Embedded System Development I-II

Major Project Report

Submitted in the partial fulfillment of the requirements for the degree of

Bachelor of Technology

In

ELECTRONICS AND COMMUNICATIONS ENGINEERING

By

POOJA GANDHI [05BEC022]

RAJVI SHAH [05BEC076]

Under the guidance of
Mr. Akash Mecwan

Department of Electrical Engineering
Electronics & Communication Engineering Branch
Institute of Technology
Ahmedabad 382 481
April 2009

CERTIFICATE

This is to certify that the Major Project Report entitled **"Embedded System Development-I/II"** submitted by the following students as the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics & Communication of Institute of Technology **Nirma University** is the record of work carried out by them under our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for the examination. The results embodied in this minor 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.

Roll No.	**Name of the Student**
05BEC022	POOJA GANDHI
05BEC076	RAJVI SHAH

Date:

Mr. Akash Mecwan
Project Guide.

Prof. A.S.Ranade
HOD, (EE).

ACKNOWLEDGEMENT

We would like to take the opportunity to express our sincere gratitude to our Project Guide, **Mr. Akash Mecwan**, Lecturer, Electronics and Communication Department, Institute of Technology, Nirma University, for first of all agreeing to guide us for the project. Without his constant motivation and support, this work may not have reached the level that it has. We would like to thank Institute of Technology, Nirma University for providing us valuable resources and the opportunity to carry out the work seamlessly.

Also we would like to thank *Texas Instruments, Analog Devices, Maxim Dallas Semiconductors, Avnet Electronics India* for providing us sample components for experiments. Also, with heart-felt gratitude, we would like to thank all the Embedded Hobbyists and AVR Freaks across the globe for providing great help and support through Internet Community.

Not to forget our Team members: Digant, Dhaval, Ritesh and Sarth. This project and its success was a result of the joint efforts we made.

On top of everything we wish to express our gratitude to our family and friends for being the solid support system that pushed us to reach the success that we have achieved today.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The great development in semiconductor technology has given the world a magical word: “miniaturization”. The odyssey of electronics has traveled long ways, from Hall size vacuum tube computers to laptops and palmtops and the era has begun where the market is captured by portable electronics with a variety of applications. With the platform provided by the semiconductor world in terms of Application processors and by the software industry in terms of OS, Embedded Systems are becoming more and more powerful and cost effective. This project was started with a goal of making use of the best of both worlds- the processing capabilities of an 8- bit microcontroller (ATmega128) without any Operating System as well as an Open Source Operating System (Embedded Linux) installed on a powerful 32- bit processor (AP7000). This report is divided in two parts. The first involves a comprehensive review of the modules used in developing a Portable Multimedia Player that is based on the 8- bit controller and the details of its functionalities including the communication protocols that were used. While the second part covers phase-2 of the project i.e. the exploration and development of audio/ video applications on the 32- bit processor for Embedded Linux.

Chapter 1

Introduction

Introduction

1.1 Defining an Embedded System

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Since the embedded system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product, or increasing the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically, embedded systems range from portable devices such as digital watches and MP3 players to large stationary installations like traffic lights, factory controllers, or the systems which control nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not an exactly defined term, as many systems have some element of programmability. For example, Handheld computers share some elements with embedded systems — such as the operating systems and microprocessors which power them — but are not truly embedded systems, because they allow different applications to be loaded and peripherals to be connected.

1.2 Motivation

The use of technology began with the conversion of natural resources into simple tools. The pre-historical discovery of the ability to control fire increased the available sources of food and the invention of the wheel helped humans in traveling in and controlling their environment. Technology has affected society and its surroundings in a number of ways.

The MP3 player is the most recent in an evolution of music formats that have helped consumers enjoy their tunes. Records, 8-track tapes, cassette tapes and CDs -- none of these earlier music formats provide the convenience and control that MP3 players deliver. With an MP3 player in

hand or pocket, a consumer can create personalized music lists and carry thousands of songs wherever they go. They come in many different brands and with many different options, but all have a common purpose: Personal audio enjoyment.

All of that stored music and the MP3 player itself fit into a device that, in some cases, weighs less than one ounce. Portability is a large factor in the popularity of the MP3, considering the ease of transportation in comparison to a CD player and CD storage case. In addition, some devices provide additional technology, like video and photo viewing, alarm and calendar functions, and even cell phone and Internet service.

1.3 Goal

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems control many of the common devices in use today.

With the platform provided by the semiconductor world in terms of Application processors and by the software industry in terms of OS, Embedded Systems are becoming more and more powerful and cost effective. As explained the domain of Embedded System is wide and thousands of embedded applications do exist. But here we intend to focus on Embedded Systems with Multimedia Computing capabilities as it will touch all the aspects of Embedded Systems and also a few aspects of Multimedia Systems, Signal Processing and Networking.

The aim of the project is to explore the field of Embedded Technology and develop a multimedia device which fulfills three basic criteria: portability, scalability and performance. This aimed multimedia device is expected to include both, the high degree of performance of media players and scalability of OS based PDAs and Mobile phones.

1.4 Phases

Audio / Video / Imaging applications are high end applications and implementing them simultaneously requires more resources in terms of memory as well as processing power than a General Purpose 8-bit Microcontroller Unit. Though an 8 / 16-bit MCU with moderate to high clock support and sufficient buffer memory can suffice for WAVE Audio decoding and Picture Viewing but Video, Imaging and streaming media applications require a higher clock and more processing power in terms of its MIPS. These requirements are necessarily fulfilled by Application Processors.

Hence the project is divided into two phases:

1. Developing the low end features on an 8/16 bit General Purpose MCU

   Among various available 8-bit MCUs architecture by Intel **8051** is the oldest and most popular. 8051 and its derivatives are now manufactured by almost all big MCU manufacturers such as Intel, Atmel and is still the most popular 8 bit MCU. But many new architectures are in fact better than 8051 and provides more resources than 8051. Two of such architectures are **PIC** (by Microchip) and **ATmega** (by AVR). PIC being cost effective and efficient, ATmega is an equal competition and is becoming more and more popular among students and electronics hobbyists because of wide range of software support being provided. For 16-bit MCUs, PIC16 (by Microchip), LPC (by Phillips) and MSP (by Texas Instruments) are some other 16-bit MCUs available. Here, we have opted for an 8 bit processor for initial phase, an ATmega family microcontroller - ATmega128 , which includes following features,

   Features of ATmega128 [1]

   - Performance: up to 16 MIPS a 16MHz
   - Memory: 32 x 8 General Purpose Registers, 128KB Flash Memory, 4KB EEPROM, 4KB SRAM
   - Peripherals: Two 8 bit & Two 16 bit timer/Counter, RTC, 1 ADC, 1 USART, WDT, SPI, I2C, etc

2. Developing a high-end device on a 32-bit Application Processors with an Operating System

   The MCU selection for phase II of the project is crucial. Application Processors are specifically designed for Multimedia and Networking applications with Low power and high performance considerations. The following parameters were chosen as a selection criteria,

   - CPU core and Performance

   Some of the Application processors may have a core of a new architecture where as some APs incorporate the 8051 architecture or ARM based core. The potential of these APs is very high clock rates they operate at and high MIPS they provide. Some Application Processors (OMAP) have dual core: one GP core and another DSP core, providing dual features. ARM based cores (specifically ARM TDMI and Strong ARM) gives the best MIPS performance (up to 750 MIPS).

   - OS support

   Most of the AP manufacturers provide either OS or OS kernel firmware to be loaded into the Application Processor. The OS is generally developed by AP Manufacturer Company itself. Such as uC OS or the OS may be a Linux derivative like Embedded Linux or uC Linux etc. If the firmware for desired OS is not provided by the manufacturer than it is required to port the desired OS to the Application Processor.

   - On-chip Peripheral Controllers

   The most basic criteria for any processor to be an Application Processor are to include the required peripheral controllers [such as LCD controller, USB controller etc] on chip. Different APs provide different peripheral controllers. According to the application, selection of AP is determined by these peripherals as well.

   Some of the available Application Processors are OMAP series (by TI), C64xx series (by TI), Mobile SoC series (by Samsung) and AP7000 series (by AVR). For phase II AP7000 is selected though having OMAP L137 in strong competition because of low cost of starter kit with more features included.

Features of AP7000 [2]

- Core: Single core (AVR 32 Architecture)
- Performance : 210 MIPS at 150MHz,
- Memory: 16KB instruction as well as data cache,32 KB SRAM + Ext. Memory
- External Memory: SDRAM, SRAM, DataFlash, SD, MMC, CF
- On chip controllers: DMA, Pixel Coprocessor, LCD controller, USB controller
- Peripherals: 6 Timers/Counters, 6 USART, 16-bit audio DAC, I2C, SPI etc.

Embedded System Development - I

Portable Multimedia Player using ATmega128

Chapter 2

Black-box Design

Black-box Design

This chapter describes the hardware design in a simple black box diagram manner. In this design the connection and interfaces between all the individual modules are illustrated and functionality of each unit is is explained in brief without going into details of processing and protocols.

2.1 Block Diagram

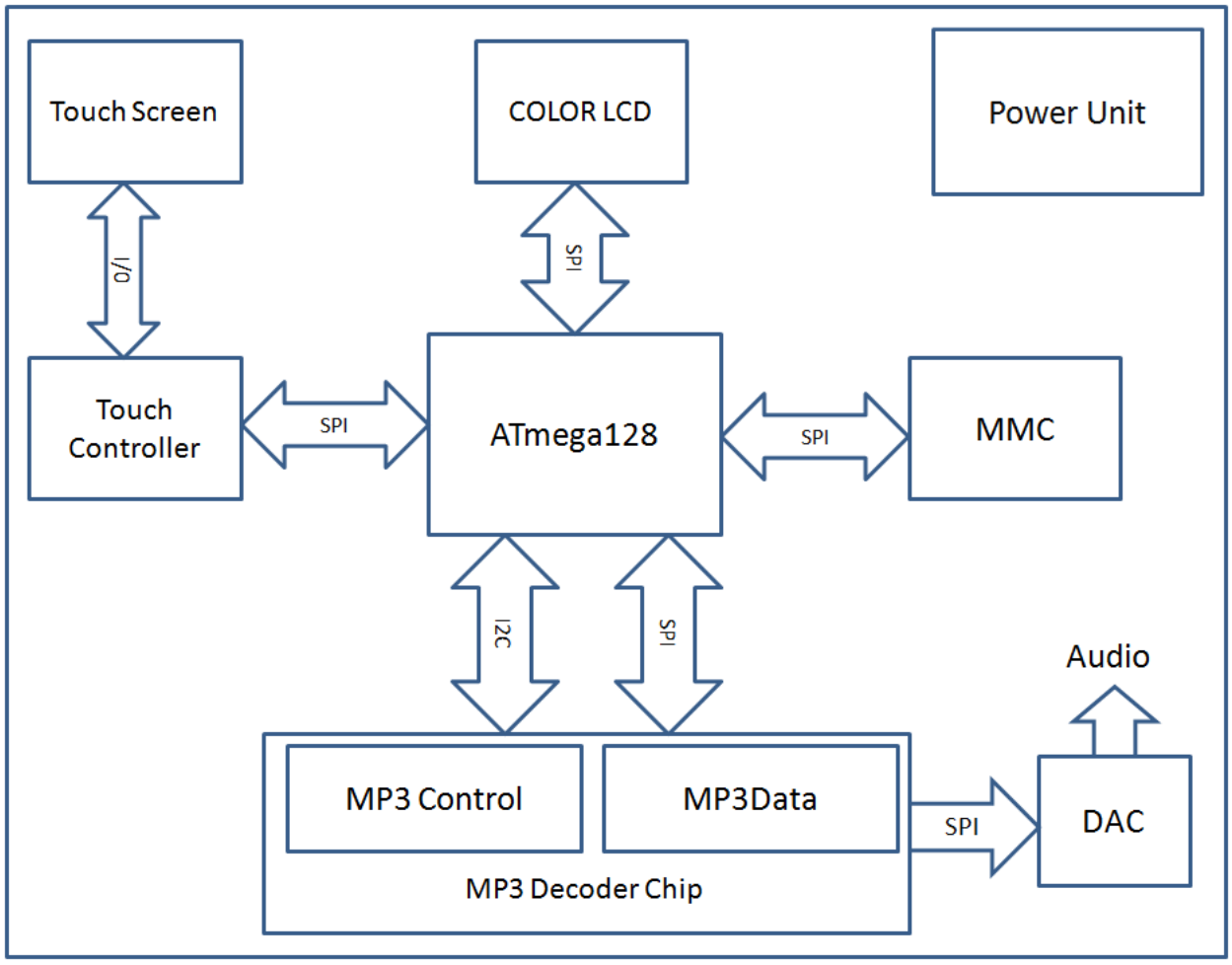


Fig 2.1 A general block diagram of complete system

2.2 Functional Units

The various functional units are described in brief here. Details of interfacing etc. are given in the chapters that follows.

2.2.1 ATmega128

ATmega128 is the heart of the design. It is an 8-bit RISC MCU which controls all the peripheral devices as well as does the book keeping work. ATmega128 is a 64 pin chip. Providing 7 GPIO ports which has multiplexed functionality. Hence, provides great ease to accommodate all the peripherals. Details about the functionalities of ATmega128 used can be found in Chapter 3.

2.2.2 Color LCD

The aim of keeping a color LCD in design is to display pictures, simple animations etc, making it a Multimedia Player in a real sense. Color LCD also makes the player look more elegant of course at some cost. To lower the cost, we used a 132x132 pixel, 12 bit RGB color LCD used in Nokia 6610/6100 cell phone series. The details of this LCD and its interfacing are provided in Chapter 4.

2.2.3 Touch Screen

Though including touch Screen to the player design is not essential, but it surely adds a great functionality, also reduces the overall design size. It is a great alternative to mechanical keypad. Here, in the design a 4-wire resistive touch screen is used. Unlike other peripherals, this touch screen does not have an inbuilt controller. Hence it is interfaced using a touch screen controller ADS7843. Details of its interfacing are provided in Chapter 5.

2.2.4 Flash Memory Interface

The aimed multimedia device has to be portable and portability imposes constraints on type of memory to be interfaced. Hence, low power, compact size and economical memory is preferable to be interfaced. The obvious solution is Flash Memory with additional advantage of High Data Transfer Speed. In market Flash Memory is available in form of Memory cards which is Flash

memory with controller. From variety of available Memory Cards MMC is selected. The reason is the same software with little modification can be used for SD cards also. Details on this can be found in Chapter 6.

2.2.5 MP3 Decoder unit

MP3 is the most popular standard today for digital audio playback. The key behind its encoding is to exploit human auditory limitations or auditory masking. Though decoding is far simpler than encoding process it yet demands the processing power that can't be delivered by ATmega128. Hence a dedicated hardware mp3 decoder chip "STA013" is used to perform data decoding. ATmega128 controls the chip as well as act as data buffer between MMC and STA013. Output of this chip is PCM digital data which is then passed through a DAC CS4334 to deliver audio output data. Details on the same can be found in Chapter 8.

Chapter 3

ATmega128

3.1 SPI Interface

3.2 8-bit Timer/Counter 0

3.3 16-bit Timer/Counter 1

3.4 ADC

ATmega128

This chapter concentrates on various features of ATmega128 which are used for different peripheral applications. We have seen various features of ATmega128 in previous chapter, this chapter deals with only the features used by application program. From available interfaces for our application, we have used SPI Bus Interface for MMC interfacing and In system Programming, 8 bit Timer (Timer 0) and 16 bit Timer (Timer 1) for WAVE playback application, ADC for Touch Screen Interfacing.

3.1 SPI – Serial Peripheral Interface

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATmega128 and peripheral devices with SPI communication hardware. The ATmega128 SPI includes the following features:

- Full-duplex, Three-wire Synchronous Data Transfer
- Master or Slave Operation
- LSB First or MSB First Data Transfer
- Seven Programmable Bit Rates
- End of Transmission Interrupt Flag
- Double Speed (CK/2) Master SPI Mode

The interconnection between Master and Slave CPUs with SPI is shown in Fig.3.1.The system consists of two Shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select SS pin of the desired Slave. Master and Slave prepare the data to be sent in their respective Shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In– Slave Out, MISO, line. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select line (SS active low).

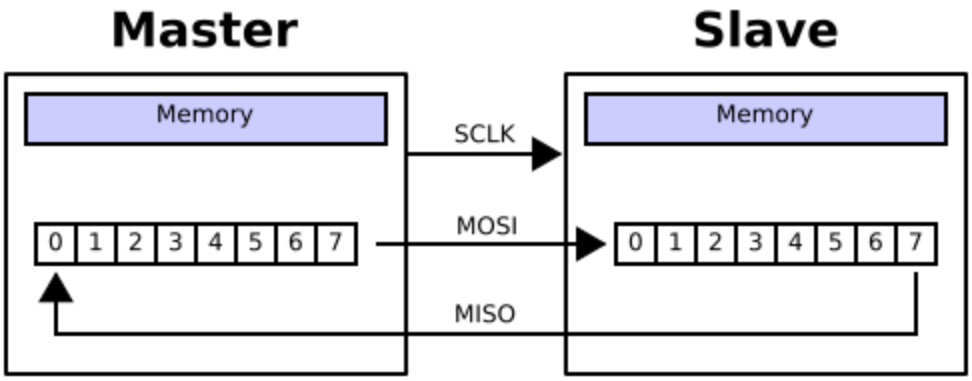


Fig 3.1 SPI communication between Master and Slave

3.1.1 SPI Registers

The SPI communication is controlled by three very important registers, Shift Register SPDR, Control register SPCR and Status Register SPSR.

1. **SPCR – Serial Peripheral Control Register**

   The various bit fields of SPCR register and their roles in SPI communication are explained as follows:

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7 – SPIE: SPI Interrupt Enable: This bit causes the SPI interrupt to be executed if SPIF bit in the SPSR Register is set and the if the global interrupt enable bit in SREG is set.

- Bit 6 – SPE: SPI Enable: When the SPE bit is written to one, the SPI is enabled. This bit must be set to enable any SPI operations.

- Bit 5 – DORD: Data Order: When the DORD bit is written to one, the LSB of the data word is transmitted first. When the DORD bit is written to zero, the MSB of the data word is transmitted first.

- Bit 4 – MSTR: Master/Slave Select: This bit selects Master SPI mode when written to one, and Slave SPI mode when written logic zero. If SS is configured as an input and is driven low while MSTR is set, MSTR will be cleared, and SPIF in SPSR will become set. The user will then have to set MSTR to re-enable SPI Master Mode.

- Bit 3 – CPOL: Clock Polarity: When this bit is written to one, SCK is high when idle. When CPOL is written to zero, SCK is low when idle.

- Bit 2 – CPHA: Clock Phase: The settings of the clock phase bit (CPHA) determine if data is sampled on the leading (first) or trailing (last) edge of SCK.

  These two bits CPHA and CPOL together are used to configure the clock polarity and phase with respect to the data. The combination of these two bits determines four modes of SPI communication.

  Mode 0 and Mode 1

  At CPOL=0 the base value of the clock is zero

  - Mode 0 : For CPHA=0, data are read on the clock's rising edge and data are changed on a falling edge.
  - Mode 1: For CPHA=1, data are read on the clock's falling edge and data are changed on a rising edge.

  Mode 2 and Mode 3

  At CPOL=1 the base value of the clock is one (inversion of CPOL=0)

  - Mode 2: For CPHA=0, data are read on clock's falling edge and data are changed on a rising edge.
  - For CPHA=1, data are read on clock's rising edge and data are changed on a falling edge.

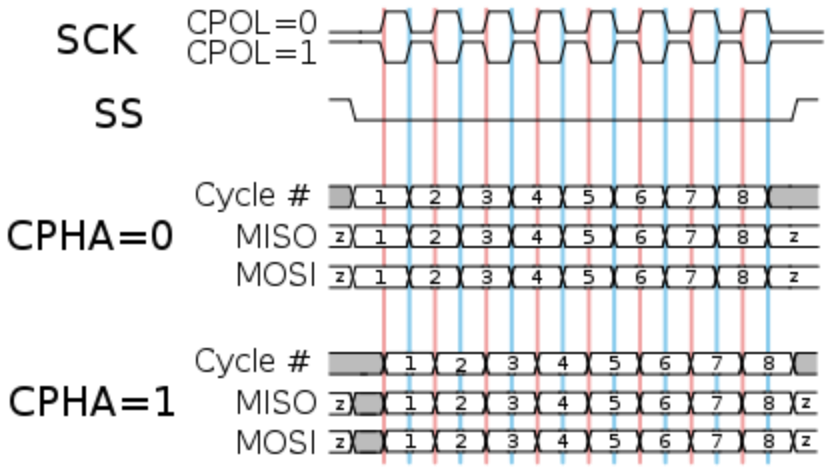


Fig.3.2 Timing Diagram for different values of CPOL and CPHA

- Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0: These two bits control the SCK rate of the device configured as a master. SPR1 and SPR0 have no effect on the slave. The relationship between SCK and the Oscillator Clock frequency fosc according to these bit values is shown in the following table,

SPI1	SPI0	SCK Freq
0	0	Fosc/4
0	1	Fosc/16
1	0	Fosc/64
1	1	Fosc/128

Table 3.1 Setting SPI Frequency

2. SPSR – Serial Peripheral Status Register

The various bit fields of SPSR register and their roles in SPI communication are explained as follows:

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7 – SPIF: SPI Interrupt Flag: When a serial transfer is complete, the SPIF flag is set. An interrupt is generated if SPIE in SPCR is set and global interrupts are enabled.

- Bit 6 – WCOL: Write COLlision flag: The WCOL bit is set if the SPI Data Register (SPDR) is written during a data transfer. The WCOL bit (and the SPIF bit) is cleared by first reading the SPI Status Register with WCOL set, and then accessing the SPI Data Register.

- Bit 5..1: Reserved Bits: These bits are reserved bits and will always read as zero.

- Bit 0 – SPI2X: Double SPI Speed Bit: When this bit is written logic one the SPI speed (SCK Frequency) will be doubled when the SPI is in Master mode. This means that the minimum SCK period will be 2 CPU clock periods. When the SPI is configured as Slave, the SPI is only guaranteed to work at fosc /4 or lower.

3. SPDR – Serial Peripheral Data Register
   The SPI Data Register is a Read/Write Register used for data transfer between the register file and the SPI Shift Register. Writing to the register initiates data transmission. Reading the register causes the Shift Register Receive buffer to be read.

*The SPI interface on the ATmega128 is also used for program memory and EEPROM downloading or uploading. Details on programming can be obtained from datasheet of ATmega128 pg. 300

3.1.2 SPI Master Mode

When configured as a Master, the SPI interface has no automatic control of the SS line. This must be handled by user software before communication can start. When this is done, writing a byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the 8 bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of transmission flag (SPIF). If the SPI interrupt enable bit (SPIE) in the SPCR Register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or

signal the end of packet by pulling high the Slave Select, SS line. The last incoming byte will be kept in the buffer register for later use.

3.2 8-bit Timer/Counter 0

Timer 0 is an 8-bit Timer Counter which is used in WAVE playback application to generate interrupt at sampling frequency. Typically the timer's operation is to count the input frequency. The input frequency may be a pre-scaled value of main clock frequency [Timer] or a separate oscillator circuit may be used [Counter]. Our end application uses a prescaled input frequency to perform counting.

The timer 0 has four different modes of operations and there are two interrupts associated with timer 0.One is Timer 0 overflow which is generated when timer 0 reaches it's maximum value [0xFF] and another is Timer 0 compare match. For Sampling Rate generation Output Compare Match Interrupt is used. Prior to describing Timer 0 compare match interrupt , it is appropriate to list the registers crucial for timer operation.

3.2.1 Timer 0 Registers

1. TCCR0 – Timer Counter Control Register

This register controls timer's mode of operation [WGM00:1], timer unit input frequency [CS02:1:0] and output at OC0 [PIN 14] on compare match.

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7 – FOC0: Force Output Compare

  When writing a logical one to the FOC0 bit, an immediate compare match is forced on the waveform generation unit. The OC0 output is changed according to its COM01:0 bits setting. Note that the FOC0 bit is implemented as a strobe. Therefore it is the value present in the COM01:0 bits that determines the effect of the forced compare. A FOC0 strobe will not

generate any interrupt, nor will it clear the timer in CTC mode using OCR0 as TOP. The FOC0 bit is always read as zero.

