uiRegAddr)

{
  unsigned int uiValue;
```

```
  I2CRead(uiI2CAddr, uiRegAddr,& uiValue,2);

  return uiValue,;
}

/************************************************************
*    Procedure name:    SetRegister
*
*    Function:  Sets the current register to entered value.
*
*    Input-parameters: unsigned int I2CAddr,RegisterAddr
*                      and Value
*                      range : 01 .. 65535
*    Output-parameters:
*
*    Return-value:
*
*    Global variables:
************************************************************
**************************/

void SetRegister(unsigned int uiI2CAddr,unsigned int
                 uiRegAddr,unsigned int    uiValue)
{

  I2CWrite(uiI2CAddr, uiRegAddr,&uiValue,2);

}
```

In table 5.1, we see that some of registers of DSP is listed. DSP is a part of the MSP3410G. Here we can easily learn the register addresses and what the values mean when written to those registers.

For example, if we write 0 to the address 0x0002, this means that Bass level will be +20 dB, if we write 5, then Bass level will be +15 dB.

Table 5.1 DSP Write Registers; If necessary, these registers are readable as well.

DSP Write Register	Address	High/ Low	Adjustable Range, Operational Modes	Reset Mode
Volume loudspeaker channel	0000_{hex}	H	[+12 dB .... –114 dB, MUTE]	MUTE
Volume / Mode loudspeaker channel		L	1/8 dB Steps, Reduce Volume / Tone Control	00_{hex}
Balance loudspeaker channel [L/R]	0001_{hex}	H	[0...100 / 100% and vv][–127 .. 0 / 0 dB and vv]	100% / 100%
Balance Mode loudspeaker		L	[Linear mode / logarithmic mode]	linear mode
Bass loudspeaker channel	0002_{hex}	H	[+20 dB .... –12 dB]	0 dB
Treble loudspeaker channel	0003_{hex}	H	[+15 dB .... –12 dB]	0 dB
Loudness loudspeaker channel	0004_{hex}	H	[0 dB .... +17 dB]	0 dB
Loudness Filter Characteristic		L	[NORMAL, SUPER_BASS]	NORMAL
Spatial effect strength loudspeaker ch.	0005_{hex}	H	[–100%...OFF...+100%]	OFF
Spatial effect mode/customize		L	[SBE, SBE+PSE]	SBE+PSE
Volume headphone channel	0006_{hex}	H	[+12 dB .... –114 dB, MUTE]	MUTE
Volume / Mode headphone channel		L	1/8 dB Steps, Reduce Volume / Tone Control	00_{hex}
Volume / SCART1 channel	0007_{hex}	H	[00_{hex} ... $7F_{hex}$].[+12 dB .... –114 dB, MUTE]	00_{hex}
Volume / Mode SCART1 channel		L	[Linear mode / logarithmic mode]	linear mode

5.2 Software Blocks and Organization

Let us briefly explain the general structure of the software and the connection to the hardware. At the bottom there is hardware and ports, which connect the software to the hardware via I2C ports of the microcontroller. Then we see that the driver level resides upon the ports and hardware.Drivers drive the registers and ports of ICs. With the menu handler, software connects the GUI(Graphic User Interface) to drivers, so that the menu

design and improvement is isolated from the low level. This both decreases the complexity and muddle of the software design and reduces the codesize. GUI is drawing of menu and teletext acquisition. With the help of GUI, the user can adjust the brightness, contrast etc.

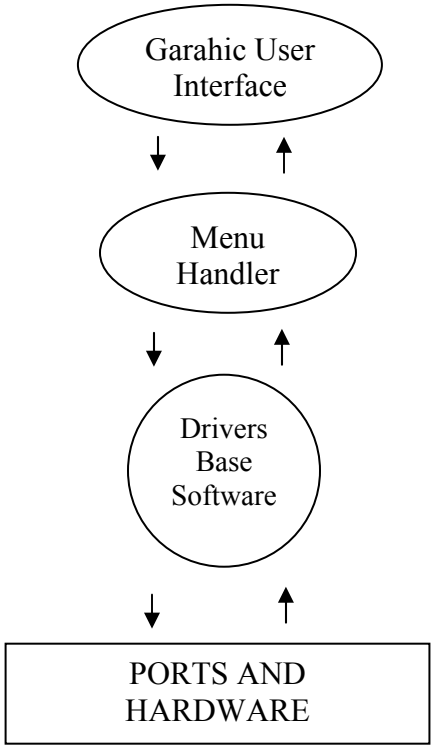

Figure 5.2 Software Layers

CHAPTER SIX
CONCLUSION

6.1 Introduction

This thesis has been completed for those companies, who want to decrease the service costs and make their technical team work more efficiently. We believe that the results of this thesis will have a wide range of use in industry, if applied completely.

This service solution based approach of this thesis is a very big cost reduction process for producer companies. Though we mainly put this thesis on the ground of TV-set and satellite receiver type of electonics goods, readers can imagine the benefits of this thesis for other electronis goods like cell-phones, computers etc.

Since the accessibility range will be the whole world, this way companies will be able to take any kind of feedback from customers. Besides, solution can be seen right at the place where the problem has been observed, and confirmed by customer so that the quality level of devices will be increased and the customer reliance will be improved.

Engineers will not spend their time out of the company, so that other urgent issues regarding production will still be handled and solved with a higher priority . This means higher efficency in the production as well.

6.2 Future Work Applications

The future work might include the reporting of test values of registers of IC's, automatically by an internet or GSM connection to the product. This connection and compatibility might have an increase effect on the cost of the product, both as a hardware

and quaility test requirements base. The unit price of the product will have an effect on the decision of producing goods with such a compatibility or not.

Registers of the TV-set or the other electonic good can be written and read by using internet access and via web-cam, results can be transferred digitally to the R&D department of the regarding company. This way, software and hardware engineers can come together and try to come to a conclusion about the reason of the outcoming problem and the appliable solution.

REFERENCES

Tv and RemoteControl pictures.
Retrieved May 20, 2006 from www.alibaba.com/productsearch/ Tv_Remote_Control.html

Definition of I2C.
Retrieved July 09, 2006 from www.webopedia.com/TERM/I/I2C.html

More about I2C.
Retrieved July 09, 2006 from http://www.esacademy.com

Example for register reading,writing Retrieved August 08, 2006 from
http://www.eserviceinfo.com/downloadsm/14370/MICRONAS_MSP34x0D.html

Software design environment. Retrieved August 12, 2006 from
http://www.keil.com/product/brochures.asp

APPENDIX A

APPLICATION SOURCE CODE

A.1 I2C_bus.c