- Bit 6, 3 – WGM01:0: Waveform Generation Mode

  These bits control the counting sequence of the counter, the source for the maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode, Clear Timer on Compare match (CTC) mode, and two types of Pulse Width Modulation (PWM) modes.

Mode	WGM01(1) (CTC0)	WGM00(1) (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Table 3.2 Setting Mode of Operation

- Bit 5:4 – COM01:0: Compare Match Output Mode

  These bits control the output compare pin (OC0) behavior. If one or both of the COM01:0 bits are set, the OC0 output overrides the normal port functionality of the I/O pin it is connected to. When OC0 is connected to the pin, the function of the COM01:0 bits depends on the WGM01:0 bit setting. *Table 3.3* shows the COM01:0 bit functionality when the WGM01:0 bits are set to a normal or CTC mode (non-PWM).

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Table 3.3 Setting Output Behavior

- Bit 2:0 – CS02:0: Clock Select

  The three clock select bits select the clock source to be used by the Timer/Counter. Total 7 different clock values are supported. Lowest being Clk/1024 and Highest is Clk itself. If all three bits are zero valued, the timer is stopped

2. TCNT0 – Timer Counter 0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCNT0 is an 8-bit timer counter register. Its value is incremented at each timer clock. It allows read/write access during counting but writing or modifying TCNT0 during counting introduces risk of missing a compare match.

3. OCR0 – Output Compare register 0

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Register contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC0 pin.

4. ASSR0 – Asynchronous Status Register

This register has significance for Asynchronous mode of operation. But this mode is not used by any of the end application and hence not discussed here. Interested reader can refer to ATmega128 Datasheet pg. 107.

5. TIMR and – Timer Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 1 – OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable**

  When the OCIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Compare Match interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, i.e., when the OCF0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

  When the TOIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

6. TIFR – Timer Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 1 – OCF0: Output Compare Flag 0

  The OCF0 bit is set (one) when a compare match occurs between the Timer/Counter0 and the data in OCR0 – Output Compare Register0. OCF0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0 is cleared by writing a logic one to the flag. When the I-bit in SREG*, OCIE0 (Timer/Counter0 Compare Match Interrupt Enable), and OCF0 are set (one), the Timer/Counter0 Compare Match Interrupt is executed.

- Bit 0 – TOV0: Timer/Counter0 Overflow Flag

  The bit TOV0 is set (one) when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable), and TOV0 are set (one), the Timer/Counter0 Overflow Interrupt is executed.

  *SREG is AVR status register and I-bit is used for a global interrupt enable.

3.2.2 Timer 0 operation in CTC Mode

In Clear Timer on Compare or CTC mode (WGM01:0 = 2), the OCR0 Register is used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNT0) matches the OCR0. Also if interrupt is enabled, a Timer 0 compare match interrupt is generated. The OCR0 defines the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. Hence, it is used to generate sampling frequency for audio playback. The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0, and then counter (TCNT0) is cleared. The operation of Timer 0 can be illustrated by following figure,

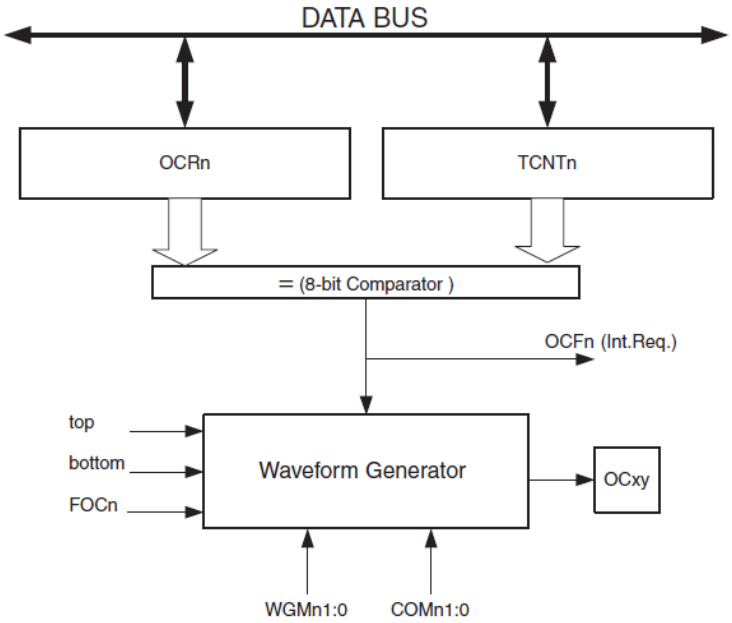


Fig.3.3 Output Compare Unit

The 8-bit comparator continuously compares TCNT0 with the Output Compare Register (OCR0). Whenever TCNT0 equals OCR0, the comparator signals a match. A match will set the output compare flag (OCF0) at the next timer clock cycle. If enabled (OCIE0 = 1), the output compare flag generates an output compare interrupt. The OCF0 flag is automatically cleared when the interrupt is executed. Alternatively, the OCF0 flag can be cleared by software by writing a logical one to its I/O bit location. The waveform generator uses the match signal to generate an output according to operating mode set by the WGM01:0 bits and compare output mode (COM01:0) bits.

3.3 16-bit Timer / Counter 1

Timer 1 is a 16-bit Timer Counter which is used in WAVE playback application to achieve digital to analog conversion using Fast PWM [Pulse Width Modulation] mode of timer. This timer is a source of ten different interrupts but they are not discussed here because they aren't used by the application. Just like timer 0 it has a set of different registers to control its operation and it can have different modes of operations. Here, only the Fast PWM mode is explained with the register configuration.

3.3.1 Timer1 Registers

1. Timer/Counter Control Register 1 A

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7:6 – COM1A1:0: Compare Output Mode for Channel A
- Bit 5:4 – COM1B1:0: Compare Output Mode for Channel B
- Bit 3:2 – COM1C1:0: Compare Output Mode for Channel C

For WAVE playback of mono channel files only Channel A is used and for stereo both cannels A and B are used. Channel C is not used by any end program. The COM1A1:0,

COM1B1:0, and COM1C1:0 control the output compare pins (OC1A, OC1B, and OC1C [PIN 15,16] respectively) behavior. If one or both of the COMnA1:0 bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. Same holds true for COM1B1:0. However, note that the *Data Direction Register* (DDR) bit corresponding to the OC1A, OC1B. When the OC1A, OC1B or OC1C is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting, these bits determine the mode of operation

- Bit 1:0 – WGMn1:0: Waveform Generation Mode

  Combined with the WGMn3:2 bits found in the TCCR1B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes. We have used Fast PWM mode – 8 bit. That is mode 5 [WGM13:0 = 0101b]. For other modes refer to ATmega128 datasheet pg. 135.

2. Timer/Counter Control Register 1 B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bits 7,6 are used to count external events and hence they are not used. and bit 5 is a reserved bit.

- Bit 4:3 – WGMn3:2: Waveform Generation Mode

  These two bits along with two bits in TCCR1A decides waveform generation mode.

- Bit 2:0 – CSn2:0: Clock Select

  The three clock select bits select the clock source to be used by the Timer/Counter. Similar to Timer 0 clock source, we can have 7 pre-scaled values.

3. TCNT1H and TCNT1L

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCNT is increased to reflect counting. The two Timer/Counter I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This Temporary Register is shared by all the other 16-bit registers. Modifying or writing the counter (TCNT1) while the counter is running introduces a risk of missing a compare match between TCNT1 and one of the OCR1x Registers.

4. OCR1AH and OCR1AL

Bit	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

5. OCR1BH and OCR1BL

Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNT1). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OC1x pin.

3.3.2 Fast PWM operation

The fast Pulse Width Modulation or fast PWM mode (WGMn3:0 = 5,6,7,14, or 15 , we have used mode 5) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the output compare (OC1x) is cleared on the compare match between TCNT1 and OCR1x, and set at BOTTOM. In inverting compare output mode output is set on compare match and cleared at BOTTOM.

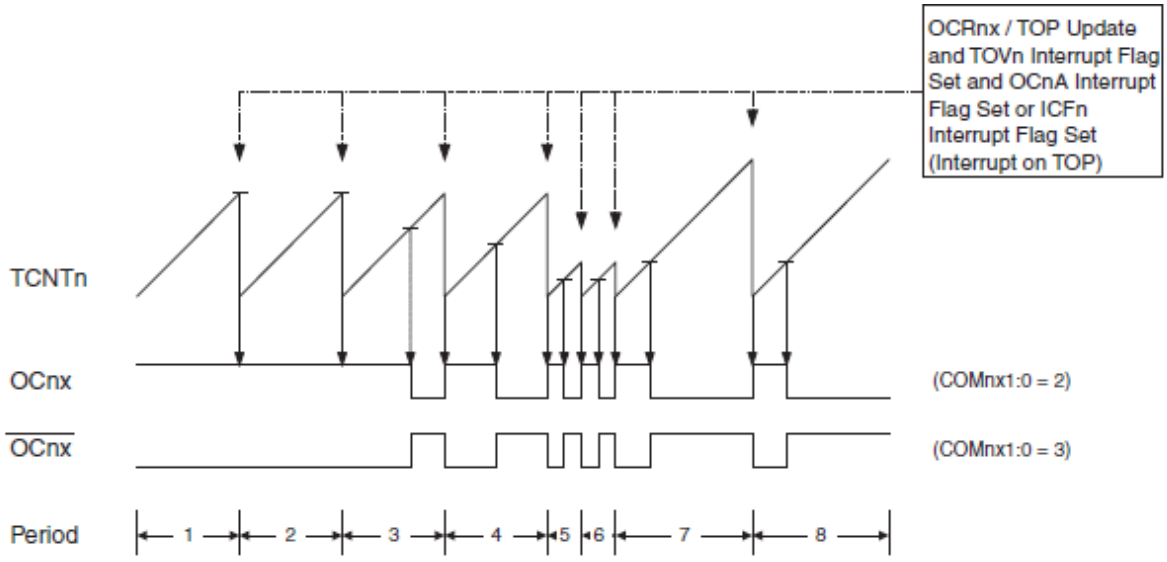


Fig 3.4 Timing Diagram for Fast PWM mode

3.4 ADC – Analog to Digital Converter

The ATmega128 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port F. The single-ended voltage inputs refer to 0V (GND). The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion.

3.4.1ADC Operation:

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation.The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The analog input channel and differential gain are selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed bandgap voltage reference, can be selected as single ended inputs to the ADC. A selection of ADC input pins can be selected as positive and negative inputs to the differential gain amplifier. If differential channels are selected, the differential gain stage amplifies the voltage difference between the selected input channel pair by the selected gain factor. This amplified value then ecomes the analog input to the ADC. If single ended channels are used, the gain amplifier is bypassed altogether.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the data registers belongs to the same conversion. Once ADCL is read, ADC access to data registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the data registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

3.4.2 ADC Communication:

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

In Free Running mode, the ADC is constantly sampling and updating the ADC Data Register. Free Running mode is selected by writing the ADFR bit in ADCSRA to one. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC Interrupt Flag, ADIF is cleared or not.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of an first conversion. When a conversion is complete, the result is written to the ADC data registers, and ADIF is set. In single conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high.

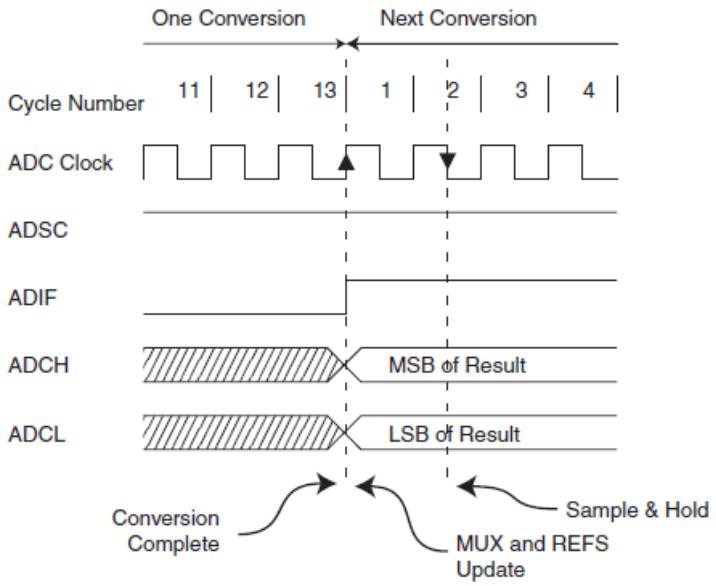


Fig 3.5 Timing Diagram of ADC Conversion

3.4.3 ADC Registers

1. ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7- 6 REFS1 and REFS0: These bits select the voltage reference for the ADC, as shown in below. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Table 3.4 Voltage Reference Selection for ADC

- Bit 5 ADLAR: The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted. Changing the ADLAR bit will affect the ADC Data Register immediately, regardless of any ongoing conversions

- Bit 4:0 MUX 4:0:The value of these bits selects which combination of analog inputs are connected to the ADC. These bits also select the gain for the differential channels. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete.

2. ADC Control and Status Register A – ADCSRA

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7: ADEN: ADC Enable: Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- Bit 6: ADSC: ADC Start Conversion: In Single Conversion mode, write this bit to one to start each conversion. In Free running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

- Bit 5 ADFR: ADC Free Running Select: When this bit is written to one, the ADC operates in Free Running mode. In this mode, the ADC samples and updates the data registers continuously. Writing zero to this bit will terminate Free Running mode.

- Bit 4: ADIF: ADC Interrupt Flag: This bit is set when an ADC conversion completes and the data registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a read-modify-write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

- Bit 3 ADIE: ADC Interrupt Enable: When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

- Bit 2:0 ADPS2:0: ADC Prescaler Select Bits: These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

3. The ADC Data Register – ADCL and ADCH

When an ADC conversion is complete, the result is found in these two registers.

ADLAR = 0:

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1:

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Chapter 4

Interfacing LCD

Interfacing LCD

This Chapter concentrates on two different types of LCD interfacing with our target MCU ATmega128. The first type of LCD is Passive Matrix LCD with HD44870 controller, this type of LCD was mainly used for debugging purpose, also it can be used for only audio player. Another type of LCD is the 12 bit RGB color LCD, the one used in Nokia 6100. It has in built controller PCF8833.

4.1 Introduction to LCD

A Liquid Crystal Display (LCD) is an electronically-modulated optical device shaped into a thin, flat panel made up of any number of color or monochrome pixels filled with liquid crystals and arrayed in front of a light source (backlight) or reflector. It is often utilized in battery-powered electronic devices because it uses very small amounts of electric power. The LCD is manufactured with an inbuilt microcontroller which takes care of taking commands and data and displays the data as required.

LCDs are classified in two categories. Passive matrix displays and Active matrix displays (Also known as TFTs). In our project, we have used both the LCDs. 16X2 b/w LCD and 132X132 Nokia color LCD. Both the types are described in brief below.

4.2 Passive Matrix LCD

LCDs with a small number of segments, such as those used in digital watches and pocket calculators, have individual electrical contacts for each segment. An external dedicated circuit supplies an electric charge to control each segment. This display structure is unwieldy for more than a few display elements. These types of LCD are basically character LCD and they are identified as No of Characters x No of Lines format. A few popular matrix displays are 16x2, 20x4 etc. We have used 16x2 LCD display with HD44870 controller.

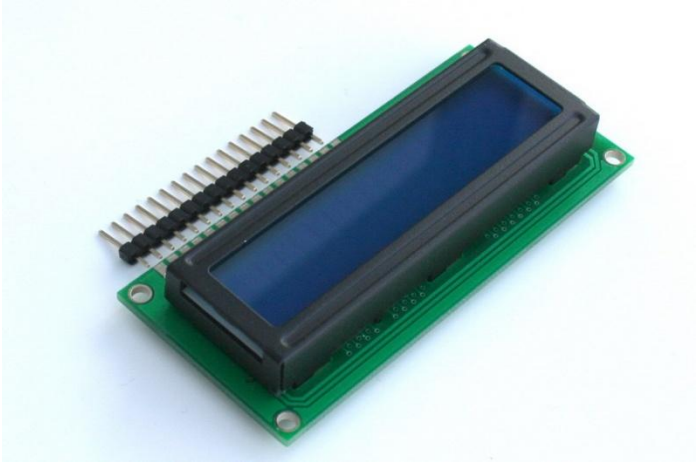


Fig. 4.1 16x2 Matrix LCD

4.2.1 Interfacing 16x2 Matrix LCD with HD44870 controller

The 44780 standard requires 3 control lines as well as either 4 or 8 I/O lines for the data bus. The user may select whether the LCD is to operate with a 4-bit data bus or an 8-bit data bus. If a 4-bit data bus is used the LCD will require a total of 7 data lines (3 control lines plus the 4 lines for the data bus). If an 8-bit data bus is used the LCD will require a total of 11 data lines (3 control lines plus the 8 lines for the data bus).

The three control lines are referred to as EN, RS, and RW. The EN line is called "Enable." This control line is used to tell the LCD that you are sending it data. To send data to the LCD, your program should make sure this line is low (0) and then set the other two control lines and/or put data on the data bus. When the other lines are completely ready, bring EN high (1) and wait for the minimum amount of time required by the LCD datasheet (this varies from LCD to LCD), and end by bringing it low (0) again.

The RS line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor, etc.). When RS is high (1), the data being sent is text data which should be displayed on the screen. For example, to display the letter "T" on the screen you would set RS high.

The RW line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD. Only one instruction ("Get LCD status") is a read command. All others are write commands--so RW will almost always be low.

Finally, the data bus consists of 4 or 8 lines (depending on the mode of operation selected by the user). In the case of an 8-bit data bus, the lines are referred to as DB0, DB1, DB2, DB3, DB4, DB5, DB6, and DB7. Here, we have used 4-bit mode to communicate so each byte is transferred in two steps, higher nibble and lower nibble [3].

4.3 Thin Film Transistor (Active Matrix) Color LCD

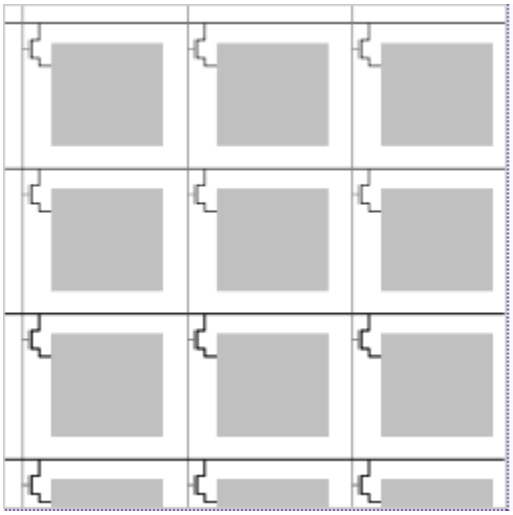


Fig 4.2 TFT display internal structure

Small liquid crystal displays as used in calculators and other devices have driven image elements- a voltage can be applied across one segment without interfering with other segments of the display. This is impractical for a large display with a large number of picture elements (pixels), since it would require millions of connections—top and bottom connections for each one of the three colors (red, green and blue) of every pixel. To avoid this issue, the pixels are addressed in rows and columns which reduce the connection count from millions to thousands. If all the pixels in one row are driven with a positive voltage and all the pixels in one column are driven with a negative voltage, then the pixel at the intersection has the largest applied voltage

and is switched. The problem with this solution is that all the pixels in the same column see a fraction of the applied voltage as do all the pixels in the same row, so although they are not switched completely, they do tend to darken. The solution to the problem is to supply each pixel with its own transistor switch which allows each pixel to be individually controlled. These are thin film transistors, special kind of field effect transistors made by depositing thin films of a semiconductor active layer as well as the dielectric layer and metallic contacts over a supporting substrate. The low leakage current of the transistor prevents the voltage applied to the pixel from leaking away between refreshes to the display image. Each pixel is a small capacitor with a layer of insulating liquid crystal sandwiched between transparent conductive ITO layers.

4.3.1 Interfacing Nokia 6610 Color LCD

The aimed media player has the capability of displaying images. A small size color LCD was the basic requirement for our project. But cost of color LCD hinders the process. Low cost solution is to use color LCD of cell phones. In this project, we have used Nokia 6610 color LCD with ATmega128. Nokia 6610 LCD is available having one of the two graphic drivers, either EPSON S1D15G00 or PHILIPS PCF 8833. Both types of LCDs can be controlled in a different manner. However, here, we have used Phillips PCF 8833 controller. There is a question always that how to find out the type of the controller inside the LCD. A solution to this is, we need to check the color of the flexible PCB coming out for external connection. If the color is orange/brown, the LCD has Phillips controller. If the color is Green, it has Epson controller [4].

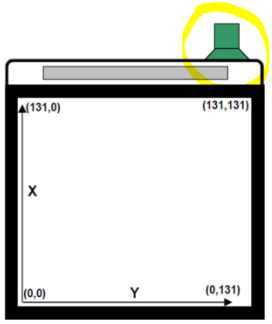


Fig. 4.3 How to identify type of controller [Front view of Nokia 6610]

4.3.1.1 LCD specifications

- 132 x 132 pixels
- 8 bit, 12 bit and 16 bit color rendition (4 bits red, 4-bits green, 4-bits blue)
- 3.3 volts
- 9-bit SPI serial interface (clock/data signals)

4.3.1.2 LCD Display Orientation

The Nokia 6610 display has 132 x 132 pixels; each one with 12-bit color (4 bits RED, 4 bits GREEN and 4 bits BLUE). Practically speaking, you cannot see the first and last row and columns. The normal orientation is shown in the figure given. That, of course, is upside-down on the application board as if the silk-screen lettering is used as the up/down reference. So we set the "mirror x" and "mirror y" command to rotate the display 180 degrees, as shown in figure given here.

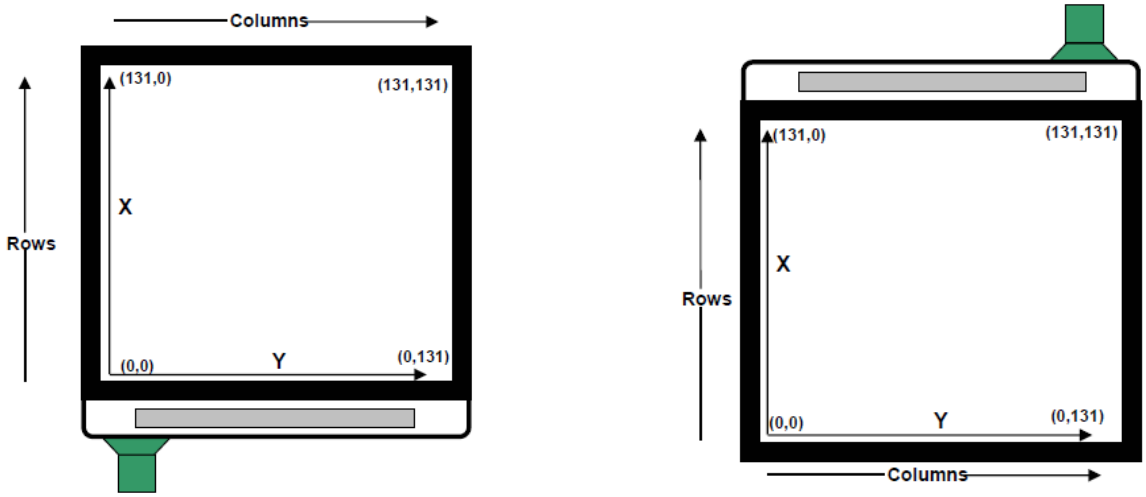


Fig 4.4 Orientation of Display

4.3.1.3 Communication with the Display

The Nokia 6610 uses a two-wire serial SPI interface (clock and data). It requires 9 bits to the display serially, the ninth bit indicates if a command byte or a data byte is being transmitted. The timing diagram of the communication is shown below:

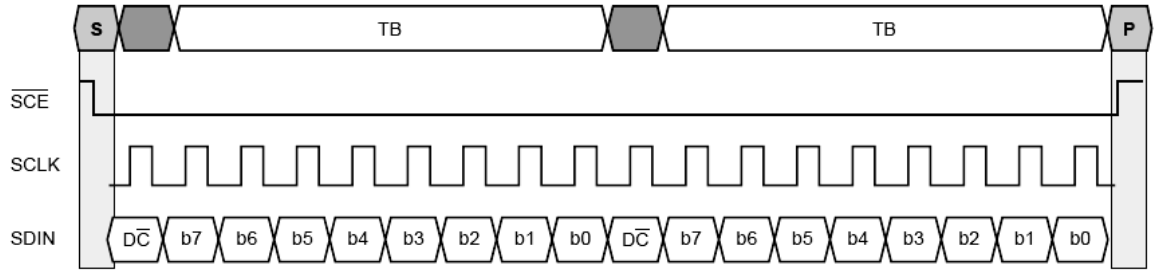


Fig. 4.5 Timing Diagram of 9-bit SPI interface

4.3.1.4 Software SPI

As mentioned above, the LCD requires 9 bits packet of SPI, and the microcontroller AtMega128 only supports 8 bits SPI communication. This prevents us to use ATmega128 hardware SPI pins. The only solution to this is, we need to define general purpose I/O pins as SPI by the software. By manually setting and resetting the defined pins, we can communicate in desired manner. The LCD driver has three functions supporting the SPI interface to the LCD:

- InitSpi ( ) - sets up the SPI interface to communicate with the LCD
- WriteSpiCommand (command) - sends a command byte to the LCD
- WriteSpiData (data) - sends a data byte to the LCD

For example:

- InitSpi ( ); // Initialize SPI interface to LCD
- WriteSpiCommand (SETCON); // Write contrast (command 0x25)
- WriteSpiData (0x30); // contrast 0x30 (range is -63 to +63)

4.3.1.5 Addressing Pixel Memory

The Philips PCF8833 controller has a 17424 word memory (132 x 132), where each word is 12 bits (4-bit color each for red, green and blue). You address it by specifying the address of the desired pixel with the Page Address Set command (rows) and the Column Address Set command (columns).