```
/************************************************************
*
*
*    Module name:           File: I2c_bus.c
*
*    Function:              I2C management procedures source
*                           file
*
*    Revision number:       Revision: 1.1
*
*    Last modification:     Date: 11/5/2006
*
*    Last modification by:  Author: Serdar
*
*
************************************************************/

#define extern
#include <i2cbus.h>
#undef extern


#define SDAHigh      { pSDA = 1; ACC=ACC*B;}
#define SDALow       { pSDA = 0; ACC=ACC*B;}
```

```
#define SCLHigh      { cStretchTime = MAX_CLOCKSTRETCHING;\
                       pSCL = 1; ACC=ACC*B; ACC=ACC*B;
ACC=ACC*B; ACC=ACC*B; NOP; NOP;\
                       while((pSCL == 0) && (--cStretchTime >
0));}
#define SCLLow       { pSCL = 0; ACC=ACC*B; ACC=ACC*B;
ACC=ACC*B; ACC=ACC*B;}
#define SDAIn(cMask) { cMask = pSDA;}


static unsigned char cStretchTime;

/************************************************************
*
*
*    Procedure name:    I2CSetEnable
*
*    Function:          Enables or disables the transmission
*
*    Input-parameters:  cEnable:    0 : disabled
*                                   1 : enabled
*
*    Output-parameters: --
*
*    Return-value:      --
*
*    Global variables:  gcBusEnable
*
*
************************************************************/
```

```
void I2CSetEnable(unsigned char cEnable)
{
   if(cEnable)
      gcBusEnable = 1;
   else
      gcBusEnable = 0;
}