The Page Address Set and Column Address Set command specify two things, the starting pixel and the ending pixel. This has the effect of creating a drawing box. This sounds overly complex, but it has a wrap-around and auto-increment feature that greatly simplifies writing character fonts and filling rectangles. The pixel memory has 132 rows and 132 columns, as shown below in Figure:

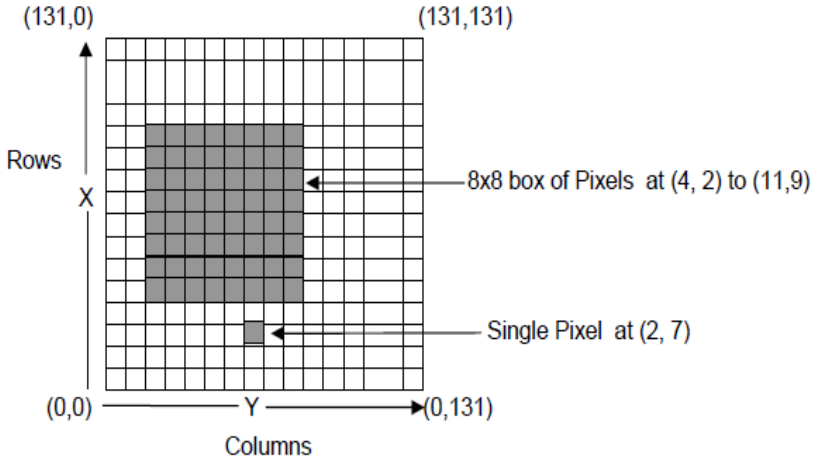


Fig 4.6 Addressing Pixel Memory

To address a single pixel, just specify the same location for the starting pixel and the ending pixel on each axis. For example, to specify a single pixel at (2, 7) use the following sequence.

- WriteSpiCommand(PASET); // Row address set (command 0x2B)
- WriteSpiData(2); // starting x address
- WriteSpiData(2); // ending x address (same as start)
- WriteSpiCommand(CASET); // Column address set (command 0x2A)
- WriteSpiData(7); // starting y address
- WriteSpiData(7); // ending y address (same as start)

To address a rectangular area of pixels, just specify the starting location and the ending location on each axis, as shown below. For example, to define a drawing rectangle from (4, 3) to (11, 9) use the following sequence.

- WriteSpiCommand (PASET); // Row address set (command 0x2B)
- WriteSpiData (4); // starting x address
- WriteSpiData (11); // ending x address
- WriteSpiCommand (CASET); // Column address set (command 0x2A)
- WriteSpiData (3); // starting y address
- WriteSpiData (9); // ending y address

Once the drawing boundaries have been established (either a single pixel or a rectangular group of pixels), any subsequent memory operations are confined to that boundary. For instance, if you try to write more pixels than defined by the boundaries, the extra pixels are discarded by the controller.

4.3.1.6 Color Formats

PCF 8833 supports 3 color formats: 8 bits, 12 bits, 16 bits.

1. 8 bits per pixel

   Selection of the reduced resolution 8 bits/pixel mode is accomplished by sending the Color Interface Pixel Format command (0x3A) followed by a single data byte containing the value This encoding requires a Memory Write command and a single subsequent data byte to specify a single pixel. The data byte contains all the color information for one pixel. The color information is encoded as 3 bits for red, 3 bits for green and 2 bits for blue, as shown in Figure.

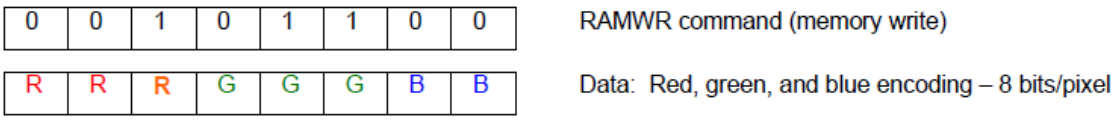


Fig 4.7 8 bit color mode

2. 12 bits per pixel (native mode)

   Selection of the native 12 bits/pixel mode is accomplished by sending the Color Interface Pixel Format command (0x3A) followed by a single data byte containing the value 3.

This encoding requires a Memory Write command and 1.5 subsequent data bytes to specify a single pixel. The bytes are packed so that two pixels will occupy three sequential bytes and the process repeats until the drawing boundaries are used up. Figure illustrates the 12 bits/pixel encoding.

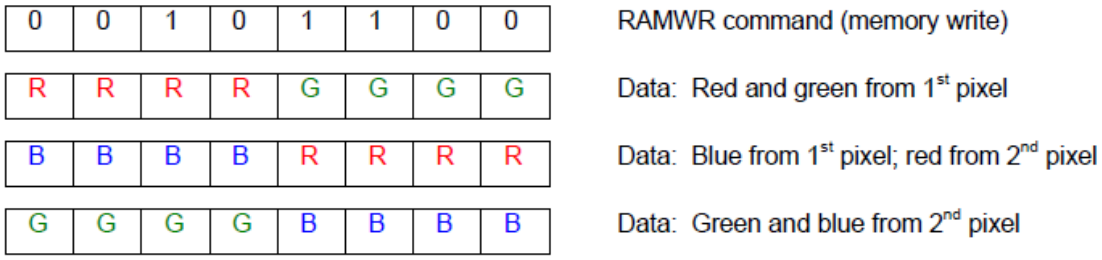


Fig 4.8 12 - bit color mode

The question might arise, "What happens if I specify a single pixel with just two data bytes. Will the 4-bits of red information from the next pixel (usually set to zero) perturb the neighboring pixel? The answer is no, since the PCF8833 controller writes to display RAM only when it gets a complete pixel. The straggler red bits from the next pixel wait for the completion of the remaining colors which will never come. Appearance of any command will cancel the previous memory operation and discard the unused pixel information. To be safe, I added a NOP command in the LCDSetPixel ( ) function to guarantee that the unused red information from the next pixel is discarded. Figure demonstrates how to send a single pixel using 12-bit encoding. Note that the last 4 red bits from the next pixel will be ignored.

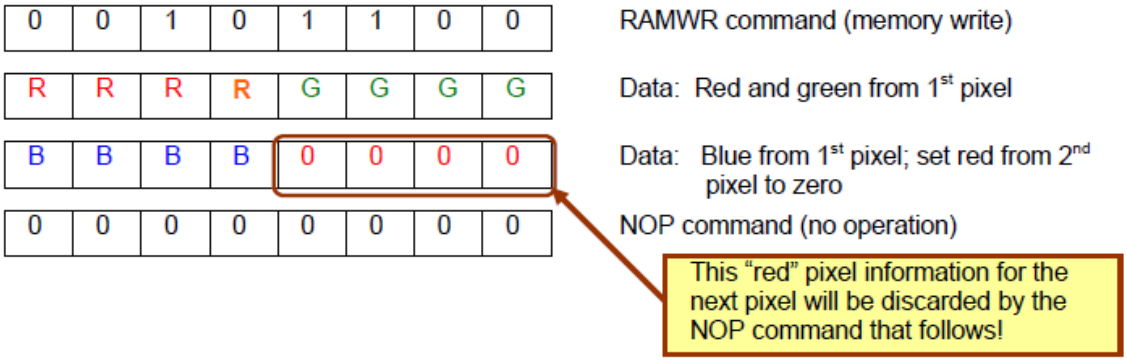


Fig. 4.9 Auto discard of unused pixel

3. 16 bits per pixel

   Selection of 16 bits/pixel mode is accomplished by sending the Color Interface Pixel Format command (0x3A) followed by a single data byte containing the value 5. This encoding requires a Memory Write command and two subsequent data bytes to specify a single pixel. The color information is encoded as 5 bits for RED, 6 bits for GREEN and 5 bits for BLUE, as shown in Figure below.

0	0	1	0	1	1	0	0	RAMWR command (memory write)
R	R	R	R	R	G	G	G	Data: Red (5 bits), Green (6 bits)
G	G	G	B	B	B	B	B	Data: Green (6 bits), Blue (5 bits)

Fig. 4.10 16 bit color mode

4.3.1.7 Wrap-Around and Auto Increment

The wrap-around feature is the cornerstone of the controller's design and it amazes me how many people ignored it in drawing rectangles and character fonts. This feature allows you to efficiently draw a character or fill a box with just a simple loop – taking advantage of the wrap-around after writing the pixel in the last column and auto-incrementing to the next row. To draw an 8 x 8 character font, define the drawing box as 8 x 8 and do a simple loop on 64 successive pixels. The row and column addresses will automatically increment and wrap back when you come to the end of a row, as shown in Figure below.

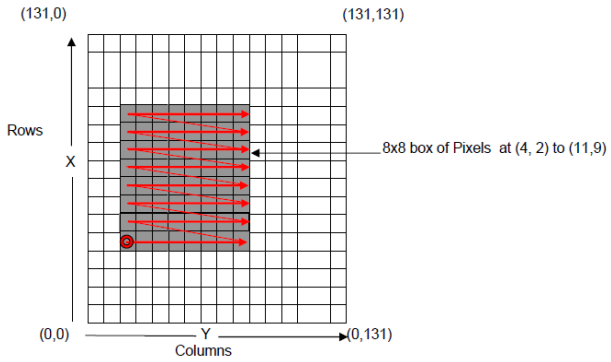


Fig 4.11 Wrap around Function

The rules for Auto-incrementing and Wrap-Around are:

- Set the column and row address to the bottom left of the drawing box.
- Set up a loop to do all the pixels in the box. Specifically, since three data bytes will specify the color for two pixels, the loop will typically iterate over ½ the total number of pixels in the box.
- Writing three memory bytes will illuminate two pixels (12-bit resolution). Each pixel written automatically advances the column address. When the "max" column address pixel is done, the column address wraps back to the column starting address AND the row address increments by one. Now keep writing memory bytes until the next row is illuminated and so on.

4.4 Algorithm for LCD Display (Philips PCF8833)

Philips PCF8833 does not quite boot into a "ready to display" mode after hardware reset. The following is the minimal commands/data needed to place it into 8-bit color mode.

1. First, we do a hardware reset with a simple manipulation of the port pin. Reset is asserted low on this controller.

```
// Hardware reset
LCD_RESET_LOW;
Delay (20000);
LCD_RESET_HIGH;
Delay (20000);
```

2. The controller boots into SLEEPIN mode, which keeps the booster circuits off. We need to exit sleep mode which will also turn on all the voltage booster circuits.

```
// Sleep out (command 0x11)
WriteSpiCommand (SLEEPOUT);
```

3. We need to invert the picture as it is upside down in the device.

```
// Inversion on (command 0x20)
WriteSpiCommand (INVON); // seems to be required for this controller
```

4. For this driver, we selected to use the 8-bit color pixel format exclusively.

```
// Color Interface Pixel Format (command 0x3A)
WriteSpiCommand (COLMOD);
WriteSpiData (0x02); // 0x02 = 8 bits-per-pixel
```

5. In setting up the memory access controller, we selected to use the “mirror x” and “mirror y” commands to reorient the x and y axes to agree with the silk screen lettering on the application board. If you want the default orientation, send the data byte 0x08 instead. Finally, we had to reverse the RGB color setting to get the color information to work properly.

```
// Memory access controller (command 0x36).
WriteSpiCommand (MADCTL);
WriteSpiData (0xC8); // 0xC0 = mirror x and y, reverse rgb
```

6. We found that setting the contrast varies from display to display. It can be checked by the following sequence of commands for various values of contrast

```
// Write contrast (command 0x25)
WriteSpiCommand (SETCON);
WriteSpiData (0x30); // contrast 0x30
Delay (2000);
```

7. Now that the display is initialized properly, we can turn on the display and we’re ready to start producing characters and graphics.

```
// Display On (command 0x29)
WriteSpiCommand (DISPON);
//Send the image data and See!!!
```

Chapter 5

Interfacing Touch Screen

Interfacing Touch Screen

This Chapter focuses on the touch screen technology and detection methods for 4- wire touch screen. Two basic approaches are explained- one implements a data- averaging calculation on Analog to Digital Converter (ADC) inputs without using any controller while the other uses a dedicated touch screen controller ADS7843 for either 8- bit or 12- bit conversion.

5.1 Introduction to Touch Screen

A touch screen is a display which can detect the presence and location of a touch within the display area. The term generally refers to touch or contact to the display of the device by a finger or hand. Touch screens can also sense other passive objects, such as a stylus. However, if the object sensed is active, as with a light pen, the term touch screen is generally not applicable. The ability to interact directly with a display typically indicates the presence of a touch screen.

Many of the modern portable devices come with a user touch interface. The technology is often referred to as “Touch control”. This is being implemented using ‘touch screen’, a thin transparent film used as input for the control signals, providing the device an elegant look and an easy user interface.

The touch screen has two main attributes. First, it enables one to interact with what is displayed directly on the screen, where it is displayed, rather than indirectly with a mouse or touchpad. Secondly, it lets one do so without requiring any intermediate device, again, such as a stylus that needs to be held in the hand. Such displays can be attached to computers or, as terminals, to networks. They also play a prominent role in the design of digital appliances such as the personal digital assistant (PDA), satellite navigation devices, mobile phones, and video games.

5.1.1 Types of Touch Screen

Depending on the technology used in its manufacturing the various types of touch screens are as follows:

- Resistive
- Surface acoustic wave
- Capacitive
- Infrared
- Strain gauge
- Optical imaging
- Dispersive signal technology
- Acoustic pulse recognition

Here a 4- wire resistive touch screen has been used.

5.1.2 Technology used in a Touch Screen

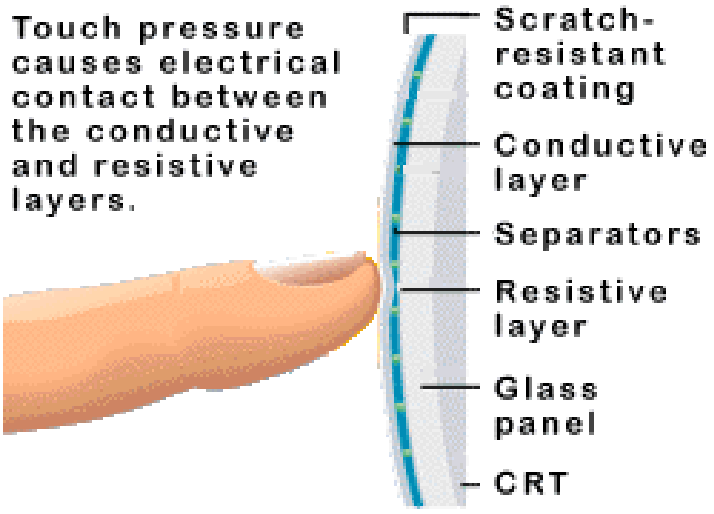


Fig 5.1 Touch Screen Technology

Resistive screens are generally the most affordable type of touch screen, which explains their success in high-use applications like PDAs and Internet appliances. Although clarity (75-80%) is

not as good as with other touch-screen types, resistive screens are very durable. Keeping cost affordability and availability in mind, Resistive Touch screen was selected for this project.

Resistive touch technology consists of a glass or acrylic panel that is coated with electrically conductive and resistive layers. The thin layers are separated by invisible separator dots. When operating, an electrical current moves through the screen. When pressure is applied to the screen the layers are pressed together, causing a change in the electrical current and a touch event to be registered. Resistive Touch Screens are further classified into: 4 wire touch screens and 8 wire touch screens. They are characterized from the number of lines used for sensing the information. The clarity and efficiency increases with the number of lines used for getting information.

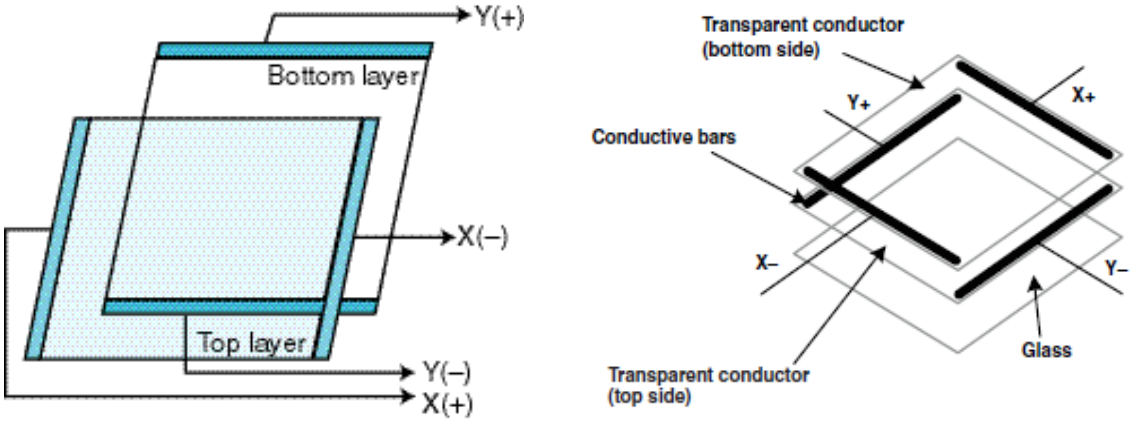


Fig 5.2 4-Wire Touch Screen Configuration

4 wire touch screens, cheapest of all, use only 4 wires: X+, X-, Y+ and Y- for analog inputs. During the initial stages of this project, the co-ordinates of a touch were detected and calculated using ADC, without using a touch controller. Based on voltage distribution all along the thin plates, touch changes the resistance of the material which is fed to the Analog to Digital Converter of the target controller and using a mathematical formula of voltage divider circuit, the co-ordinates are calculated. This process involves manual programming and is less precise. However, later, the touch screen was interfaced using a touch screen driver controller ADS7843 as explained below.

5.2 Detecting a Touch

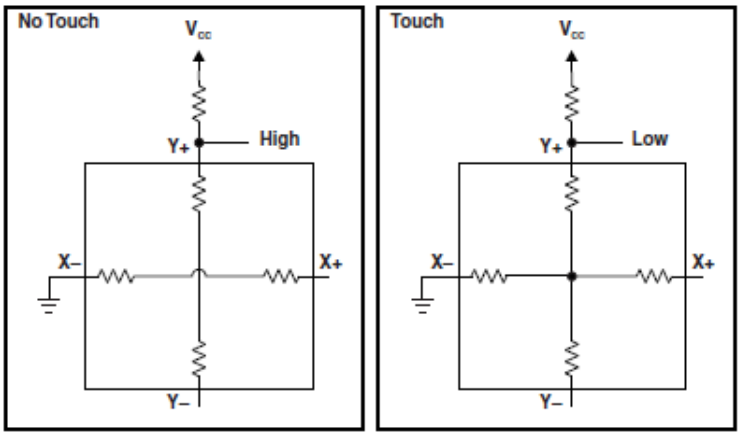


Fig 5.3 Touch Detection in a 4 Wire Touch Screen

To know if the coordinate readings are valid, there must be a way to detect whether the screen is being touched or not. This can be done by applying a positive voltage (VCC) to Y+ through a pull-up resistor and applying ground to X–. The pull-up resistor must be significantly larger than the total resistance of the touch screen, which is usually a few hundred ohms. When there is no touch, Y+ is pulled up to the positive voltage. When there is a touch, Y+ is pulled down to ground as shown in Figure 5.3. This voltage-level change is used to generate a pin-change interrupt.

5.3 Reading a 4-wire Touch Screen

The x- and y- coordinates of a touch on a 4-wire touch screen can be read in two steps. First, Y+ is driven high, Y– is driven to ground, and the voltage at X+ is measured. The ratio of this measured voltage to the drive voltage applied is equal to the ratio of the y coordinate to the height of the touch screen.

The y coordinate can be calculated as shown in Figure below. The x coordinate can be similarly obtained by driving X+ high, driving X– to ground, and measuring the voltage at Y+. The ratio of this measured voltage to the drive voltage applied is equal to the ratio of the x coordinate to the width of the touch screen. This measurement scheme is shown in Figure below.

$Y = V_x/V_{drive} * Height$ $X = V_y/V_{drive} * Width$

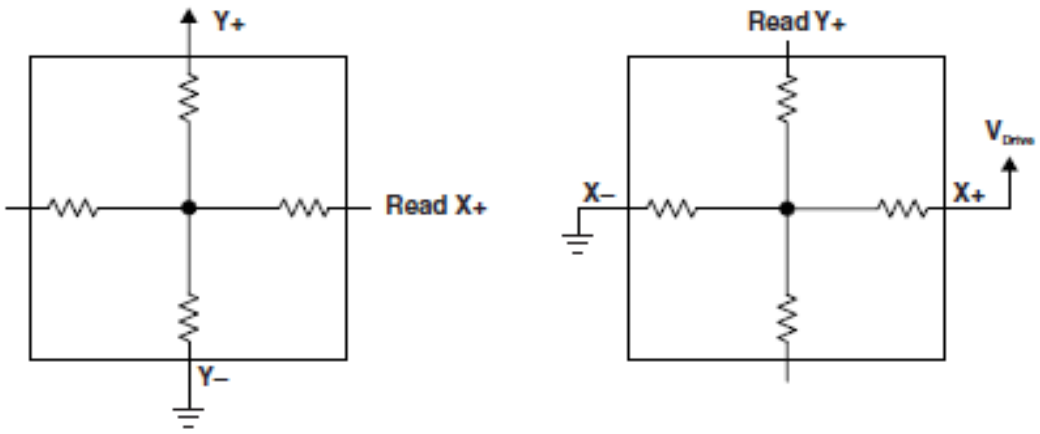


Fig 5.4 Reading a 4- Wire Touch Screen

5.4 Data Average Algorithm using ADC

This approach for reading a 4 wire touch screen does not use any controller chip to calculate the co- ordinates of the detected touch. Instead it uses a crude method of polling the four values X+, Y+, X- and Y- and averaging their values to measure the result [5].

Measurements are made using a 10-bit ADC. A 10-bit ADC can resolve 2-to-the-10th power or 1024 different input values in each the horizontal and vertical direction. The four-wire system resolution is, however, less than 1024 due to losses in the drive voltage that occur before it reaches the touch screen ITO. Touch point coordinates are reported to the microcontroller through a general purpose port, configured as an ADC port.

Since several measurements for one coordinate pair are being taken, the designer has the opportunity to do some processing on this data, like averaging. This will help prevent spurious readings that may make dealing with the human interface difficult.

The averaging algorithm reduces noise resulting from contact bounce during use of the touch screen. Successive X and Y samples are tested to determine if their values differ by no more than a certain range. If one or more samples fall outside this range, the samples are discarded and the

process is restarted. This is continued until successive X samples (then Y samples) fall within the range. The average of these values is used as the X and Y coordinates, respectively. Once independent X and Y samples are obtained, coordinate pairs are sampled to eliminate the effects of noise. If a sample does not fall within an internal range, all X and Y coordinate pairs are discarded and the independent X and Y sequence is restarted. Once acceptable coordinate pairs have been obtained, an average coordinate pair is determined.

5.5 Touch Detection using Touch Screen Controller

This approach for detecting a touch and calculating its co- ordinates on a screen uses a touch screen controller. For this project the controller ADS7843 manufactured by Texas Instruments is used.

The ADS7843 is a 12-bit sampling Analog-to-Digital Converter (ADC) with a synchronous serial interface and low on resistance switches for driving touch screens [6].

For its basic operation the device requires an external reference and an external clock. The analog input to the converter is provided via a four channel multiplexer. A unique configuration of low on-resistance switches allows an unselected ADC input channel to provide power and an accompanying pin to provide ground for an external device. By maintaining a differential input to the converter and differential reference architecture, it is possible to negate the switch's on-resistance error (should this be a source of error for the particular measurement).

When the converter enters the hold mode, the voltage difference between the +IN and –IN inputs is captured on the internal capacitor array. During the sample period, the source must charge the internal sampling capacitor. After the capacitor has been fully charged, there is no further input current. The rate of charge transfer from the analog source to the converter is a function of conversion rate.

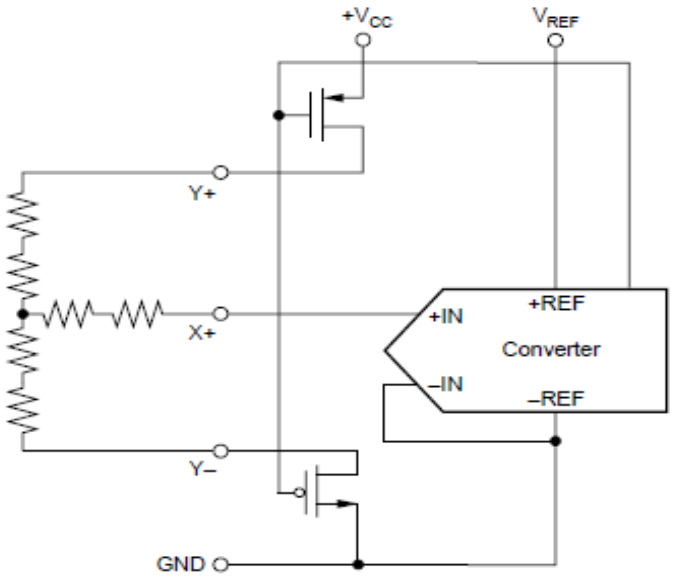


Fig 5.5 Simplified Diagram of Differential Reference

The voltage difference between +REF and –REF (shown in the Figure 5.4) sets the analog input range. As the reference voltage is reduced, the analog voltage weight of each digital output code is also reduced. As the current from the reference is drawn on each bit decision, clocking the converter more quickly during a given conversion period will not reduce overall current drain from the reference. There is also a critical item regarding the reference when making measurements where the switch drivers are on. For this discussion, it's useful to consider the basic operation of the ADS7843 as shown in Figure above. This particular application shows the device being used to digitize a resistive touch screen. A measurement of the current Y position of the pointing device is made by connecting the X+ input to the ADC, turning on the Y+ and Y– drivers, and digitizing the voltage on X+. For this measurement, the resistance in the X+ lead does not affect the conversion (it does affect the settling time, but the resistance is usually small enough that this is not a concern).

However, since the resistance between Y+ and Y– is fairly low, the on-resistance of the Y drivers does make a small difference. Under the situation outlined so far, it would not be possible to achieve a 0V input or a full-scale input regardless of where the pointing device is on the touch screen because some voltage is lost across the internal switches. In addition, the internal switch resistance is unlikely to track the resistance of the touch screen, providing an additional source of

error. This situation can be remedied by setting the SER/DFR bit LOW, the +REF and –REF inputs are connected directly to Y+ and Y–. This makes the A/D conversion ratiometric. The result of the conversion is always a percentage of the external resistance, regardless of how it changes in relation to the on-resistance of the internal switches. Note that there is an important consideration regarding power dissipation when using the ratiometric mode of operation, see the Power Dissipation section for more details. As a final note about the differential reference mode, it must be used with +VCC as the source of the +REF voltage and cannot be used with VREF. It is possible to use a high precision reference on VREF and single-ended reference mode for measurements which do not need to be ratiometric. Or, in some cases, it could be possible to power the converter directly from a precision reference. Most references can provide enough power for the ADS7843, but they might not be able to supply enough current for the external load (such as a resistive touch screen).

The “Control Byte” for the touch controller is defined as follows:

The first bit, the ‘S’ bit, must always be HIGH and indicates the start of the control byte. The ADS7843 will ignore inputs on the DIN pin until the start bit is detected. The next three bits (A2-A0) select the active input channel or channels of the input multiplexer. The MODE bit determines the number of bits for each conversion, either 12 bits (LOW) or 8 bits (HIGH). The SER/DFR bit controls the reference mode: either singleended (HIGH) or differential (LOW). (The differential mode is also referred to as the ratiometric conversion mode.) In singleended mode, the converter’s reference voltage is always the difference between the VREF and GND pins. In differential mode, the reference voltage is the difference between the currently enabled switches. The last two bits (PD1-PD0) select the power-down mode. If both inputs are HIGH, the device is always powered up. If both inputs are LOW, the device enters a power-down mode between conversions. When a new conversion is initiated, the device will resume normal operation instantly—no delay isneeded to allow the device to power up and the very first C onversion will be valid. There are two power-down modes: one where PENIRQ is disabled and one where it is enabled.

The ADS7843 provides an 8-bit conversion mode that can be used when faster throughput is needed and the digital result is not as critical. By switching to the 8-bit mode, a conversion is

complete four clock cycles earlier. This could be used in conjunction with serial interfaces that provide 12-bit transfers or two conversions could be accomplished with three 8-bit transfers. Not only does this shorten each conversion by four bits (25% faster throughput), but each conversion can actually occur at a faster clock rate. This is because the internal settling time of the ADS7843 is not as critical—settling to better than 8 bits is all that is needed. The clock rate can be as much as 50% faster. The faster clock rate and fewer clock cycles combine to provide a 2x increase in conversion rate.

Chapter 6

Memory Interface

Memory Interface

The aimed multimedia device has to be portable and portability imposes constraints on type of memory to be interfaced. Hence, low power, compact size and economical memory is preferable to be interfaced. The obvious solution is Flash Memory with additional advantage of High Data Transfer Speed. This Chapter is dedicated to interfacing of a specific type of Flash Memory card (MultiMedia Card) with ATmega128.

6.1 Flash Memory

Flash is a specific type of EEPROM (Electrical Erasable Programmable Read Only Memory). Cost/bit is more than Disk Drives but they are extremely light weight and operate on low power. In flash memory data bits can be read/written into blocks in random access fasion, hence faster data transfer can be achieved. The disadvantage is it can be read or written bit by bit but minimum unit for erasure is a block. Also the integrity of storage is not promised after 1,00,000 write –erasure.

For embedded application, best suited memory is available as Memory cards. Memory Card is flash memory with inbuilt controller in it. Of commercially available Memory Cards such as Compact Flash, Secure Digital, MultiMedia Card etc, SD/MMC are most popular one for embedded devices.

6.2 Multimedia Card

Though SD and MMC are developed by two different organizations - SDCA (Secure Digital Card Association) and MMCA (MultiMedia Card Association), SDC are backward compatible with their predecessor MMC. Here, we have developed the software for MMC, but with a few initialization changes, SD card can also be used. A minimized derivative of MMC is Reduced Sized MMC. The description below holds true for both of them.

6.2.1 Features of Multimedia Card

1. **Flash Independent Technology –** Memory of MMC is partitioned as 512 byte sector. To read or write a sector, the host computer simply issues a command specifying the address to read or write to. The controller on MMC takes care of the command issued. The main advantage of this is that, the host software does not need to get involved with the details of how the read/write/erase operations occur. As the Flash Technology is expected to evolve day by day, this is very important. No matter how complex the flash technology becomes, the host software will remain the same.
2. **Defect and Error Management –** MMC do a read after write to verify that the written data is correct. If a bit is found incorrect, it is replaced. If required an entire sector can also be replaced. Hence, card's error rate is very low.
3. **Automatic Sleep Mode –** Upon completion of an operation, the card enters sleep mode if no command is issued within 5ms. The host does not have to take any steps for this.
4. **Hot Insertion –** For hot insertion support, the connector providers manufacture the connectors such that the power pins are normally longer than the data pins.

6.3 Interfacing MMC with Microcontroller

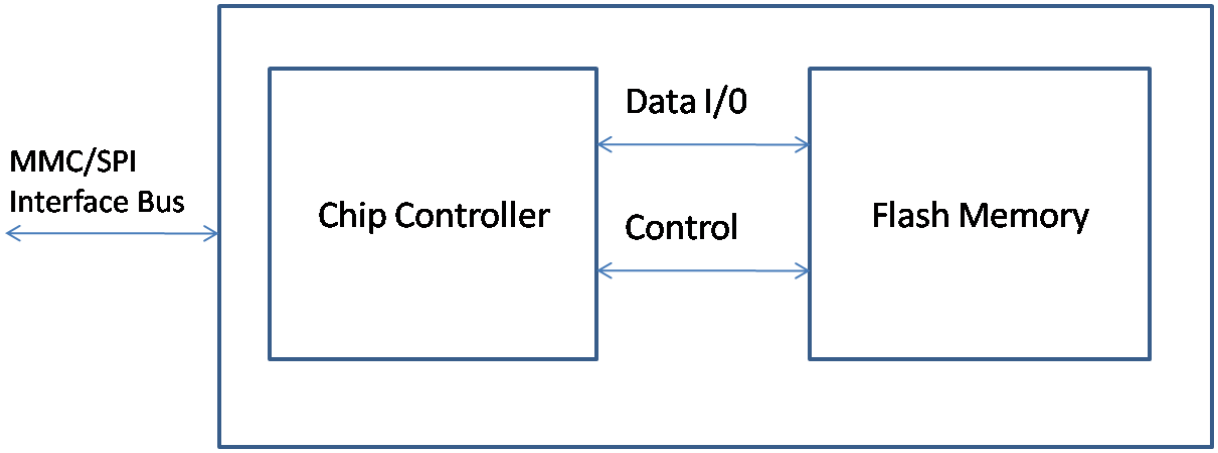


Fig 6.1 MMC card – Flash memory with controller

Multimedia Card is a System made up of Flash Memory as well a chip controller. The Chip controller is responsible for MMC's functionality. The main functionality is to handle communication with Host. Communication with MMC card can be carried out by any of the two modes,

1) MMC Mode – Multimedia Card Mode is the native mode of communication. On system reset , the card is always in MMC mode. This communication protocol is described by Multimedia Card Association and is documented under the title "*the MultiMedia Card Standard Specifications v3.3"*

2) SPI Mode – SPI (Serial Peripheral Interface) is the secondary Communication protocol for MultiMedia Card. It is a subset of MultiMedia Card Mode. SPI mode is preferable for embedded applications. Here, we have carried out the communication using SPI mode.

6.3.1 SPI mode communication with MMC

1. Setting up SPI for Communication

To initiate the communication using SPI, first of all we need to configure the Master Device. In our application, our MCU, ATmega128 is the Master, where as MMC Controller is the slave. The sequence of steps to be taken to Initialize SPI is as follows.

1. Enable SPI – Done by setting SPE bit in SPCR.
2. Make ATmega128 Master – Done by setting MSTR bit in SPCR.
3. Set the SPI frequency .

   Here, the frequency of SPI communication should not exceed the frequency supported by the slave. Our slave MMC can operate on highest 20MHz frequency. To be in safe limits, we have kept the SPI frequency 8MHz. This is done by resetting bit couple SPR0 and SPR1 (fclk/4) and setting bit SPI2X of SPSR (Double the SPI frequency).
4. Select SPI mode – Select Mode 0 by setting CPHA=0,CPOL=0 (Mode 3 also works).
5. Select Data Order – Kept MSB first by resetting the DORD bit.

Hence SPCR needs to be loaded with binary values 01010000b or 01011100b [0x50 or 0x5C]. And SPSR is kept 0x01.

2. Data Transmission
    1. Select the slave by lowering the Chip Select line
    2. Move the data byte to SPDR [The shift operation of SPI will shift the data into slave buffer] [Refer chap on spi]
    3. Wait for SPIF flag bit to be set

3. Data Reception
    1. Select the slave by lowering the Chip Select
    2. Move a dummy byte to SPDR [For MMC we've kept 0xFF]
    3. Wait for SPIF flag bit to be set
    4. Move SPDR to buffer to store received byte

6.3.2 Operations on MMC

With functions at hand to handle SPI communication, we can write routines to carry out different MMC operations. All the communication with MMC and Host is done in CMD/Response format i.e. the Host sends a particular command for a particular operation. On receiving the command the MMC controller will either report an error in the command or will perform the operation and send a valid response. The Command/ Response format is described in the following figure:

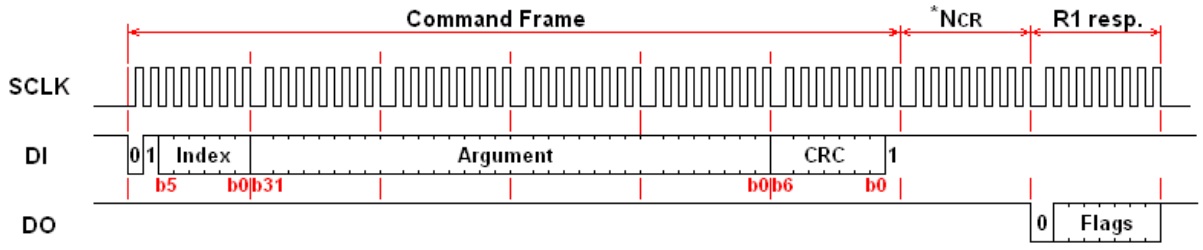


Fig 6.2 Command/Response Communication over SPI

In SPI mode, the data direction on the signal line is fixed and the data is transferred in byte oriented serial communication. The command frame from host to card is a fixed length (six bytes) packet as shown below.

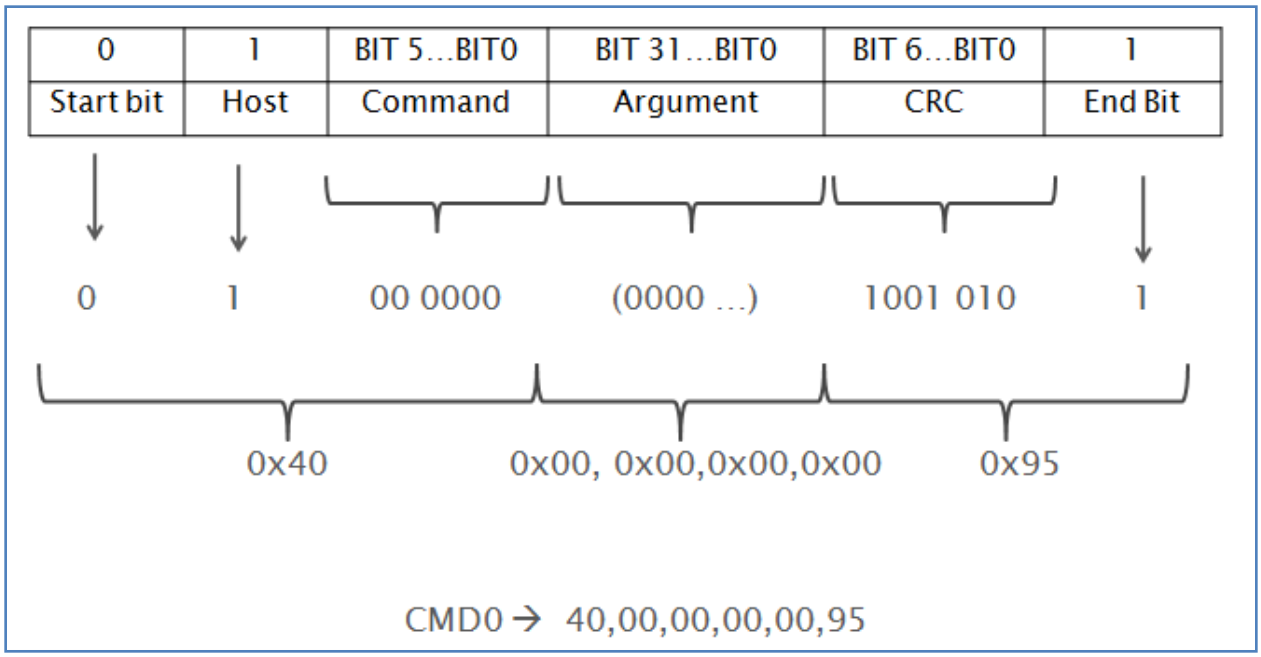


Fig 6.3 Command Format Example for Command 0

When a command frame is transmitted to the card, a response to the command (R1, R2 or R3) will be sent back to the host. Most of the commands are responded by R1. R2 is used only for SD cards. The Response bit fields and their meanings are depicted in the figure below,

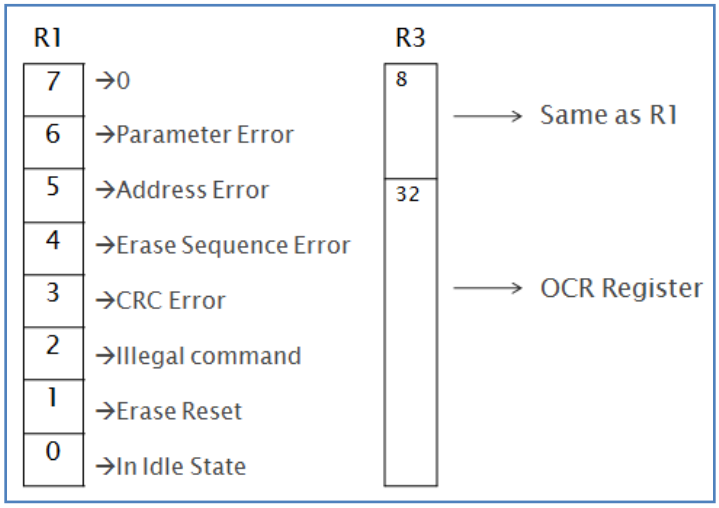


Fig 6.4 Response Format [R1 and R3]

Because data transfer is driven by serial clock generated by host, the host must continue to read bytes until receive any valid response. The command response time (NCR) is, 1 to 8 bytes for MMC. The CS signal must be held low during a transaction (command, response and data transfer if exist). The CRC field is optional in SPI mode, but it is required as a bit field to compose a command frame. The DI signal must be kept high during read transfer.

Here is a sequence of commands that we need to issue for various MMC operations [7]-

A. **Initialization:** Upon power up, MMC cards need to be instructed to change to SPI from their default operating mode. The sequence to do so is as follows:
   1. After power on, wait at least a millisecond and set CS and D-in High.
   2. Send 80 clock pulses.
   3. Set CS low and send a "CMD0" (40h,00h,00h,00h,00h,95h*) to reset the card. (The card checks the CS line when CMD0 is received and goes into SPI mode if CS is low.)
   4. When CMD0 is accepted, the card enters idle state and so responds with "01h".
   5. Repeatedly send CMD1 (41h,00h,00h,00h,00h,00h) and check the response.
   6. When response is 00h, the card is ready (this can take hundreds of milliseconds).

* Note the command CRC value is not 00h here as the card won't be in SPI mode yet and actually requires a correct CRC code. "95h" is the "hardwired" value, valid only for CMD0.

B. **Reading a sector:** The command to read single sector is CMD17. The argument is the BYTE address of the sector to read (so set it at sector number * 512). The CRC is 00h as usual.

   1. Send: CMD17 (51h,00h,24h,68h,00h,00h- address, null CRC)
   2. Read: xx - NCR Time
   3. Read: xx - Command Response (should be 00h)
   4. Read: until FEh is received - Wait for Data token (see note 1)
   5. Read: yy * 512 - Get 512 bytes from sector
   6. Read: zz - Read CRC lo byte
   7. Read: zz - Read CRC hi byte

*Note : In a simple implementation, we can simply wait for the data response "FEh" in a loop with a time-out and report a general error if it isn't received. What actually happens though is this: If the card is unable to send the requested data, an error token will be returned instead of the data token. This is a single byte with the three MSBs set to zero.

C. **Writing a sector:** The command to write a single sector is CMD24. The argument is the same as for reading a sector and the CRC is 00h as normal. Following the command response, a write delay byte is required (send an FFh), then the data token (FEh) should be sent. Next, the data to fill the sector can be sent with a 2 byte CRC following on (i.e. send two zeros). After the last CRC byte is received the card will respond with a data response byte with bits in the format "xxx0sss1" Here "xxx" aren't used, bit 4 is zero, bits 3:1 (sss) hold the status code and bit 0 is a one.

Status codes: 010 = Data accepted, 101 = Data rejected due to CRC error and 110 = Data rejected due to write error.

After the data response is received, the card will start programming the data it buffered into the card. We must now wait for the busy signal to clear (by reading byte for a non-zero byte to be returned) before carrying out further operations. Finally, it’s advisable to check the card's status bytes to check the sector was actually programmed correctly.

1. Send: CMD24 (58h,00h,24h,68h,00h,00h- address, null CRC)
2. Read: xx - NCR Time
3. Read: xx - Command Response - should be 00h
4. Send: FFh - One byte gap
5. Send: FEh - Send Data token
6. Send: yy * 512 - Send bytes for sector
7. Send: zz - Send (null) CRC lo byte
8. Send: zz - Send (null) CRC hi byte
9. Read: vv - Read packet response*
10. Read: until value is NOT 00h - Read busy status, wait till done

*Packet Response: If (value AND 1Fh) >> 1 = 02h, packet received OK

In a similar manner multiple blocks can also be read from and written into using various commands. The table below gives a compiled list of commands, their arguments, the Hex sequence for the command and expected response,

CMD	ARGUMENTS	RESPONSE	Data	Description	HEX VALUE
0	-	R1	No	Reset	40,00,00,00,00,95
1	-	R1	No	Initialize	41,00,00,00,00,00
17	SEC_N0*512	R1	Yes	Read single sector	51,xx,xx,xx,xx,00
24	SEC_N0*512	R1	Yes	Write single sector	58,xx,xx,xx,xx,00
58	-	R3	No	Read OCR	7A,00,00,00,00,00

Table 6.1 Important Commands

6.4 FAT File System

Till now we have seen how to read data by indexing sector numbers. In practice user does not store the data to "particular" sector of MMC. The Data stored on MMC card is in form of Files written to them by a computer system. To create, read, manipulate files there is a need of a file system. In computing, a file system (often also written as filesystem) is a method for storing and organizing computer files and the data they contain to make it easy to find and access them.

Most file systems make use of an underlying data storage device that offers access to an array of fixed-size blocks, sometimes called sectors, generally a power of 2 in size (512 bytes or 1, 2, or 4 KiB are most common). The file system software is responsible for organizing these sectors into files and directories, and keeping track of which sectors belong to which file and which are not being used. Such a track is kept by creating and updating tables known as File Allocation Tables. Among various types of filesystems like Disk FS, Network FS and Special Purpose FS, to read

MMC/SD we need to understand Disk FS. Despite being Flash memory they are structured into sectors so as to make them analogous to diskettes and hard drives for an operating system. So, most of these MMC/SD cards are formatted to have FAT filesystem. FAT Fs was developed by microsoft but it is used by almost all operating systems in the market [8].

To access the files stored on MMC we would need to write a program which can read the FAT table on the MMC, get the File id, which sectors contain the data that belongs to the card. These sectors may not be contiguous. In that case their order. Also the file's structure itself needs to be read to know file size, attributes, permissions etc. When we started to implement this we realized it was a challenging as well as time consuming task.