/*************************************************************
*
*
*    Procedure name:    I2CGetEnable
*
*    Function:          Returns the enable/disable status of
*                       the I2C bus
*
*    Input-parameters:  --
*    Output-parameters: --
*
*    Return-value:              0 : enable
*                               1 : disable
*
*    Global variables:  --
*
*
*************************************************************/

unsigned char I2CGetEnable()
{
   if(gcBusEnable)
```

```
      return 0;
   else
      return 1;
}


/*************************************************************
*
*
*    Procedure name:    I2CMasterStop
*
*    Function:          Generates a stop condition
*
*    Input-parameters:  --
*
*    Output-parameters: --
*
*    Return-value:      --
*
*    Global variables:  --
*
*************************************************************
**************************/

void I2CMasterStop()
{
   SCLLow
   SDALow
   SDALow      // timing!
   SCLHigh
   SDAHigh
```

```
}


/************************************************************
*
*
*    Procedure name:    I2CMasterInit
*
*    Function:          Initialises the I2C Bus software
*                       interface
*
*    Input-parameters:  --
*
*    Output-parameters: --
*
*    Return-value:      --
*
*    Global variables:  --
*
************************************************************/

void I2CMasterInit()
{
unsigned char cRc;

   SDAIn(cRc)
   if(!cRc)
      I2CMasterStop();
}
```

```
/*************************************************************
*
*
*    Procedure name:    I2CMasterStart
*
*    Function:          Generates a startcondition
*
*    Input-parameters:  --
*
*    Output-parameters: --
*
*    Return-value:      --
*
*    Global variables:  --
*
*
*************************************************************/

void I2CMasterStart()
{
   SDAHigh
   SCLHigh
   SDALow
   SDALow      // timing!
   SDALow      // timing!
   SCLLow
}


/*************************************************************
*
```

```
*
*    Procedure name:    I2CMasterWrite
*
*    Function:          Transmits one byte
*
*    Input-parameters:  cEx : byte to write
*
*    Output-parameters: --
*
*    Return-value:      1 : if no acknowledge from I2C Bus
*                           slave receiver
*                       0 : if acknowledge from I2C Bus slave
*                           receiver
*
*    Global variables:  --
*
*
*************************************************************/

unsigned char I2CMasterWrite(unsigned char cEx)
{
unsigned char cMask, cCnt;

   cMask = 0x80;
   for(cCnt = 0; cCnt < 8; cCnt++)
   {
      if(cMask & cEx)
         SDAHigh
      else

         SDALow
```

```
      cMask = cMask >> 1;
      SCLHigh
      SCLLow
   }
   SDAHigh
   SCLHigh
   SDAIn(cMask)
   SCLLow
   return cMask;
}


/*************************************************************
*
*
*    Procedure name:    I2CMasterRead
*
*    Function:          Reads one byte on
*
*    Input-parameters:  cAck
*                          1 : send acknowledge after
*                              byte transmission
*                          0 : no acknowledge after
*                              transmission(=>last byte send)
*
*    Output-parameters: --
*
*    Return-value:      received byte
*
*    Global variables:  --
*
```

```
*
************************************************************/

unsigned char I2CMasterRead(unsigned char cAck)
{
unsigned char cMask, cReceiveByte, cCnt, cRc;

   cReceiveByte = 0;
   SDAHigh
   cMask = 0x80;
   for(cCnt = 0; cCnt < 8; cCnt++)
   {
      SCLHigh;
      SDAIn(cRc)
      if(cRc)
         cReceiveByte |= cMask;
      SCLLow
      cMask = cMask >> 1;
   }
   SDAHigh
   if(cAck)
      SDALow
   SCLHigh
   SCLLow
   return cReceiveByte;
}


/************************************************************
*
*
```

```
*    Procedure name:    I2CSendDeviceAddress
*
*    Function:          Generates start condition, device
*                       address, subaddress
*
*    Input-parameters:    cAddress : IC slave address
*                       iSubAddress : subaddress of I2C bus
*                                     slave IC,
*                                     start address (first
*                                     byte) of data block,
*                                     if '-1': no subaddress
*                                     to be send
*
*    Output-parameters: --
*
*    Return-value:               0 : operation successful
*           ERR_I2C_NO_ADDRESS_ACK : no ack for IC-address
*        ERR_I2C_NO_SUBADDRESS_ACK : no ack for
*                                     subaddress/first byte
*
*    Global variables:  --
*
*
*************************************************************/

unsigned char I2CSendDeviceAddress(unsigned char cAddress,
int iSubAddress)
{
unsigned char cCnt;