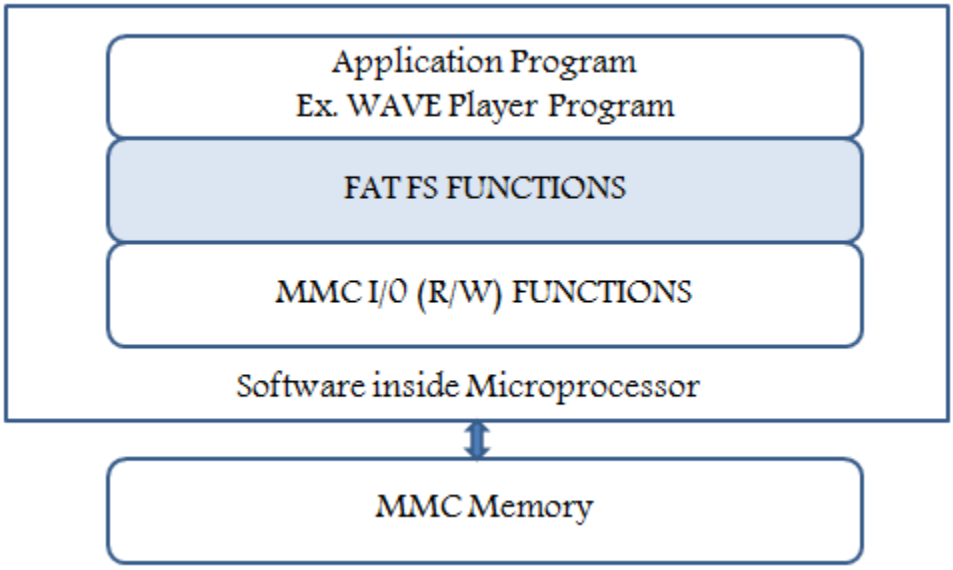


Fig 6.5 Hierarchy of programs to access files on MMC

Fortunately, we found a FatFs library module written in ANSI C, which was platform independent (unaware of Hardware used). The only critical task left was to write MMC read/write routines in such a manner expected by the file systems. That is taking care of the arguments and the return values of the MMC functions. The figure given above depicts how the application program can access the memory through a hierarchy of programs.

Chapter 7

WAVE Playback

WAVE Player

The aimed multimedia device has the capability of playing audio media. The most popular audio format is .mp3 (MPEG Audio Layer- 3). But the ATmega128 lacks the processing power to perform software decoding of .mp3 files. Hence mp3 decoding is taken care of by a dedicated decoder chip STA013. Considering the capability of ATmega128 software decoding of 8 bit PCM uncompressed WAVE was successfully tried and performed.

7.1 Introduction to WAVE Playback

A waveform audio file, also known as a *WAVE* file, or simply *.wav* after its extension, is a common type of sound file. The wav format is based on the *Resource Interchange* File *Format (RIFF)*, which stores audio files in indexed "chunks" and "sub-chunks". It could digitize sounds 100% faithful to the original source because it is a *lossless* format. "Lossless" means that the wav file format does not compromise audio quality even when it holds compressed data. It is the main format used on Windows systems for raw and typically uncompressed audio. The usual bit stream encoding is the Pulse Code Modulation (PCM) format.

The ATmega128 has four timers that can be used in Timer/ Counter or Pulse Width Modulation (PWM) modes. Using an 8- bit/ 16- bit timer in the Fast PWM mode a Digital to Analog Converter can actually be realized to play 8- bit uncompressed Audio files. This arrangement gives a good quality of sound with lesser components without using any external Digital to Analog Converters.

It is necessary to understand how PWM is used as Digital to Analog converter prior to moving to the configuration and algorithm part.

The idea behind this is to use a pulse-width modulation (PWM) digital output signal, and average it with a low-pass filter to create the analog output signal. In comparison to using a true DAC, this method generally produces a less accurate, less precise signal but on the other hand this method is easier and faster to implement and it costs almost nothing.

Here there are two approaches. We can either use two timers: one for generating PWM and the other for generating an interrupt according to the sampling rate of the Audio File (around 125 us for 8 KHz and 90 us for 11.025 KHz) or only a single timer is used for the fast PWM generation and the delay of 125 us is generated using the in-built AVR library (util/ delay.h) function _delay_ms. Also while using the timer 1 in the Fast PWM mode prescalar value was kept for dividing the frequency as "1" in order to get the maximum precision.

7.2 Pulse Width Modulation in ATmega128

First of all, the counter is just a variable-- a variable in a hardware register-- that indicates how many times some event has happened. While the counter can be used to count something asynchronous, like widgets passing a point on a conveyor belt, it can also be used to count a periodic clock signal, in which case the counter could be called a timer. The fastest signal that can be counted is the internal system clock. Our chip has an internal RC clock oscillator, with a maximum rate of 16 MHz [9].

Suppose that our counter variable starts out at zero. With each incoming clock tick, the counter variable is incremented, and after one second has reached a value of about sixteen million. Well, it would, except that we're using a 16-bit timer which can only count up to 65535, which it reaches in about 4 ms. When the counter reaches its upper limit, we reset (roll over) the value to zero and keep counting. At 4 ms per cycle, the counter can go through its full counting cycle at a rate of about 244 Hz. To make the counter do something interesting, we use an additional variable, stored in the compare register that tells the timer when to change its output. When the counter starts out at zero, its physical output pin outputs a logical low level (zero volts). When the counter variable is equal to the value in the compare register, the output pin is switched to be high (e.g., 5 V), and when the counter resets at its upper limit, the output pin goes low again. To make things even more interesting, we can also change that upper limit to some number lower than 65535, which changes the overall frequency at which the sequence repeats.

Below is an example of generating PWM signals with 8 MHz Clock:

Suppose that we're counting an 8 MHz clock, the upper counting limit is 65535, and the compare register is set to 32767. Then, during each cycle, the output pin will be low until the counter gets to 32767, and then high until it gets to 65535 and resets. The output waveform is then a square wave with 50 percent duty cycle (on one half of the time) and frequency near 122 Hz. Next, we might change the compare register value to 45000. This results in the output being on about 30 percent of the time, with the same 122 Hz frequency. We might picture the PWM generation like this:

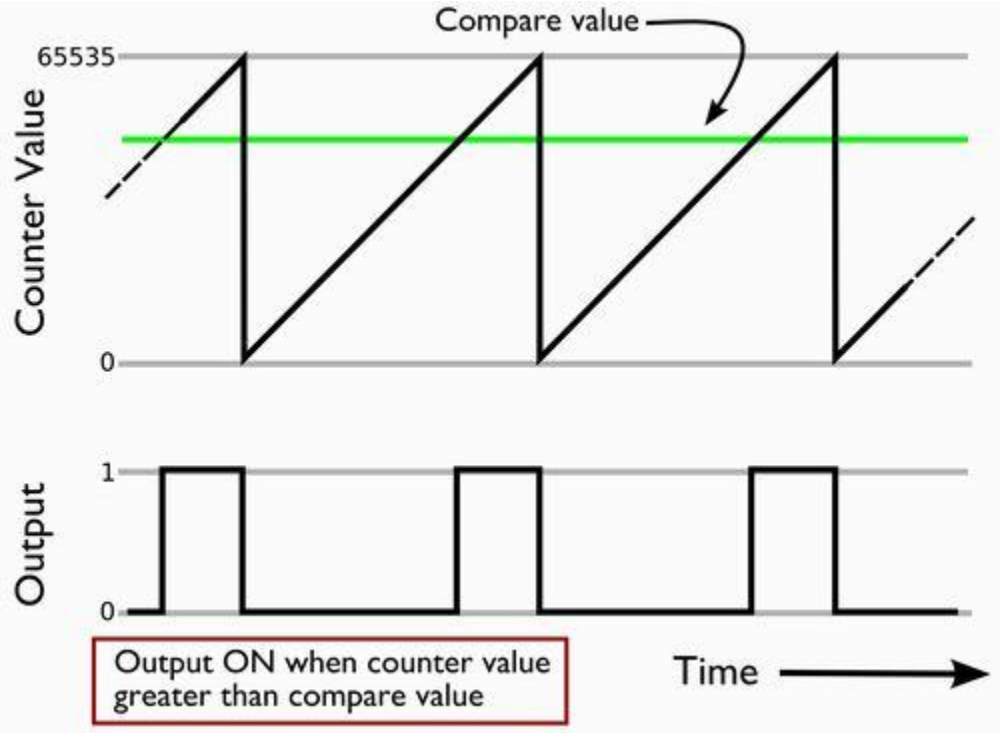


Fig 7.1 Pulse Width Modulation

Again, the counter goes from 0 to 65535, counting at 8 MHz, and resets when it reaches the upper limit-- at a rate of 122 Hz. When the counter passes the value in the compare register (45000), the output turns high. The output goes low again upon reset. The resulting output waveform has about a 30 percent duty cycle.

If the upper counting limit is then changed from 65535 to 32767, the counter goes through a complete cycle in 4 ms, so the clock frequency is about 244 Hz. Since the compare register is

again set to one half of the upper counting limit, the duty cycle is again about 50 percent. The average value of the output signal can be varied from zero to 100 percent of the logical output level, and that change is made by varying the time width of the regular output pulses, hence the name "pulse width modulation."

To make the chip actually do these things the following needs to be done:

(1) setting the clock source and divider, (2) setting the upper counting limit, (3) setting an initial value for the compare register (equal to the upper counting limit), and (4) setting the timer control registers as to make it operate in the Fast PWM mode.

As discussed earlier, the system clock is run off of the internal RC oscillator at its full rate of 16 MHz. It's possible to run at a lower rate by using a clock prescaling divider, but doing that makes PWM less precise.

Next, Timer 1 is configured by setting its bits as shown below: (Refer to Chapter 3, Topic 3.3)

- Set Fast PWM 8- bit Mode. WGM13: 10 = 0101
- Set Non- inverting Mode: COM1A1: A0 = 10
- Set Prescalar = 1: CS12: 10 = 001

After setting this timer to work in the PWM mode a delay of 125 us is generated (for 8 KHz Audio File) and at every 125 us interval a Byte from the PCM audio file is fetched from the Memory Card and stored in the Compare Register of the Timer1, which gives its output on the OCR1A and OCR1B pins directly connected to the ear- phones as explained below.

Alternatively, a block of data bytes can be fetched from the file on the MMC and can be stored into a buffer variable in ATmega128. And then one by one the byte values can be given to the OCR1A and OCR1B registers.

7.3 Reading a Wav File from the Memory Card

First the header of WAV format is discussed and then the algorithm for reading a WAV file and playing it is given.

7.3.1 Wav Format Header

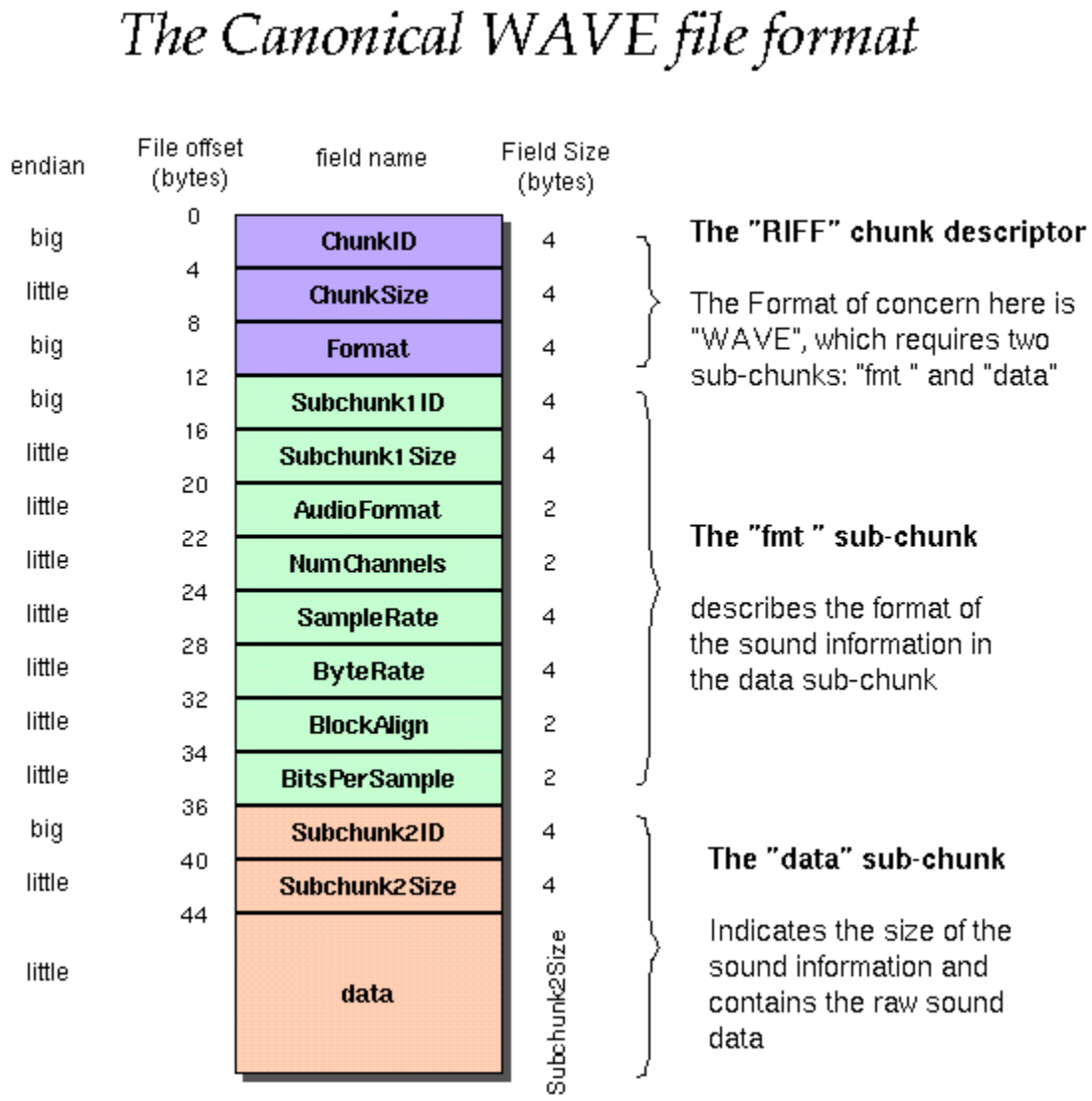


Fig 7.2 Wave File Format

As shown in the figure 7.2 the WAV file format header contains some useful information like “format”, “RIFF type”, “Number of Channels (1- Mono/ 2- Stereo)”, “Byte Rate” etc. The header has been used in the WAVE PLAYER for displaying relevant information on the LCD.

For example, first bytes 8- 11 are checked for containing the value “WAVE” to return if it is a valid WAV file.

Then the bytes 22- 23 are checked for number of channels. If it is a MONO file then this word will contain 1 and if it is a stereo file it will contain 2. Accordingly the file is played by either reading a single byte on every delay/ interrupt of 125 us or reading two bytes simultaneously.

Also the total playback time for the Audio file is displayed on the LCD by calculating according to the following formula:

Playback Time = Wave File Size/ (Bits per Sample * Sample Rate / No of Channels * 8)

7.3.2 Algorithm

1. Define a valid file pointer.

2. Open existing wav file in read mode.

3. Check for error in opening the file. If an error occurs go back to Step 1, else proceed.

4. Define a variable Buffer (Size: 2048 Bytes).

5. Point the File Pointer to Byte 44 (Start of RAW Audio Data).

6. Read 2048 Bytes into the Buffer. (File Pointer automatically incremented).

7. Loop 2048 times: Reading byte by byte values from the Buffer and outputting on the OCR pin, with a delay of 125 us everytime (for 8 KHz Audio).

8. Check if File Pointer = EOF. If not then go to Step 6. Else end.

The above algorithm is for an 8 KHz MONO WAV file. For other options there are minor changes as explained below:

1. If it is an 11.025 KHz file then the delay/ interrupt needs to be changed to 90 us instead of 125 us.

2. If it is a STEREO file then every time the loop is played two bytes are simultaneously read from the buffer and output to pins OCR1 and OCR2 respectively. The left channel of the earphones isd connected with pin OCR1 and the right with pin OCR2.

Chapter 8

MP3 Playback

8.1 Introduction to mp3 decoder chip: STA013

8.2 5V to 3V interface

8.3 STA013 Connections

8.4 Communication and Configuration via I2C

8.5 Sending MP3 Data

8.6 Algorithm for MP3 playback

MP3 Playback

MP3 is the most popular standard today for digital audio playback. Conceptually it is a lossy audio compression format, which usually generates audio files of size less than 1/10th of original CD audio with no perceivable difference. The key behind its encoding is to exploit human auditory limitations or auditory masking. Such kind of coding is also called perceptual coding. This is done by passing the audio data to a series of filter banks controlled by a psycho-acoustic model. Though decoding is far simpler than encoding process it yet demands the processing power that can't be delivered by ATmega128. Hence a dedicated hardware mp3 decoder chip "STA013" is used to perform data decoding. ATmega128 controls the chip as well as act as data buffer between MMC and STA013. This chapter describes STA013 interfacing with ATmega128 and CS4334 DAC interfacing which converts the PCM output of STA013 to analog output.

8.1 Introduction to mp3 Chip: STA013

The mp3 IC STA013 is a highly configurable chip. The focus is of the "normal" usage for the essential requirements of mp3 audio playback.

The chip is used in the Multimedia mode [10]. In this mode, the STA013 automatically detects the bit rate of MP3 data, and gives a signal requesting more data. The data is fed into the STA013 as rapidly as possible (up to 20 Mbits/sec), as long as it keeps asserting its data request signal. The MP3's sample rate (32, 44.1, 48 kHz) is also automatically detected, and the correct clock and data waveforms are created for the DAC. There is also a broadcast mode that requires the data to be fed at the correct bitrates. Broadcast mode is much more difficult to use and isn't very useful for most MP3 player designs. All the information here is focused only on multimedia mode.

The STA013 can create all the signals needed to drive a Digital to Analog Converter (DAC). The STA013 can also accept an external DAC clock, but for most applications this is unnecessary and more difficult.

It is possible to configure the STA013 to work with many different DAC chips and crystals, but here a 10 MHz crystal has been, which is inexpensive and easily available in India. Also for Digital to Analog Conversion a CS4334 DAC IC has been used, which provides excellent quality line-level audio output, and also includes 4X interpolation and a continuous time analog output filter that eliminates the need for external op-amps and complex output filtering. The CS4334 makes it simple and easy to obtain excellent quality line-level audio output without adding noise.

8.2 5 Volt to 3 Volt Interface

The STA013 is a 3 volt chip. The data sheet claims that it can run between 2.7 to 3.6 volts. It must not be used at 5 volts. Creating a 3 volt power supply is not difficult. Interfacing the 3 volt STA013 to 5 volt microcontrollers and DACs (CS4334) requires attention to make sure that they are properly received and to prevent damage to the 3 volt STA013. Because a 5 volt microcontroller is used in this project, the 5V to 3V conversion becomes necessary. There are some simple ways to connect the chips, as long as the microcontroller inputs have TTL compatible input thresholds (nearly all microcontrollers do).

Connecting the STA013 output pins to TTL level input pins is simple, because the STA013 will output at least 85% of its power supply for logic high, and a maximum of 0.4 volts for a logic low. Even if the STA013 is run at 2.7 volts, it will still output 2.3 volts, which will satisfy the 2.0 input requirement of a TTL level input pin. The four CS4334 input signals are all TTL level.

The outputs from a 5 volt powered chip must not be directly connected to the STA013 input pins. The STA013 inputs are not 5 volt tolerant. Each pin has a pair of input protection diodes. These diodes may conduct a small current. The simplest and easiest way to interface a 5 volt output to the STA013 input pins is with a series resistor that will limit the current when the 5 volt output is high. There is some input capacitance (3.5 pF) on the input pins, so adding a small capacitor in parallel with the resistor will allow the rapid edge to properly drive the input pin. The value of

the resistor and capacitor are not critical, Figure 4 shows 4.7K and 47pF, which limits the steady-state current to less than 1/2 mA, and has been tested. The capacitor is not really necessary if the pin doesn't need to be driven rapidly, such as the RESET pin.

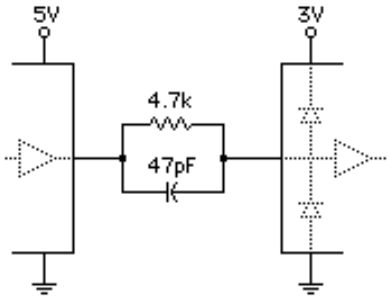


Fig 8.1 5V to 3V Conversion for STA013 [11]

8.3 STA013 Connections

(Refer to Chapter: Circuits for the Schematic and Layout)

The resistors connected to PVDD and PVSS are 4.7 Ohm. Any value near a few ohms is ok, if 4.7 ohms is not available. The capacitors connected to the crystal may need to be changed to match the resonance of the crystal. Values are usually in the range of 15 to 47 pF, and many crystals are not very sensitive to the exact capacitor values used. The inductor shown on the CS4334 power supply is intended to block noise from digital circuitry that's also connected to that line. The value of the inductor is not critical, so 100 µH would probably work well. In no inductor is available, and alternative is to create a separate supply for the CS4334 with a LM7805 or similar linear-mode voltage regulator.

8.4 Communication and Configuration via I2C

I²C is a 2-wire protocol, created by Philips. One line acts as a clock (SCL) and the other data (SDA). The protocol defines the ability to have multiple masters initiate communication, but here only the simple and common case is considered where the microcontroller is the only device that controls the bus, and all the other chips (like the STA013) respond to queries initiated by the microcontroller.

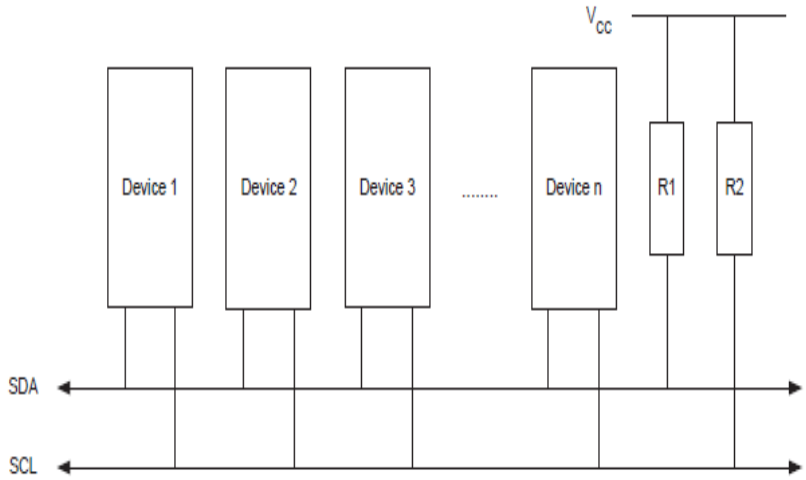


Fig 8.2 I^2C Protocol

For simple applications (like using the STA013), there are four fundamental operations. Only these four operations are needed to build routines that access the STA013 chip:

Start Condition:

This is a high-to-low transition of the SDA line, while SCL is high. Normally SDA only changes when SCL is low. When SDA changes while SCL is high, it means either the start or stop of a communication, instead of data transfer.

Send a Byte, Get ACK Bit:

Eight bits are sent by the microcontroller, each write to SDA occurs while SCL is low. A ninth clock is given and the microcontroller receives an ACK bit from the STA013. If the STA013 received the byte, it will send a zero in this bit.

Receive a Byte, Send ACK Bit:

The microcontroller gives eight clocks on SCL, and after each low-to-high clock transition, a bit is read from the STA013. During a ninth clock, the microcontroller pulls SDA low to acknowledge that it received the byte.

Stop Condition:

This is a low-to-high transition of the SDA line, while SCL is high. After the stop condition, both lines are left high, which is the idle state of the I²C bus.

This is shown in the given figure.

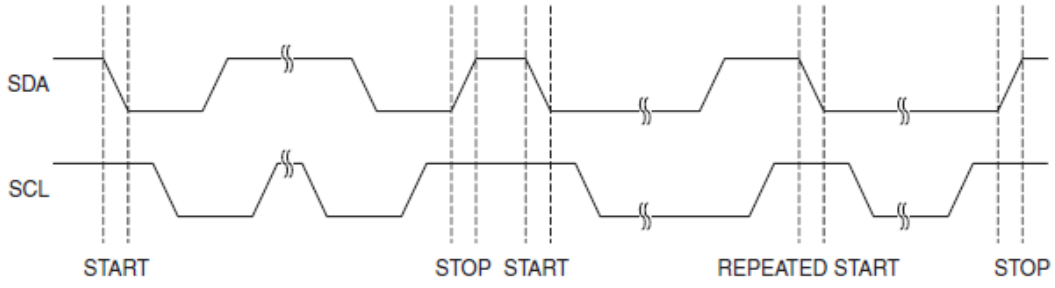


Fig 8.3 Basic I^2C Operations

8.5 Sending MP3 Data

The first step is to check that the STA013 is actually present. For that the address 0x01 is read from. If the read routine returns with an error, then no device sent an ACK bit and there is no chip installed. If there is an ACK, the data returned should always be 0xAC. Any other value means that the STA013 (or the code implementing the communication) isn't working properly. It is also possible to read the STA013 revision code at address 0, but ST gives not useful info about what this means, so it's probably not worth the trouble. The important step is to verify the ACK and 0xAC data at address 1, which means that the STA013 is present and communicating properly.