   I2CMasterInit();
```

```
   for(cCnt = 0; cCnt < MAXATTEMPTS; cCnt++)
   {
      I2CMasterStart();
      if(!I2CMasterWrite(cAddress))
      {
         if(iSubAddress != -1)
            if(I2CMasterWrite(iSubAddress))
               return ERR_I2C_NO_SUBADDRESS_ACK;
         return 0;
      }
   }
   return ERR_I2C_NO_ADDRESS_ACK;
}


/*************************************************************
*
*
*    Procedure name:    I2CMasterReceiver
*
*    Function:          Generates a repeated start condition,
*                       followed by a device read
*                       address
*
*    Input-parameters:  cAddress : IC slave address
*
*    Output-parameters: --
*
*    Return-value:      0 : operation successful(acknowledge
*                           by I2C Bus slave receiver)
*                       1 : no acknowledge by I2C Bus slave
```

```
*                           receiver
*
*    Global variables:  --
*
*
************************************************************/

unsigned char I2CMasterReceiver(unsigned char cAddr)
{
   cAddr |= 0x01;
   I2CMasterStart();
   return I2CMasterWrite(cAddr);
}


/************************************************************
*
*
*    Procedure name:    I2CWrite
*
*    Function:          Write a number of bytes on I2C Bus
*
*    Input-parameters:    cAddress : IC slave address
*                       iSubAddress : subaddress of I2C Bus
*                                     slave IC, start address
*                                    (first byte) of data
*                                    block in I2C Bus slave IC,
*                                    if '-1': no subaddress to
*                                    be send
*                                buffer : pointer (address) to
*                                     start of transfer buffer
```

```
*                           count : number of bytes to be
*                               send
*
*    Output-parameters: --
*
*    Return-value:               0 : operation successful
*              ERR_I2C_BUS_DISABLE : bus disabled by user
*              ERR_I2C_NO_DATA_ACK : no ack for data byte
*           ERR_I2C_NO_ADDRESS_ACK : no ack for IC-address
*        ERR_I2C_NO_SUBADDRESS_ACK : no ack for subaddress
*                                1 : no acknowledge by I2C
*                                    Bus slave receiver
*
*    Global variables:  --
*
*
************************************************************/

unsigned char I2CWrite(unsigned char cAddress, unsigned int
iSubAddress, void *buffer, unsigned int iCount)
{
unsigned int iCnt;

   //bus free?
   if(I2CGetEnable())
   {
      return ERR_I2C_BUS_DISABLE;
   }
   //send IC-address, subaddress
   iCnt = I2CSendDeviceAddress(cAddress, iSubAddress);
   if(iCnt)
```

```
   {
      I2CMasterStop();
      return iCnt;
   }
   for(iCnt = 0; iCnt < iCount; iCnt++)
   {
      //write databyte
      if(I2CMasterWrite(*(unsigned char*)buffer))
      {
         I2CMasterStop();
         return ERR_I2C_NO_DATA_ACK;
      }
      ((unsigned char*)buffer)++;
   }
   I2CMasterStop();
   return 0;
}


/*************************************************************
*
*
*    Procedure name:    I2CRead
*
*    Function:          Read a number of bytes from the I2C
*                      Bus
*
*    Input-parameters:    cAddress : IC slave transmitter
*                                    address
*                      iSubAddress : subaddress of I2C Bus
*                                  slave IC, start address
```

```
*                                   first byte) of data block
*                                  in I2C Bus slave IC,
*                                   if '-1': no subaddress to
*                                  be send
*                             buffer : pointer (address) to
*                                     start of receive space
*                             iCount : number of bytes to be
*                                     read
*
*    Output-parameters: --
*
*    Return-value:               0 : operation successful
*               ERR_I2C_BUS_DISABLE : bus disabled by user
*            ERR_I2C_NO_ADDRESS_ACK : no ack for IC-address
*         ERR_I2C_NO_SUBADDRESS_ACK : no ack for subaddress
*
*    Global variables:  --
*
*
**************************************************************/

unsigned char I2CRead(unsigned char cAddress, unsigned int
iSubAddress, void *buffer, unsigned int iCount)
{
unsigned int iCnt;

   //bus free?
   if(I2CGetEnable())
   {
      return ERR_I2C_BUS_DISABLE;
   }
```