Sending MP3 data to the STA013 is simple. The basic idea is to give the STA013 data when it requests more. The STA013 chip determines what the bit rate is, consumes the data at the correct speed, and gives the request signal when it needs more. As the bits are transferred, it will end the request when its buffer is nearly full. Variable bit rate files are automatically handled. It also automatically detects the required sample rate (44.1 kHz, 48 kHz, etc) from the MP3 data and automatically adjusts the DAC clock. Usually the data is sent as fast as possible (as log as the speed is below 20 Mbits/sec).

The STA013 will ignore non-MP3 data (producing no sound), so it's safe to feed entire MP3 files, which may contain ID3 tags, into the STA013 without concern for which part of the file is MP3 bit stream and what portion is the ID3 tag. The chip will just keep requesting more data until it sees valid MP3 information. It's also safe to feed damaged MP3 streams into the STA013, for example truncated files due to partial download. Most damaged MP3 data is completely ignored. Some damaged files will produce a brief chirp sound (usually depending on what follows immediately after the damaged part), but the STA013 will rapidly sync back to good data that's sent after a corrupted part.

8.6 Algorithm for mp3 playback

```
i2cinit

      i2cstart
            write 0x86
            write 0x01
            i2cstop
            i2cstart
            write 0x87
            read var
      i2cstop

if var = 0xAC then
      proceed
else
      print "error"
      i2cstart
            write 2007 values from the p02_0609.bin
            write configpll values for cs4334 and 10 MHz
            write 1 to address 114 to start running the chip
            write 1 to address 19 to start playing the music
      i2cstop
      for (;;)
      {
            while (DATA_REQ)
            {
            write data to sta013 buffer (about 200 bytes size)
            }
      }
```

Embedded System Development - II

Applications for Embedded Linux on AVR32

Chapter 9

NGW100-Evaluation board for AVR32

NGW100- Evaluation board for AVR32

This Chapter concentrates on NGW100, which is the evaluation board built around the 32 bit microcontroller of Atmel's AVR AT32AP7000. It will take into consideration the architecture and the features of the micro controller. It will also deal with the kit, its peripherals and all the facilities that it provides.

9.1 Introduction to AT32AP7000

The AT32AP7000 is a complete System-on-chip application processor with an AVR32 RISC Processor achieving 210 DMIPS running at 150 MHz. AVR32 is a high-performance 32-bit RISC microprocessor core, designed for cost-sensitive embedded applications, with particular emphasis on low power consumption, high code density and high application performance [2].

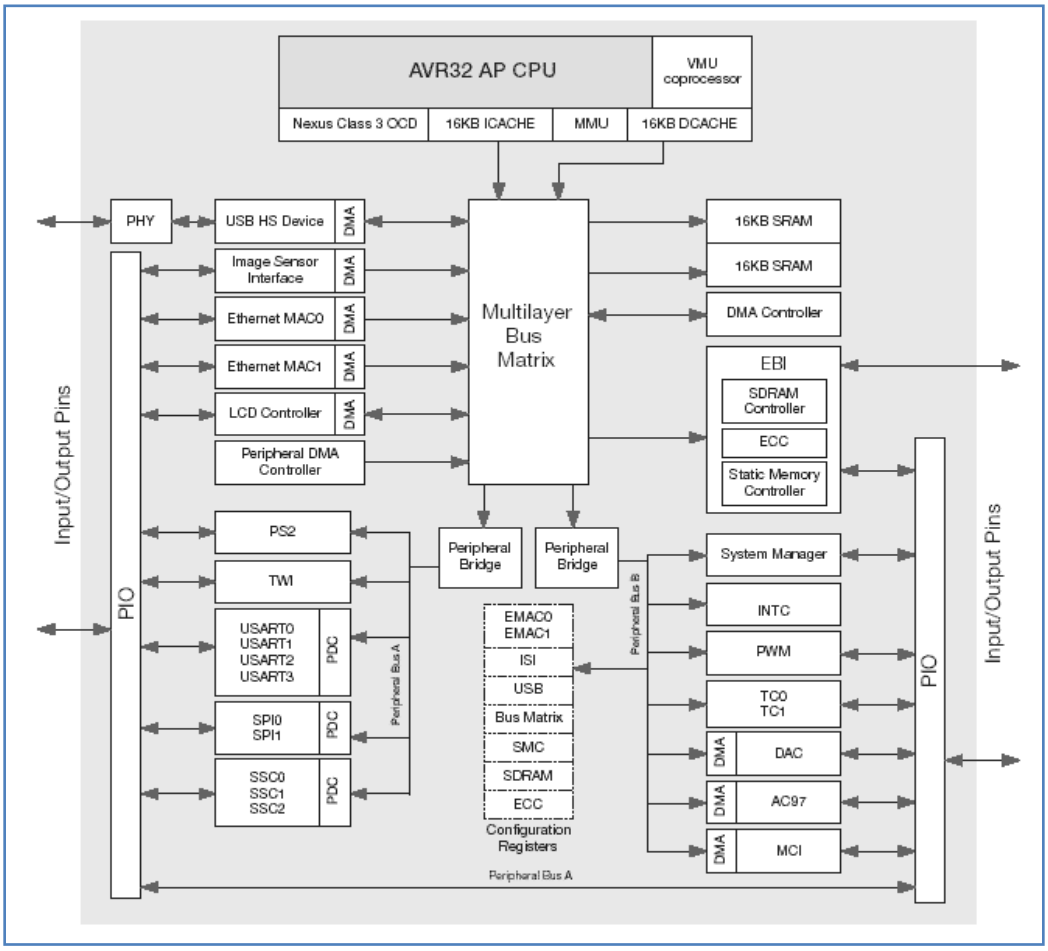


Fig 9.1 Architecture of AT32AP7000

9.1.1 Features

- 32-bit load/store AVR32B RISC architecture.

Memory load and store operations are provided for byte, half-word, word and double word data with automatic sign- or zero extension of half-word and byte data. The C-compiler is closely linked to the architecture and is able to exploit code optimization features, both for size and speed

- 7 stage pipeline

AVR32 AP is a pipelined processor with seven pipeline stages. The pipeline has three subpipes, namely the Multiply pipe, the Execute pipe and the Data pipe. These pipelines may execute different instructions in parallel. Instructions are issued in order, but may complete out of order (OOO) since the subpipes may be stalled individually, and certain operations may use a subpipe for several clock cycles.

- MMU

The AVR32 Memory Management Unit (MMU) is responsible for mapping virtual to physical addresses. When a memory access is performed, the MMU translates the virtual address specified into a physical address, while checking the access permissions. The MMU architecture uses paging to map memory pages from the 32-bit virtual address space to a 32-bit physical address space.

- 16Kbyte Instruction and 16Kbyte data caches.
- Pixel Coprocessor (PICO)

The Pixel Coprocessor (PICO) is a coprocessor coupled to the AVR32 CPU through the TCB (Tightly Coupled Bus) interface. The PICO consists of three Vector Multiplication Units (VMU0, VMU1, VMU2), an Input Pixel Selector and an Output Pixel Inserter. Each VMU can perform a vector multiplication of a 1x3 12-bit coefficient vector with a 3x1 8-bit pixel vector. In addition a 12-bit offset can be added to the result of this vector multiplication.

- Debug and test system

AVR32 AP OCD system is designed in accordance with the Nexus 2.0 standard which is a highly flexible and powerful open on-chip debug standard for 32-bit microcontrollers.

- DMA Controller

The DMA Controller (DMACA) is the central DMA controller core that transfers data from a source peripheral to a destination peripheral over one or more System Bus. It also has a Peripheral DMA Controller (PDC), which transfers data between on-chip serial peripherals such as the UART, USART, SSC, SPI, and the on- and off-chip memories. Using the Peripheral DMA Controller avoids processor intervention and removes the processor interrupt-handling overhead. This significantly reduces the number of clock cycles required for a data transfer and, as a result, improves the performance of the microcontroller and makes it more power efficient.

- Bus system

The Bus Matrix implements a multi-layer bus structure, that enables parallel access paths between multiple High Speed Bus (HSB) masters and slaves in a system, thus increasing the overall bandwidth.

9.1.2 Peripherals

- External Bus Interface

The External Bus Interface (EBI) is designed to ensure the successful data transfer between several external devices and the embedded Memory Controller of an AVR32 device. The Static Memory, SDRAM and ECC Controllers are all featured external Memory Controllers on the EBI. These external Memory Controllers are capable of handling several types of external memory and peripheral devices, such as SRAM, PROM, EPROM, EEPROM, Flash, and SDRAM.

- Serial Peripheral Interface

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave Mode. It also enables communication between processors if an external processor is connected to the system.

- Two Wire Interface

The Two-wire Interface (TWI) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 Kbits per second, based on a byte-oriented transfer format. It can be used with any Atmel two-wire bus Serial EEPROM. The TWI is programmable as a master with sequential or single-byte access. A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

- SDRAM Controller

The SDRAM Controller (SDRAMC) extends the memory capabilities of a chip by providing the interface to an external 16-bit or 32-bit SDRAM device. The page size supports ranges from 2048 to 8192 and the number of columns from 256 to 2048. It supports byte (8-bit), half-word (16-bit) and word (32-bit) accesses.

- USART

The Universal Synchronous Asynchronous Receiver Transceiver (USART) provides one full duplex universal synchronous asynchronous serial link. Data frame format is widely programmable (data length, parity, number of stop bits) to support a maximum of standards. The receiver implements parity error, framing error and overrun error detection. The receiver time-out enables handling variable-length frames and the transmitter timeguard facilitates communications with slow remote devices. Multidrop communications are also supported through address bit handling in reception and transmission.

- Serial Synchronous Controller

Provides serial synchronous communication links used in audio and telecom applications (with

CODECs in Master or Slave Modes, I2S, TDM Buses, Magnetic Card Reader, etc.). The Atmel Synchronous Serial Controller (SSC) provides a synchronous communication link with external devices. It supports many serial synchronous communication protocols generally used in audio and telecom applications such as I2S, Short Frame Sync, Long Frame Sync, etc

- AC97 Controller

The AC97 Controller is the hardware implementation of the AC97 digital controller (DC'97). The AC97 Controller communicates with an audio codec (AC97) or a modem codec (MC'97) via the AC-link digital serial interface. All digital audio, modem and handset data streams, as well as control (command/status) informations are transferred in accordance to the AC-link protocol. The AC97 Controller features a DMA Controller interface for audio streaming transfers.

- Audio Bitstream DAC

The Audio Bitstream DAC converts a 16-bit sample value to a digital bitstream with an average value proportional to the sample value. Two channels are supported, making the Audio Bitstream DAC particularly suitable for stereo audio.

- PS/2 Module

The PS/2 module provides host functionality allowing the MCU to interface PS/2 devices such as keyboard and mice. The module is capable of both host-to-device and device-to-host communication.

- Timer Counter

The Timer Counter (TC) includes three identical 16-bit Timer Counter channels. Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation

- Pulse Width Modulation Controller

The PWM macrocell controls several channels independently. Each channel controls one square output waveform. Characteristics of the output waveform such as period, duty-cycle and polarity are configurable through the user

- MultiMedia Card Interface

The MCI includes a command register, response registers, data registers, timeout counters and error detection logic that automatically handle the transmission of commands and, when required, the reception of the associated responses and data with a limited processor overhead. The MCI supports stream, block and multi-block data read and write, and is compatible with a DMA Controller, minimizing processor intervention for large buffer transfers.

- USB Interface

The USB High Speed Device (USBA) is compliant with the Universal Serial Bus (USB), rev 2.0 High Speed device specification. Each endpoint can be configured in one of several USB transfer types. It can be associated with one, two or three banks of a dual-port RAM used to store the current data payload.

- LCD Controller

The LCD Controller consists of logic for transferring LCD image data from an external display buffer to an LCD module with integrated common and segment drivers. The LCD Controller is programmable in order to support many different requirements such as
resolutions up to 2048 x 2048

- Ethernet MAC (MACB)

The MACB module implements a 10/100 Ethernet MAC compatible with the IEEE 802.3 standard using an address checker, statistics and control registers, receive and transmit blocks, and a DMA interface.

9.2 NGW100- Network Gate Way kit.

The NGW100 uses the AT32AP7000 which combines Atmel's AVR32 Digital Signal Processor CPU with an unrivalled selection of communication interfaces. It is in a true sense a Multimedia and Networking Development board. The NGW100 has two Ethernet ports, SD and MMC card reader, and connectors for USB and JTAG. An expansion header can be used for prototyping. A pre-installed Linux image on the enclosed 256 MB SD card ensures that the user can boot Linux and start program development directly after power up. The NGW100 is also an ideal development board for the AT32AP7000. All resources are available, and it supports communication on any of the device's communication interfaces. The board is preloaded with Linux and shipped with I/O interface drivers that can be called from our own code.

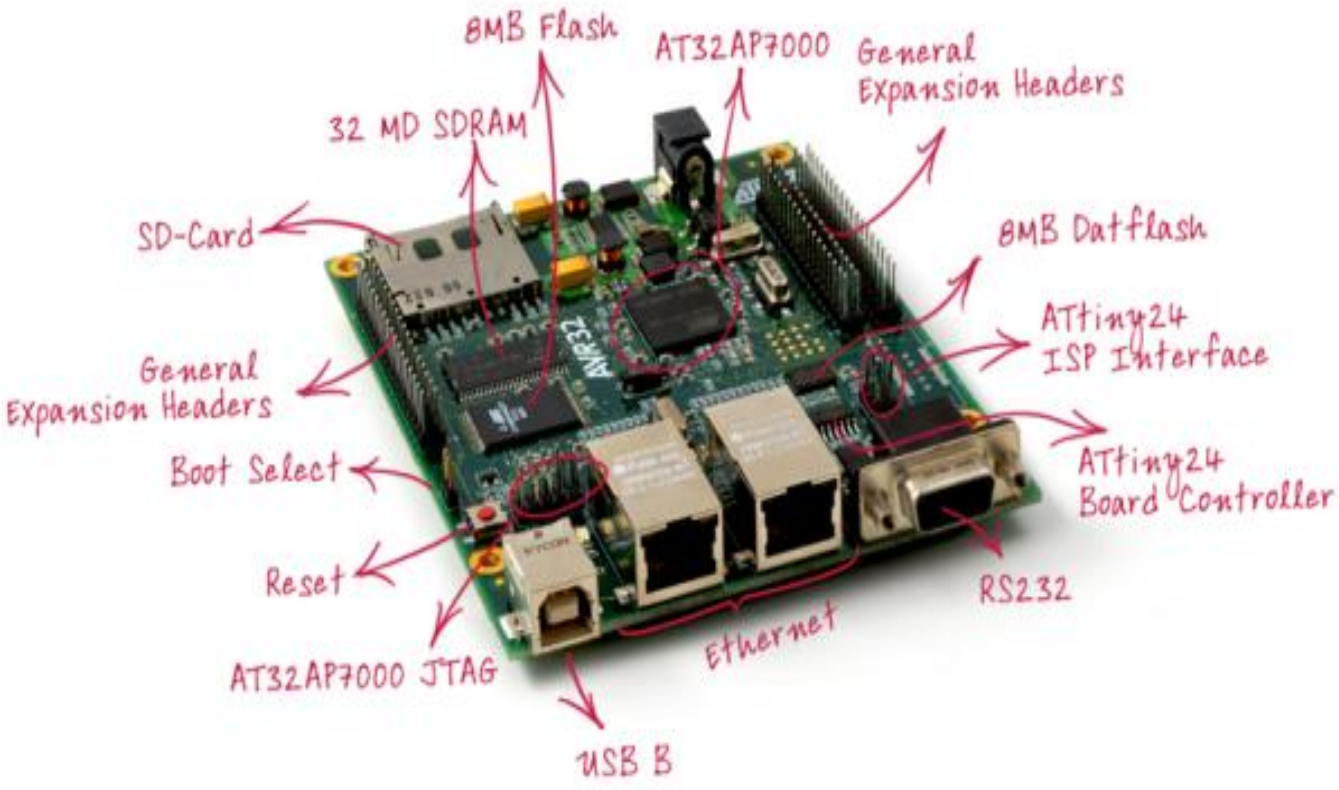


Fig 9.2 NGW100 board

9.2.1 Features

- Dual Ethernet PHY and 2 RJ45 connectors

The Ethernet subsystem on STK1000 allows testing applications that use Ethernet for connectivity to other computers. STK1000 provide a flexible solution with a dual Ethernet physical layer chip (PHY) to allow testing of both Media Independent Interface (MII) and Reduced Media Independent Interface (RMII). The two Ethernet interfaces each have a RJ45 Ethernet connector and are named ETH_A and ETH_B. These Ethernet interfaces are connected to the Media Access Control (MAC) 0 and 1 respectively.

- 2 PS2 connectors

PS2 connectors and interfaces allow connecting a PS2 compatible mouse or keyboard to STK1000. Mouse and keyboard can be used as input to Linux (AP7000 provides Linux firmware).

- 2 RS232 connectors

RS232 interfaces can be used to connect a PC to NGW100. This is particularly useful when communicating with the U-boot boot loader and Linux when these run on NGW100. With the help of hyper terminal, debugging can be carried out through the serial port.

- SD/MMC memory card connector and Compact Flash connector

NGW100 have a combined MMC and SD card connector. MMC and SD memory cards and CF cards are used to connect secondary storage to products. Secondary storage can be used to store operating systems and it can be used to store data such as music or movies. Hand held GPS devices may use the memory card to store maps or tourist guides. The SD and MMC connector on STK1000 allows to create and test such applications in the lab before you make your own product. The MMCI interface from the AVR32 device on the daughter board is designed specifically for interfacing SD and MMC memory card.

- Expansion Headers

The general expansion header allows to connect to some of the interfaces used on NGW100 for easy measurements and to expand the functionality of the board. NGW100 provide two general expansion headers, these give access to serial interfaces on NGW100. These expansion headers are generally used to interface the kit with LCD, VGA and other devices which use SPI , I2C communication protocols, etc.

- Terminals

Three possible terminal connections are available on the NGW

1. Serial terminal.

   It is available on the *serial port* (RS232) on the NGW, also used for kernel messages, to run a terminal program with communication speed to 115200 bps.

2. Telnet terminal

   It is available by connecting to *telnet* when connected to the LAN port using any telnet client. It is fast but has low security

3. SSH terminal

   Available by connecting to *ssh* when connecter to the LAN port using any ssh client. This provides high security.

Chapter 10

Embedded LINUX

Embedded Linux

This Chapter will concentrate on the Embedded Linux, the kernel which has been used as the base in our project, its compilation process, how to obtain the file system which is to be loaded on the kit. It will also focus on how to boot the kernel from the kit via the SD card.

10.1 Introduction

Linux is a multi-user, multi-tasking, network enabled operating system. In many embedded applications, the time a system requires to boot is often critical. Because Linux allows to omit the features you do not need when building the kernel, the boot time will depend highly on the features you select. Because Linux allows you to omit the features you do not need when building the kernel, the boot time will depend highly on the features you select. When working with GNU/Linux, the user can select the exact features needed in a system, from scheduler and device drivers to file system support and services. This is all done before the system is compiled and it is selected as a list of features. If the functionality is wanted in the system, it is selected. If it is not, leave it unchecked and it will not be compiled in. Compressed Linux kernels can be very small, less than 500 KB, but this is all dependent on the number of features selected. One such kernel with dedicated features especially for embedded systems is the Embedded Linux kernel, which we have used in this project. Atmel has a dedicated Linux support team for the AVR32. The team is responsible for the AVR32 port of the kernel itself, the device drivers and applications.

10.2 Buildroot

Buildroot is an open-source project, used by Atmel to build its Linux board support packages for development kits and reference designs.

Buildroot is a configurable and fully automated build system for embedded systems. The main idea is that the user selects what he wants installed on the system, and Buildroot takes care of compiling everything from sources, creating a custom file system image that can be programmed into flash, put on an MMC/SD card, or unpacked on an NFS server.

It is a set of scripts that builds an entire root file system for a given target. The scripts are based on a combination of Makefile and kconfig that is commonly used in many projects. Kconfig is used to give the user an easy configuration interface that is stored in a file. The Makefile system then reads out the values stored by kconfig and configures a set of rules in which different software is compiled. Buildroot start by compiling the toolchain if requested, or it can use an external toolchain. It then moves over to the Linux kernel, software libraries and applications. Finally it combines all the applications with the needed libraries and kernel to a file system image. This image is ready for the user to deploy on his target.

10.2.1 Requirements for using Buildroot

Buildroot is a script system that heavily depends on a Linux system. It is highly recommended that users either run a native Linux installation or run Linux within a virtual machine when running other operating systems [12].

It is supported by most host architecture and requires quite a lot of disk space. Having 5 GB free disk space before starting to work on you root file system is in general a good idea.

The build system also needs a set of host tools preinstalled on the build machine. Most Linux distributions allow the user to install a package covering the build essentials.

List of requirements for the build machine:

- C++ compiler (for Qtopia® (G++))
- GNU make
- sed
- flex
- bison
- patch
- gettext
- libtool
- texinfo
- autoconf (version 2.13 and 2.61)
- automake

- ncurses library (development install)
- zlib library (development install)
- libacl library (development install)
- lzo2 library (development install)
- GCC compiler

10.2.2 Getting started for AVR32 targets

Start off by downloading the latest release from Atmel®, extract it somewhere on you system and enter the buildroot-avr32-<version> directory.

To load the default configuration for these boards, simply type on of the following depending on your board:

make atngw100_defconfig

make atngw100-base_defconfig

Buildroot will now load the board's configuration and save it in a file called *.config*.

An optional step is to download all the source files before starting the build. This can be initiated by typing:

make source

The next step for the user is to start the build process. This is done by typing:

make

Buildroot will now start to download software packages from the Internet, extract them on your local file system and compile them for AVR®32.

When the build process is finished and successful, you will find the created root file systems in the binaries/<board name>/ directory. There can be various types of ready binaries depending on what is chosen by the configuration system.

10.2.3 Directory structure

Buildroot has a well defined structure of directories. From a fresh extraction of the tarball (analogous to winzip/winrar in windows) Buildroot will look like:

- *Config.in*
- *.defconfig*
- *docs/*
- *Makefile*
- *package/*
- *project/*
- *target/*
- *TODO*
- *toolchain/*

After a successful build there will be 6 extra directories in the base directory of Buildroot:

- *binaries/*
- *build_avr32/*
- *dl/*
- *include/*
- *project_build_avr32/*
- *toolchain_build_avr32/*

binaries/

This directory contains the successfully built images. Populated on the form binaries/<project name>/.

build_avr32/

Directory used for building each library and application. Also, by default, holds the staging_dir/ directory which contain the toolchain and libraries.

.config

File which contains the system configuration, this will appear after the first time running menuconfig or loading a default configuration.

Config.in

The top level configurations file for the kconfig system.

.defconfig

File with a default general configuration for Buildroot, mainly for x86 architecture.

dl/

This directory will contain all the downloaded source tar archives so the user only has to download libraries and applications once.

docs/

In this directory the documentation for Buildroot can be found.

include/

Directory made by kconfig, used for knowing which values are stored during configuration setup. Leave it as is, everything is auto generated.

Makefile

Top level makefile for Buildroot describing the initial rules for how to build a root file system. This file will again include all the other makefiles spread out in the sub directories.

package/

Directory containing all the makefiles that describes how each library and application shall be built.

project/

Directory containing information about project building that allows Buildroot to be able to have multiple projects ongoing in the same extracted source tree. Buildroot will share compiled applications, libraries and toolchain whenever possible.

project_build_avr32/

The project specific part of the build is located in this directory. Is populated on the form project_build_avr32/<project name>/. Here you will find the target specific Linux kernel; Busybox and uncompressed root file system.

target/

Directory containing information about targets that is where the different boards are defined, board specific patches to the Linux kernel and board specific files. Typically this is where the base for the file system is located, called target_skeleton.

TODO

TODO list for upstream Buildroot describing what areas is in the works. Feel free to work on TODO material if you are confident with how Buildroot works.

toolchain/

Directory containing makefiles for the toolchain that describes how it should be built and what options a user have. The toolchain is a vital part of buildroot, since you are not capable to build anything without a working toolchain. Given that the user has chosen an internal toolchain.

toolchain_build_avr32/

Directory containing the extracted toolchain source tarballs and where Buildroot builds the toolchain.

10.2.4 Flexibility and configuration

Buildroot is also highly flexible. For starter we recommend following the getting started guide, but you can also configure which applications to put into your image.

Enter the base directory of Buildroot and type:

make menuconfig

A curses based menu system will guide you around the different choices.

If a user wants to use the atngw100 or atstk1002 default configuration as a base, but do minor adjustment the following short guide will configure Buildroot to build it as a new project.

Start off in the base directory for Buildroot and type:

make atstk1002_defconfig or what is appropriate for the target board.

make menuconfig

Project Options ---> Project name, change this to something descriptive.

Exit the menu system and save the configuration, then type:

make

Your file system will now be located in binaries/<project name>/ directory.

The kernel can be customized as per our requirement using the menuconfig uption. The target architecture, the version of kernel that we want, the type of file system, what all additional applications we want and several other things can be selected through it.

10.3 Deploying binaries to the target system

Buildroot creates binaries (anlogous to the .exe files in other systems) ready to download onto the target, using a flash programmer like avr32program or a SD-card. In the *binaries/* directory, in the base directory of Buildroot, the user can find the following files depending on the configuration system:

<board base name>-linux-<version>.gz

rootfs.avr32.ext2

rootfs.avr32.ext2.bz2

rootfs.avr32.jffs2

rootfs.avr32.jffs2-<partition_name>

rootfs.avr32.tar

rootfs.avr32.tar.bz2

u-boot.bin

<board base name>-linux-<version>.gz

This is the kernel targeted for your board. It is also included in the root file system images so the user does not need to explicit download this into flash.

rootfs.avr32.ext2

This is the EXT2 version of the root file system. It is targeted for SD/MMC-card boot, or other media that can hold an EXT2 file system. The .bz2 version of this file is a bzip2-compressed version of the same file system.

rootfs.avr32.jffs2

The JFFS2 root file system is target for the onboard flash device. Onboard flash is typically NOR, NAND or DataFlash®.

rootfs.avr32.jffs2-<partition_name>

Buildroot can make multiple JFFS2 images when the target device has more than one flash device or partition. In this case an extension is added to each image file indicating the target flash device or partition.

For example a build for the ATNGW100 can generate three images:

rootfs.avr32.jffs2
rootfs.avr32.jffs2-root
rootfs.avr32.jffs2-usr

Where the first file without an extension is only a file touched by the make system and should be ignored. The other two files are for programming into the two flash devices on the ATNGW100. They are named according to their content, where the root image is "/" and the usr image is "/usr"

rootfs.avr32.tar

A full file system is available in a tar archive. This is useful if the user wants to have the root file system mounted over network or extract the file system to a removable medium like SD/MMC-card.

u-boot.bin

U-boot is the boot loader made for the target board; this must be programmed with a flash programming tool like avr32program. Buildroot for AVR32 can be downloaded from http://www.atmel.com/dyn/products/tools_card.asp?tool_id=4401.

10.4 U-Boot

U-Boot is a universal boot loader for embedded boards based on PowerPC, ARM, MIPS and several other processors, which can be installed in a boot ROM and used to initialize and test the hardware or to download and run OS and application code. Sample features are:

- serial console support

- integrated shell alike setup interface
- optional password protection and timeout for access to setup interface on boot
- editable configuration space
- downloads software trough ftp servers
- flash routines for EEPROMS of misc technology including NANDs
- runs test applications directly
- boots Linux

10.5 Booting Linux from SD Card

There are 3 methods to boot linux from NGW100.They are 1. SD card, 2. Network. 3. Serial.

The network method requires some (advanced) setup in your network environment, but when in place, it will ease both upgrades and development with the Network Gateway. The serial method requires nothing but an RS-232 cable and the windows' hyper terminal. This method thus has the least prerequisites, but it is rather lengthy. The Network Gateway file system consists of 3 partitions namely the Uboot boot loader, the Linux root file system and the /usr partition of Linux.

For booting linux, firstly the u-boot has to be separately downloaded and installed on the kit. Proper care should be taken while specifying the physical address as to from where linux is supposed to be loaded as in from the sd card or from the flash. There are 2 options to upgrade the firmware on the NGW100 kit.

- A tar archive with the root and usr image files to be written to the Network Gateway flash.
- A pre-built SD card image for automatic upgrade with the SD card method below.

If the first option is selected u can customize the kernel the way you want by selecting and deselecting certain features and then compile it to from a uimage. While if the second option is used, a readily available uimage can be downloaded.

onnect a RS-232 cable between the Network Gateway kit and your PC to gain access to the boot console. (com port settings 115200-8N1) When powering up the kit, press space to abort the automatic loading. This will present you with the bootloader prompt: "Uboot>"

Assuming that sdb is the device of your SD card, write the image with dd on your Linux workstation with:

~# sudo dd if=ngw_fw_upgrade.img of=/dev/sdb

Note that the image distributed is a complete SD card image, the device should thus be sdb not sdb1 etc. With the SD card ready, you must now boot the Network Gateway from the SD card and copy the usr and root images to flash. This is done automatically by the pre-built image and you can monitor progress on the rs-232 console. To boot the SD card type the following in Uboot:

Uboot> askenv bootcmd
Please enter 'bootcmd': mmcinit; ext2load mmc 0:1 0x10300000 /uImage; bootm 0x10300000
Uboot> set bootargs 'console=ttyS0 root=/dev/mmcblk0p1 ro'
Uboot> boot

The Linux system will now boot, and if you are using the pre-built image, upgrade will complete without further user interaction. The LEDs on ATNGW100 will flash while upgrading, and light steady when completed. At this point the Linux prompt will be presented at the console. Remove power and SD card and the Network Gateway upgrade is complete. If the upgrade fails, the LEDs will turn off.

Chapter 11

Applications on LINUX

Applications on LINUX

This Chapter concentrates how to install applications on the Embedded LINUX platform, the softwares to be used, where to get those from and how to make those applications work on NGW100. In our project we have included two major applications i.e. for audio playback and video through the VGA port. These two applications have been discussed in detail [13].

11.1 Busybox

BusyBox is an open-source software application that provides many standard Unix tools, much like the larger (but more capable) GNU Core Utilities. BusyBox is designed to be a small executable for use with Linux, which makes it ideal for special purpose Linux distributions and embedded devices. It has been called "The Swiss Army Knife of Embedded Linux". BusyBox can be customized to provide a subset of over two hundred utilities. It can provide most of the utilities specified in the Single Unix Specification plus many others that a user would expect to see on a Linux system.

Typical computer programs have a separate binary (executable) file for each application. BusyBox is a single binary, which is a conglomerate of many applications, each of which can be accessed by calling the single BusyBox binary with various names (supported by having a symbolic link or hard link for each different name) in a specific manner with appropriate arguments.

11.2 iPKG (Itsy Package Management System)

ipkg is a lightweight, simplistic package management system (analogous to Windows installer). A package management system does just that -- it manages precompiled software which has been bundled into modular, interdependent packages.

Advantages

- The ipkg client and libraries are relatively small

- The installed meta-data is minimal yet fully functional
- Actively maintained
- Well developed, active user base

Disadvantages

- Limited features, e.g., extra scripts are required to set up users or groups
- Documentation is limited and often outdated or inaccurate
- Significant changes in development can cause incompatibilities between versions
- Root access is generally required to build a package and to install a package
- Limited control over owner, group, and access permissions of package contents

The software packages are contained in special .ipk files. An .ipk file is a specially constructed tar archive (previous versions of .ipk files were gzipped tar archives, like .deb files - this is a discouraged but still supported format) containing three parts:

./data.tar.gz

Contains the actual files belonging to this package, organized as they should appear after installation

./control.tar.gz

Contains a file describing the package meta-data and any installation/removal scripts for the package

./debian-binary

This file is for comparability with Debian's package system and currently is ignored by ipkg

11.3 Audio from NGW100

To get audio from NGW100, the ABDAC of the controller AT32AP7000 is made use of. The ABDAC is a very simple peripheral and its use is straight forward. It needs a clock signal provided from the generic clock system and data input to the channels. See the block diagram in Fig to get an overview of the module.

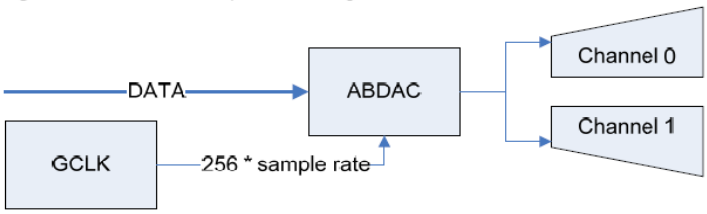


Fig 11.1 block diagram: audio from NGW100

11.3.1 Functional Description

The ABDAC uses a generic clock to provide the sample frequency. This generic clock is hard wired inside the device and must be 256 times the sample frequency. The generic clock should be configured and enabled before the ABDAC is enabled.

When the ABDAC is enabled it expects the Sample Data Register (SDR) to be updated at the same interval as the output sample rate. Both channels can be updated with one write instruction, since they are in the same I/O register (SDR). There are two interrupts available to offload the CPU. The TX_READY interrupt can be used as a trigger to signal that the next sample for each channel can be written. The ABDAC may be connected to a DMA controller on the device. This will offload the CPU when transferring data from a buffer in RAM to the ABDAC. The application will only need to fill a buffer and pass the buffer address to the DMA controller.

11.3.2 Electrical Connection

The output from the device is not intended for driving headphones or speakers. The pads are limiting the maximum amount of current. In the majority of all practical cases, this will not be enough to drive a low impedance source. Because of this limitation, an external amplifier should be connected to the output lines to amplify these signals. This amplifier device could also be used to control the volume. For testing purposes a line in or microphone input on a sound system can be used to evaluate the output signal.

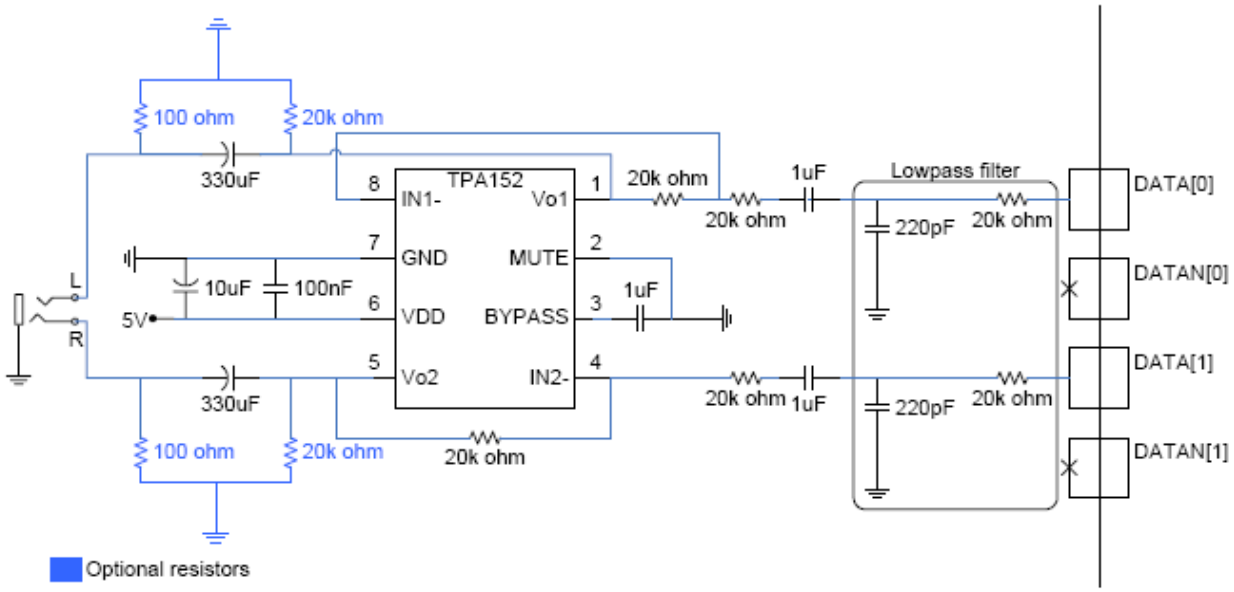


Fig 11.2 Electrical connections: Audio from NGW100

11.4 Video from AVR32

There are no direct connections from the NGW100 kit from where video output can be taken. A separate daughter board has to be made which is then connected to the expansion header on the board which is especially meant for LCD/VGA outputs [14].

11.4.1 Display interface

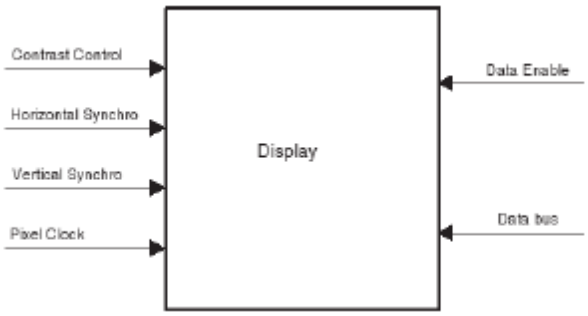


Fig 11.3 Interface for display output.

The typical interface of a display is based on analog and digital inputs. The digital lines are a data bus, pixel clock, vertical and horizontal synchronization signals and a data enable line. The

voltage input is generally used for contrast control. The LCD Controller drives all these interface lines. If more inputs are required by the display, it may be managed by GPIOs (power control, AC bias on some STN displays, backlight control, etc.) or other peripheral (SPI, I2C etc.).
The screen size is fully programmable. The maximum size supported is 2048 x 2048. Most of standard display resolution levels are compatible: VGA (640x480), QVGA (320 x 240), etc. In STN mode, single and double scan modes are supported. For a double scan mode, the maximum size for each panel is 1024 x 2048. The interface must be 3.3V compliant.

11.4.2 Bandwidth considerations

An attached display needs a lot of bandwidth from the expansion bus of the system. This must be considered in the display selection process. The bandwidth needed is calculated by following formula:

NeededBandwidth = DispaySize x BitsPerPixel x FrameRate

Depending on the applications an operating system is running in parallel enough bandwidth should be available. For instance considering that the updating of the frame buffer could be at the same frame rate as the display the above formula must be multiplied by two. To calculate the available bandwidth on the system following formula can be used:

AvailableBandwidth = BusSpeed x BusInterfaceWidth

The maximum available bandwidth for the AP700x devices is

AvailableBandwidth = 75 x 32/8 MB/s = 300 MB/s

due to the maximum bus speed of 75 MHz. With the maximum speed it is possible to use displays up to VGA with full 24bit resolution and doing MPEG4 decoding at the same time. Larger displays can be used when the resolution is decreased.

11.4.3 16-bit memory example (as on the NGW100)

The NGW100 uses a 16bit wide interface to the external memory. Thus the maximum available bandwidth is:

AvailableBandwidth = 75 x 16/8 Bytes/s = 150 MB/s

While a QVGA display leaves enough room for MPEG4 decoding and other tasks a VGA display may not be used in this constellation. Never the less it is possible to show graphics on the VGA display which do not need to be updated at high frequencies.

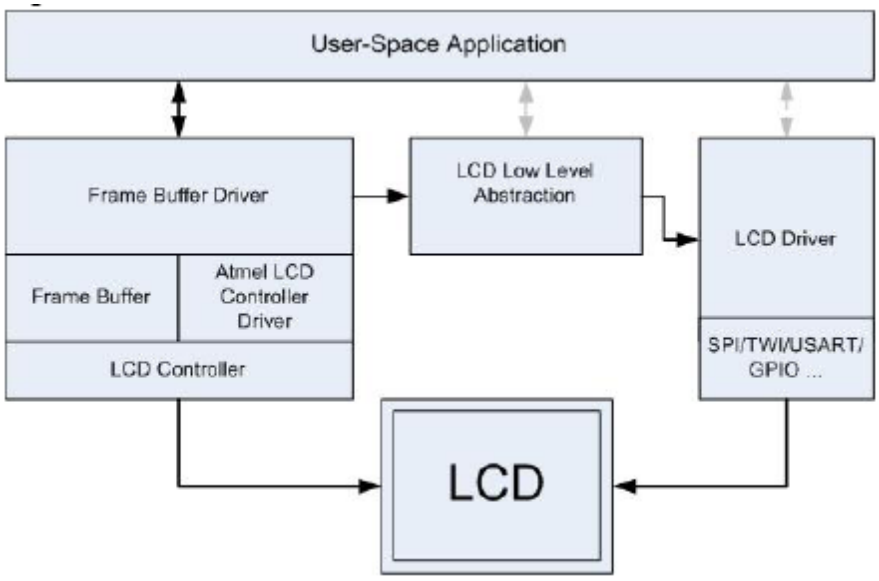


Fig 11.4 Software block diagram for VGA interface

11.4.4 LCD/VGA on NGW100

NGW100 doesn't come with a readymade LCD hardware, but there is a special expansion connector available on the board for LCD/VGA known as J7.

The main component of the video boards NGW100 is ADV7125KST, a video DAC, the three color channels can be implemented and thus is ideal for use in VGA converter.

The circuit is essentially the standard application of ADV7125KST, as also in the datasheet of the module can be found. With jumper JP1 allows the video DAC into a low-power mode may

be used. Important are the three 75-ohm resistors, they provide the correct impedance conditions on the data lines. The inverter buffers the IC 74LVC04 SYNC outputs of the AT32AP7000 and forms a protection against influences from outside.

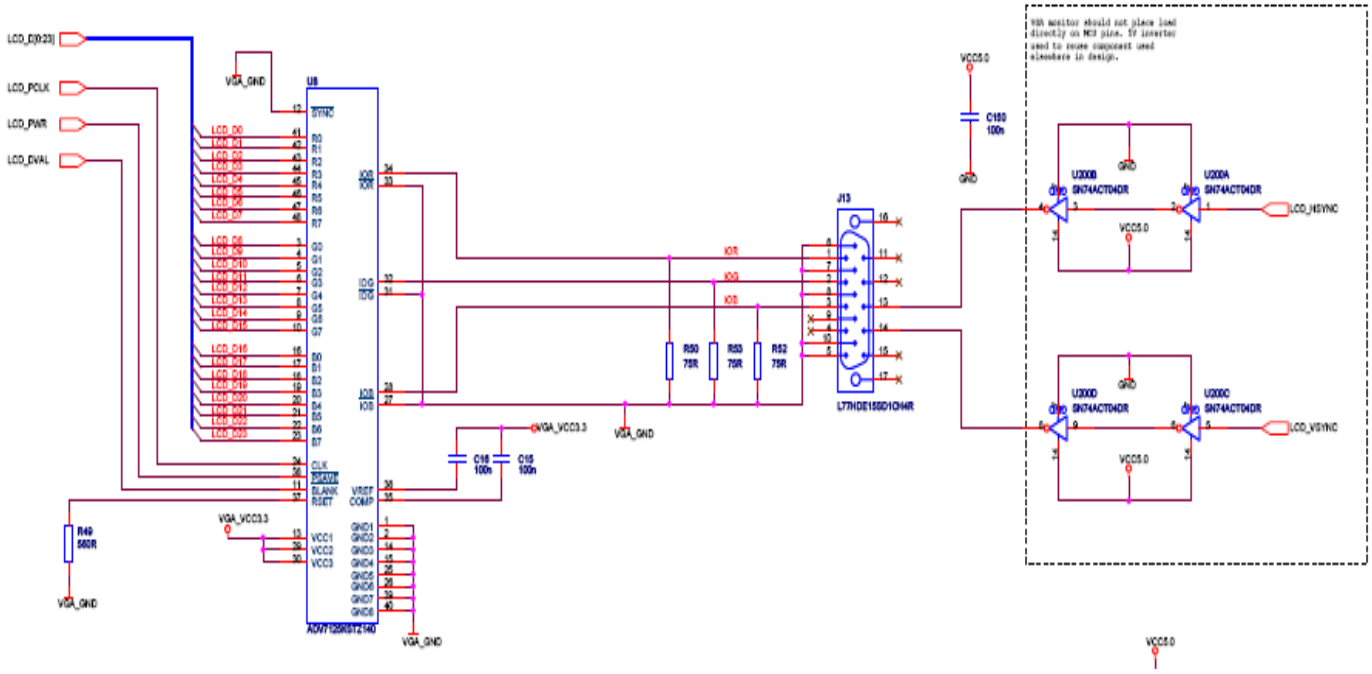


Fig 11.5 Hardware connections for VGA

11.4.5 An overview of ADV7125KST

General description:

The ADV7125 is a triple high speed, digital-to-analog converter on a single monolithic chip. It consists of three high speed, 8-bit video DACs with complementary outputs, a standard TTL input interface, and a high impedance, analog output current source. The ADV7125 has three separate 8-bit-wide input ports. A single +5 V/+3.3 V power supply and clock are all that are required to make the part functional. The ADV7125 has additional video control signals, composite SYNC and BLANK, as well as a power save mode. The ADV7125 is fabricated in a 5 V CMOS process. Its monolithic CMOS construction ensures greater functionality with lower power dissipation. The ADV7125 is available in 48-lead LQFP and 48-lead LFCSP packages.

Features:

- 330 MSPS throughput rate
- Triple 8-bit DACs
- RS-343A-/RS-170-compatible output
- Complementary outputs
- DAC output current range: 2.0 mA to 26.5 mA
- TTL-compatible inputs Internal Reference (1.235 V)
- Single-supply +5 V/+3.3 V operation
- 48-lead LQFP and LFCSP packages

Driver for the LCDC already in the kernel sources contain. Some sources have to be adjusted and the kernel needs to be re-configured and compiled. The resulting binaries will need to be transferred NGW100.

CONCLUSION

This project has been an attempt to develop a portable as well as scalable Embedded System with an aim of providing a device with multimedia and networking applications. In the first phase a portable media player board with functionalities such as Color LCD display, Touch Detection capability, Audio Playback and Multimedia Card interface was successfully developed. Also in the second phase of the project audio/ video applications were explored for the open source operating system Embedded Linux on a 32 bit processor AP7000 hardware platform. However the networking applications were not developed due to the time/ cost constraints and short resources.