```
   iCnt = I2CSendDeviceAddress(cAddress, iSubAddress);
   if(iCnt)
   {
      I2CMasterStop();
      return iCnt;
   }
   SDAHigh
   //send repeated startcondition, IC_address read
   iCnt = I2CMasterReceiver(cAddress);
   if(iCnt)
   {
      I2CMasterStop();
      return iCnt;
   }
   if(!iCount)
      return 0;
   for(iCnt = 0; iCnt < (iCount-1); iCnt++)
   {
      //read databyte with acknowledge from master
      *(unsigned char*)buffer = I2CMasterRead(1);
      ((unsigned char*)buffer)++;
   }
    //read databyte with no acknowledge from master
   *(unsigned char*)buffer = I2CMasterRead(0);
   I2CMasterStop();
   return 0;
}


/*************************************************************
*
```

```
*
*    Procedure name:    I2CReadMsp
*
*    Function:          Read a number of bytes from the I2C
*                       Bus
*
*    Input-parameters:  cAddress : IC slave transmitter
*                                 address
*                    iSubAddress : subaddress in I2C bus
*                                  slave IC, start address
*                                  first byte of data block in
*                                  I2C bus slave IC,
*                      if '-1': no subaddress to be read
*                       stMspDat : pointer to structure type
*                                  TMspI2CData
*
*    Output-parameters: --
*
*    Return-value:                       0 : operation
*                                            successful
*                    ERR_I2C_BUS_DISABLE : bus disabled by
*                                          user
*                    ERR_I2C_NO_ADDRESS_ACK : no ack for IC-
*                                             address
*                 ERR_I2C_NO_SUBADDRESS_ACK : no ack for
*                                             subaddress
*
*    Global variables:  --
*
*
************************************************************/
```

```
unsigned char I2CReadMsp(unsigned char cAddress, unsigned int
iSubAddress,struct TMspI2CData *stMspDat)
{
unsigned int iCnt;

   //bus free?
   if(I2CGetEnable())
   {
      return ERR_I2C_BUS_DISABLE;
   }
   //send IC-address,subaddress
   iCnt = I2CSendDeviceAddress(cAddress, iSubAddress);
   if(iCnt)
   {
      I2CMasterStop();
      return iCnt;
   }

   I2CMasterWrite(HIBYTE(stMspDat->iMspSubAddr));
   I2CMasterWrite(LOBYTE(stMspDat->iMspSubAddr));

   SDAHigh
   //send repeated startcondition, IC_address read
   iCnt = I2CMasterReceiver(cAddress);
   if(iCnt)
   {
      I2CMasterStop();
      return iCnt;
   }
   //read databyte with acknowledge from master
```

```
    stMspDat->cMspDataH = I2CMasterRead(1);
    //read databyte with no acknowledge from master
    stMspDat->cMspDataL = I2CMasterRead(0);
    I2CMasterStop();
    return 0;
}
```

A.2 I2C_bus.h

```
/************************************************************
*
*
*    Module name:          RCSfile: I2c_bus.h
*
*    Function:             Definitions for related source
*                          file
*
*    Revision number:      Revision: 1.1
*
*    Last modification:    Date: 11/5/2006
*
*    Last modification by: Author: Serdar
*
*
************************************************************/

#ifndef _I2CBUS_H
#define _I2CBUS_H

extern void I2CSetEnable(unsigned char cEnable);
extern unsigned char I2CGetEnable();
extern void I2CMasterStop();
extern void I2CMasterInit();
extern void I2CMasterStart();
extern unsigned char I2CMasterWrite(unsigned char cEx);
extern unsigned char I2CMasterRead(unsigned char cAck);
```

```
extern unsigned char I2CSendDeviceAddress(unsigned char
cAddress, int iSubAddress);
extern unsigned char I2CMasterReceiver(unsigned char cAddr);
extern unsigned char I2CWrite(unsigned char cAddress,
unsigned int iSubAddress, void *buffer, unsigned int iCount);
extern unsigned char I2CRead(unsigned char cAddress, unsigned
int iSubAddress, void *buffer, unsigned int iCount);

#define ERR_I2C_BUS_DISABLE         0x01
#define ERR_I2C_NO_ADDRESS_ACK      0x02
#define ERR_I2C_NO_SUBADDRESS_ACK   0x04
#define ERR_I2C_NO_DATA_ACK         0x08


#define NOP _nop_();
#define MAXATTEMPTS 3
#define MAX_CLOCKSTRETCHING 5000 // one cyclus is about 5 us;
MSP needs at least 7 ms in Read Mode

#endif
```