APPENDIX I

General LINUX 2.6.22 Bash Prompt Commands

A

alias	Create an alias
apropos	Search Help manual pages (man -k)
apt-get	Search for and install software packages (Debian)
aspell	Spell Checker
awk	Find and Replace text, database sort/validate/index

B

bash	GNU Bourne-Again SHell
bc	Arbitrary precision calculator language
bg	Send to background
break	Exit from a loop
builtin	Run a shell builtin
bzip2	Compress or decompress named file(s)

C

cal	Display a calendar
case	Conditionally perform a command
cat	Display the contents of a file
cd	Change Directory
cfdisk	Partition table manipulator for Linux
chgrp	Change group ownership
chmod	Change access permissions
chown	Change file owner and group
chroot	Run a command with a different root
chkconfig	System services (runlevel)
cksum	Print CRC checksum and byte counts
clear	Clear terminal screen
cmp	Compare two files

comm	Compare two sorted files line by line
command	Run a command - ignoring shell functions
continue	Resume the next iteration of a loop
cp	Copy one or more files to another location
cron	Daemon to execute scheduled commands
crontab	Schedule a command to run at a later time
csplit	Split a file into context-determined pieces
cut	Divide a file into several parts
D	
date	Display or change the date & time
dc	Desk Calculator
ddrescue	Datarecovery tool
declare	Declare variables and give them attributes
df	Display free disk space
diff	Display the differences between two files
diff3	Show differences among three files
dig	DNS lookup
dir	Briefly list directory contents
dircolors	Colour setup for `ls'
dirname	Convert a full pathname to just a path
dirs	Display list of remembered directories
dmesg	Print kernel & driver messages
du	Estimate file space usage
E	
echo	Display message on screen
eject	Eject removable media
enable	Enable and disable builtin shell commands
env	Environment variables
ethtool	Ethernet card settings
eval	Evaluate several commands/arguments
exec	Execute a command

exit	Exit the shell
expect	Automate arbitrary applications accessed over a terminal
expand	Convert tabs to spaces
export	Set an environment variable
expr	Evaluate expressions
F	
false	Do nothing, unsuccessfully
fdformat	Low-level format a floppy disk
fdisk	Partition table manipulator for Linux
fg	Send job to foreground
fgrep	Search file(s) for lines that match a fixed string
file	Determine file type
find	Search for files that meet a desired criteria
fmt	Reformat paragraph text
fold	Wrap text to fit a specified width.
for	Expand *words*, and execute *commands*
format	Format disks or tapes
free	Display memory usage
fsck	File system consistency check and repair
ftp	File Transfer Protocol
function	Define Function Macros
fuser	Identify/kill the process that is accessing a file
G	
gawk	Find and Replace text within file(s)
getopts	Parse positional parameters
grep	Search file(s) for lines that match a given pattern
groups	Print group names a user is in
gzip	Compress or decompress named file(s)
H	
hash	Remember the full pathname of a name argument
head	Output the first part of file(s)

history	Command History
hostname	Print or set system name
I	
id	Print user and group id's
if	Conditionally perform a command
ifconfig	Configure a network interface
ifdown	Stop a network interface
ifup	Start a network interface up
import	Capture an X server screen and save the image to file
install	Copy files and set attributes
J	
join	Join lines on a common field
K	
kill	Stop a process from running
killall	Kill processes by name
L	
less	Display output one screen at a time
let	Perform arithmetic on shell variables
ln	Make links between files
local	Create variables
locate	Find files
logname	Print current login name
logout	Exit a login shell
look	Display lines beginning with a given string
lpc	Line printer control program
lpr	Off line print
lprint	Print a file
lprintd	Abort a print job
lprintq	List the print queue
lprm	Remove jobs from the print queue
ls	List information about file(s)

lsof	List open files
M	
make	Recompile a group of programs
man	Help manual
mkdir	Create new folder(s)
mkfifo	Make FIFOs (named pipes)
mkisofs	Create an hybrid ISO9660/JOLIET/HFS filesystem
mknod	Make block or character special files
more	Display output one screen at a time
mount	Mount a file system
mtools	Manipulate MS-DOS files
mv	Move or rename files or directories
mmv	Mass Move and rename (files)
N	
netstat	Networking information
nice	Set the priority of a command or job
nl	Number lines and write files
nohup	Run a command immune to hangups
nslookup	Query Internet name servers interactively
O	
open	Open a file in its default application
op	Operator access
P	
passwd	Modify a user password
paste	Merge lines of files
pathchk	Check file name portability
ping	Test a network connection
popd	Restore the previous value of the current directory
pr	Prepare files for printing
printcap	Printer capability database
printenv	Print environment variables

printf	Format and print data
ps	Process status
pushd	Save and then change the current directory
pwd	Print Working Directory
Q	
quota	Display disk usage and limits
quotacheck	Scan a file system for disk usage
quotactl	Set disk quotas
R	
ram	ram disk device
rcp	Copy files between two machines
read	read a line from standard input
readonly	Mark variables/functions as readonly
reboot	Reboot the system
renice	Alter priority of running processes
remsync	Synchronize remote files via email
return	Exit a shell function
rev	Reverse lines of a file
rm	Remove files
rmdir	Remove folder(s)
rsync	Remote file copy (Synchronize file trees)
S	
screen	Multiplex terminal, run remote shells via ssh
scp	Secure copy (remote file copy)
sdiff	Merge two files interactively
select	Accept keyboard input
seq	Print numeric sequences
set	Manipulate shell variables and functions
sftp	Secure File Transfer Program
shift	Shift positional parameters
shopt	Shell Options

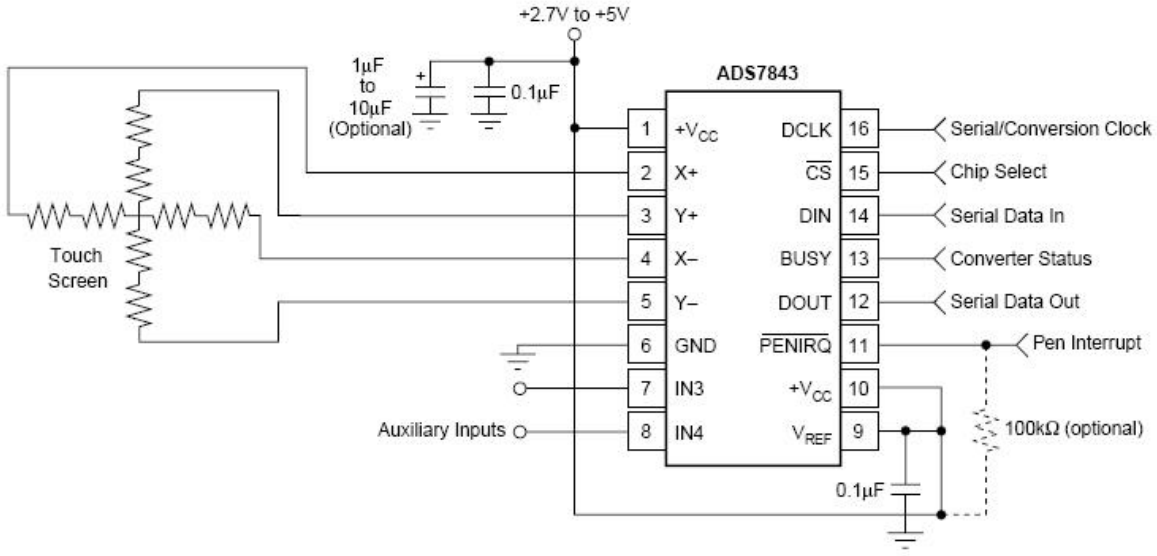
shutdown	Shutdown or restart linux
sleep	Delay for a specified time
slocate	Find files
sort	Sort text files
source	Run commands from a file `.'
split	Split a file into fixed-size pieces
strace	Trace system calls and signals
su	Substitute user identity
sudo	Execute a command as another user
sum	Print a checksum for a file
symlink	Make a new name for a file
sync	Synchronize data on disk with memory
T	
tail	Output the last part of files
tar	Tape ARchiver
tee	Redirect output to multiple files
test	Evaluate a conditional expression
time	Measure Program running time
times	User and system times
touch	Change file timestamps
top	List processes running on the system
traceroute	Trace Route to Host
trap	Run a command when a signal is set(bourne)
tr	Translate, squeeze, and/or delete characters
true	Do nothing, successfully
tsort	Topological sort
tty	Print filename of terminal on stdin
type	Describe a command
U	
ulimit	Limit user resources

umask	Users file creation mask
umount	Unmount a device
unalias	Remove an alias
uname	Print system information
unexpand	Convert spaces to tabs
uniq	Uniquify files
units	Convert units from one scale to another
unset	Remove variable or function names
unshar	Unpack shell archive scripts
until	Execute commands (until error)
useradd	Create new user account
usermod	Modify user account
users	List users currently logged in
uuencode	Encode a binary file
uudecode	Decode a file created by uuencode
V	
vdir	Verbosely list directory contents (`ls -l -b')
vi	Text Editor
vmstat	Report virtual memory statistics
W	
watch	Execute/display a program periodically
wc	Print byte, word, and line counts
whereis	Report all known instances of a command
which	Locate a program file in the user's path.
while	Execute commands
who	Print all usernames currently logged in
whoami	Print the current user id and name (`id -un')
Wget	Retrieve web pages or files via HTTP, HTTPS or FTP
write	Send a message to another user
X	
xargs	Execute utility, passing constructed argument list(s)

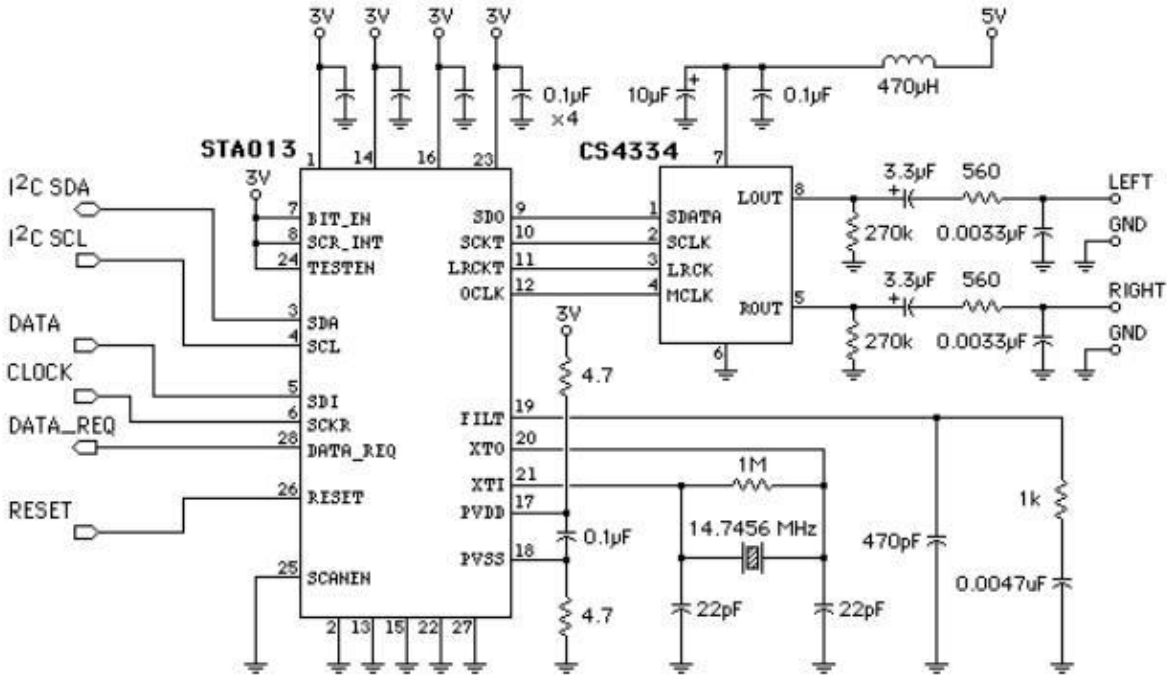
APPENDIX II

A. Schematics

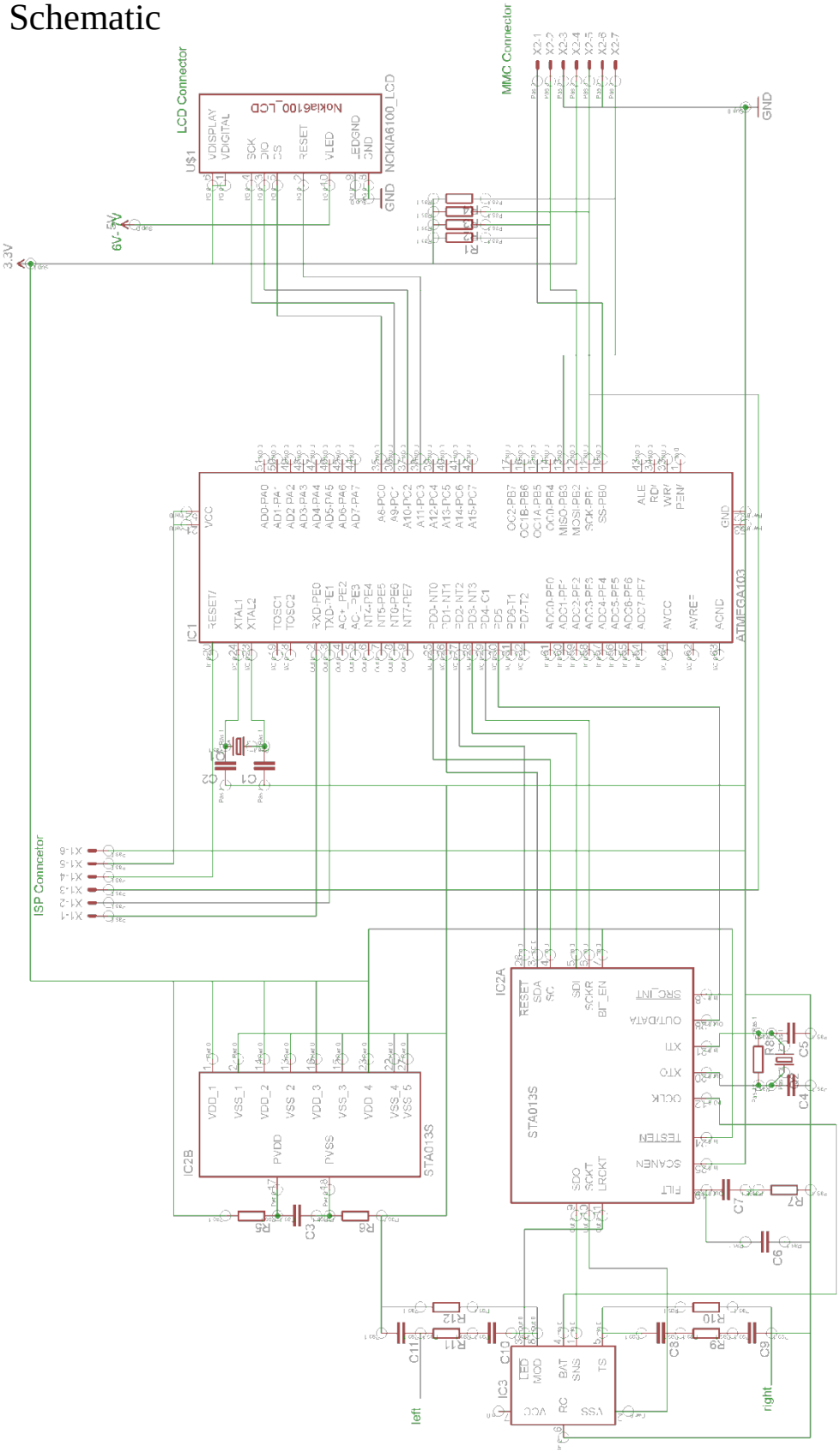
1. Touch Screen Controller



2. STA013 (mp3 Decoder IC) and CS4334 (DAC IC):

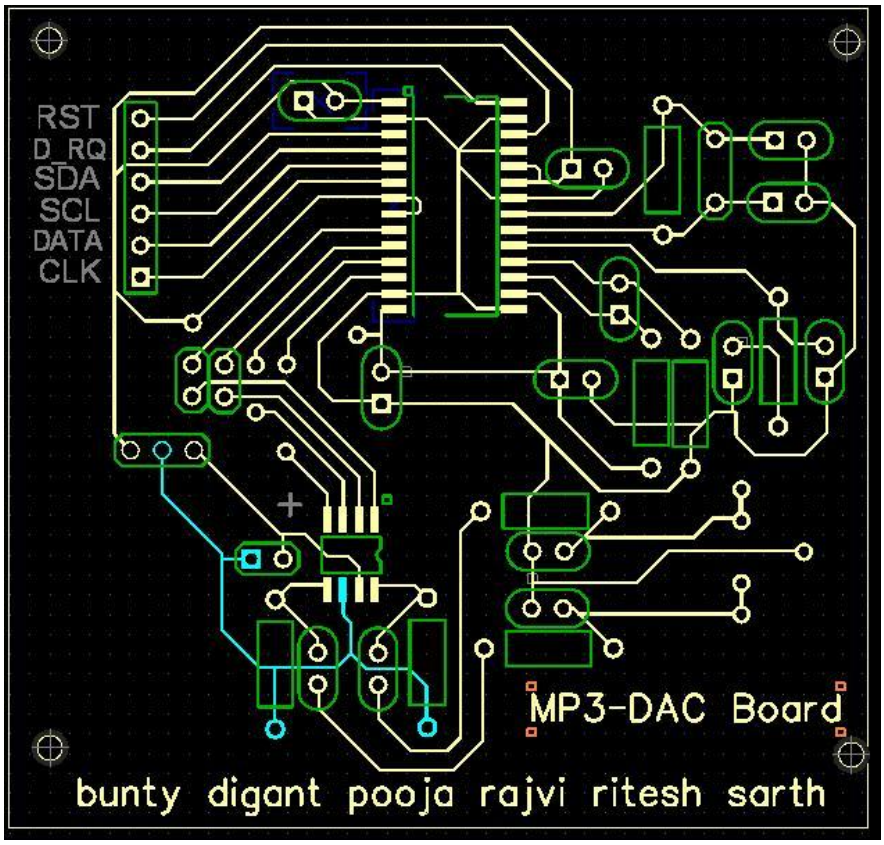


3. Final Schematic

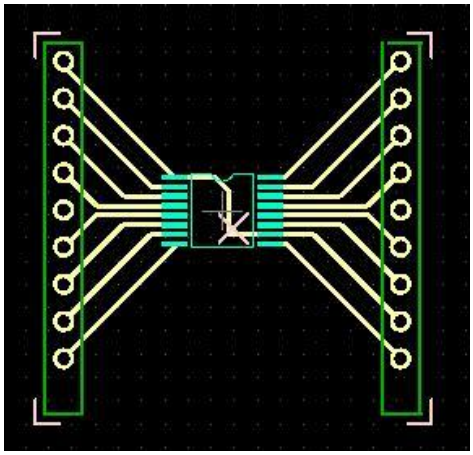


B. Layouts

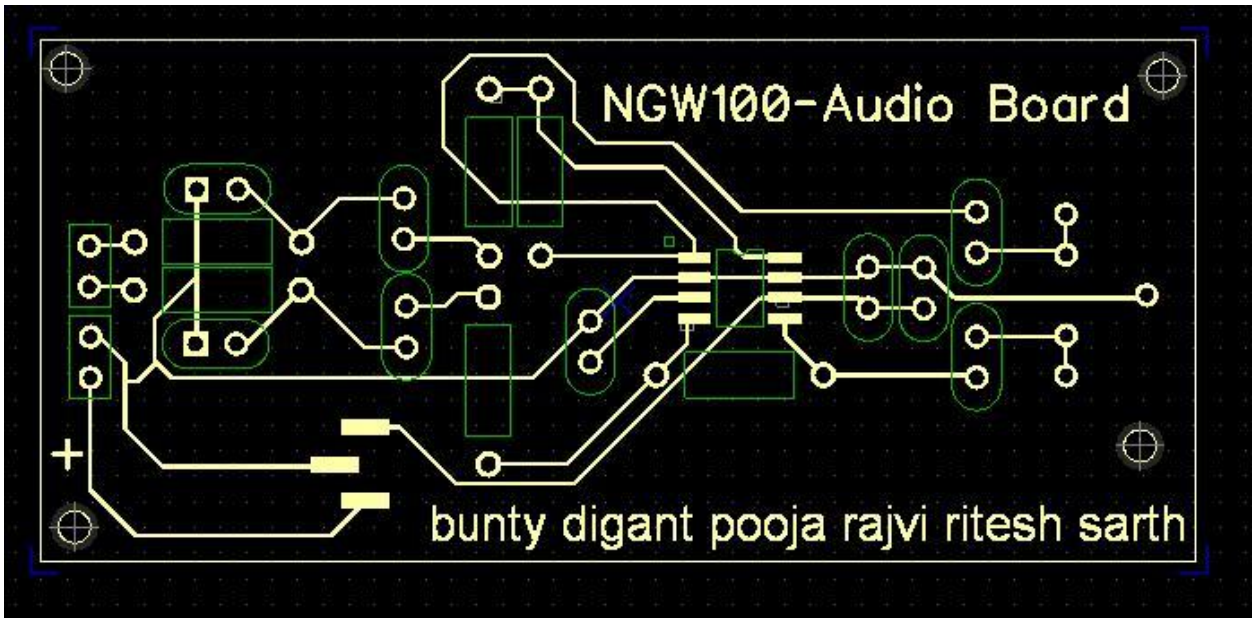
1. Mp3 IC STA013, DAC IC CS4334 Layouts



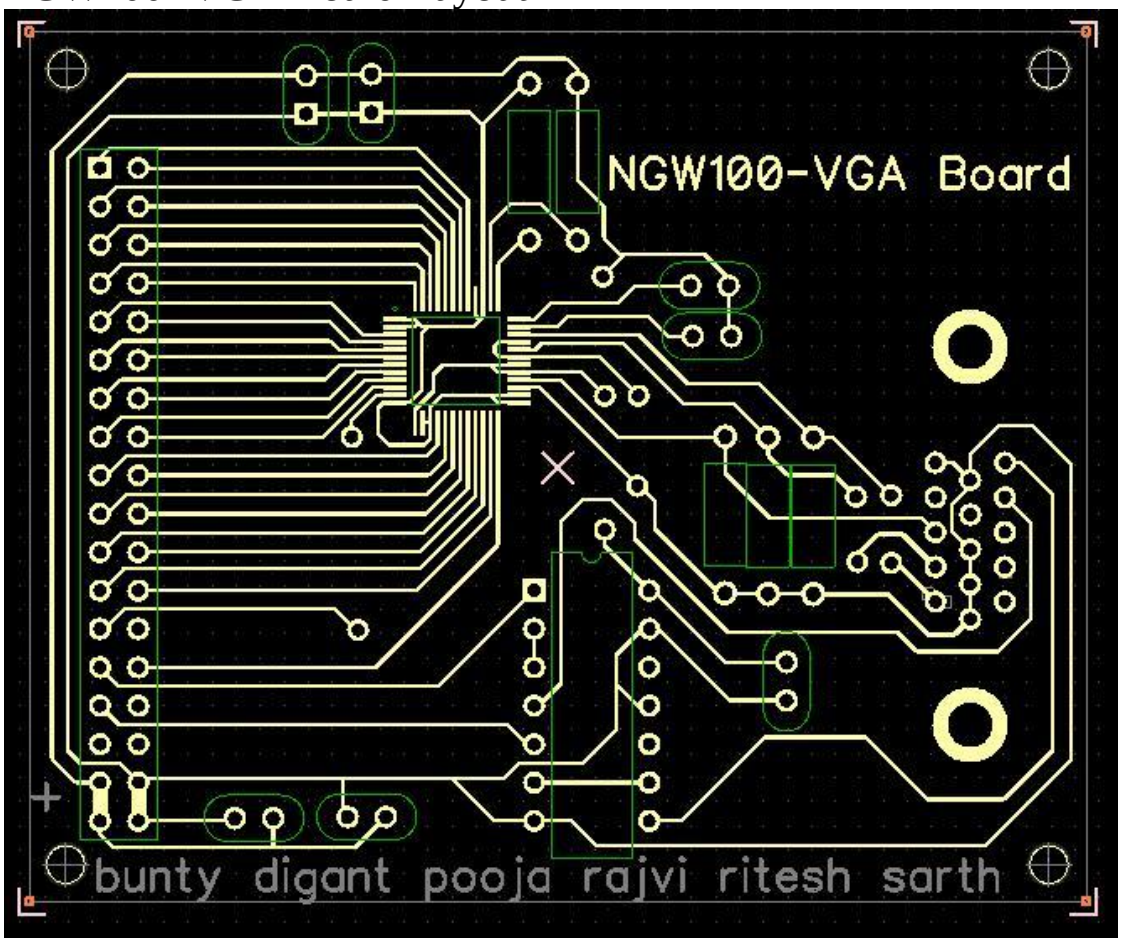
2. Touch Controller ADS7843 IC Layout



3. NGW100- Audio Board Layout



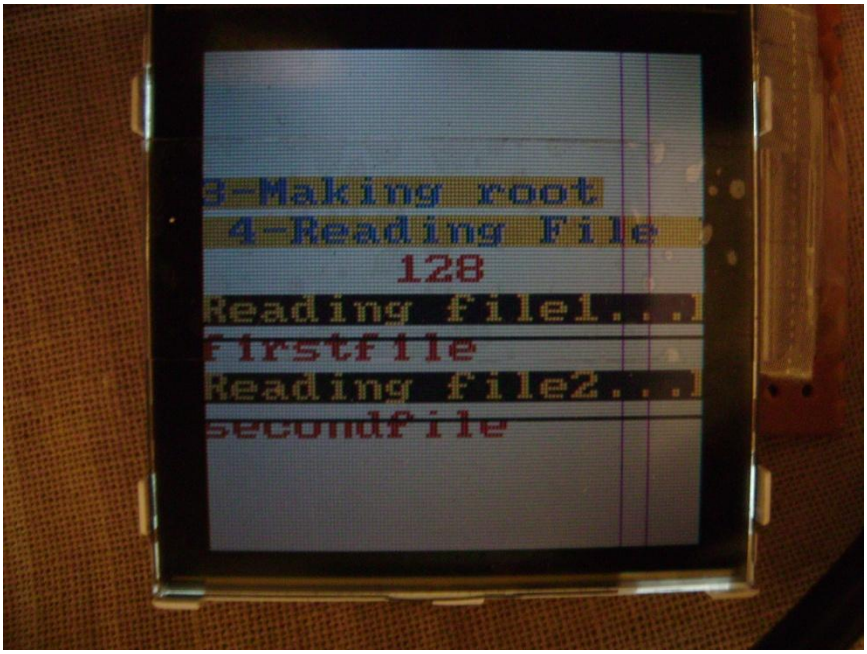
4. NGW100- VGA Board Layout



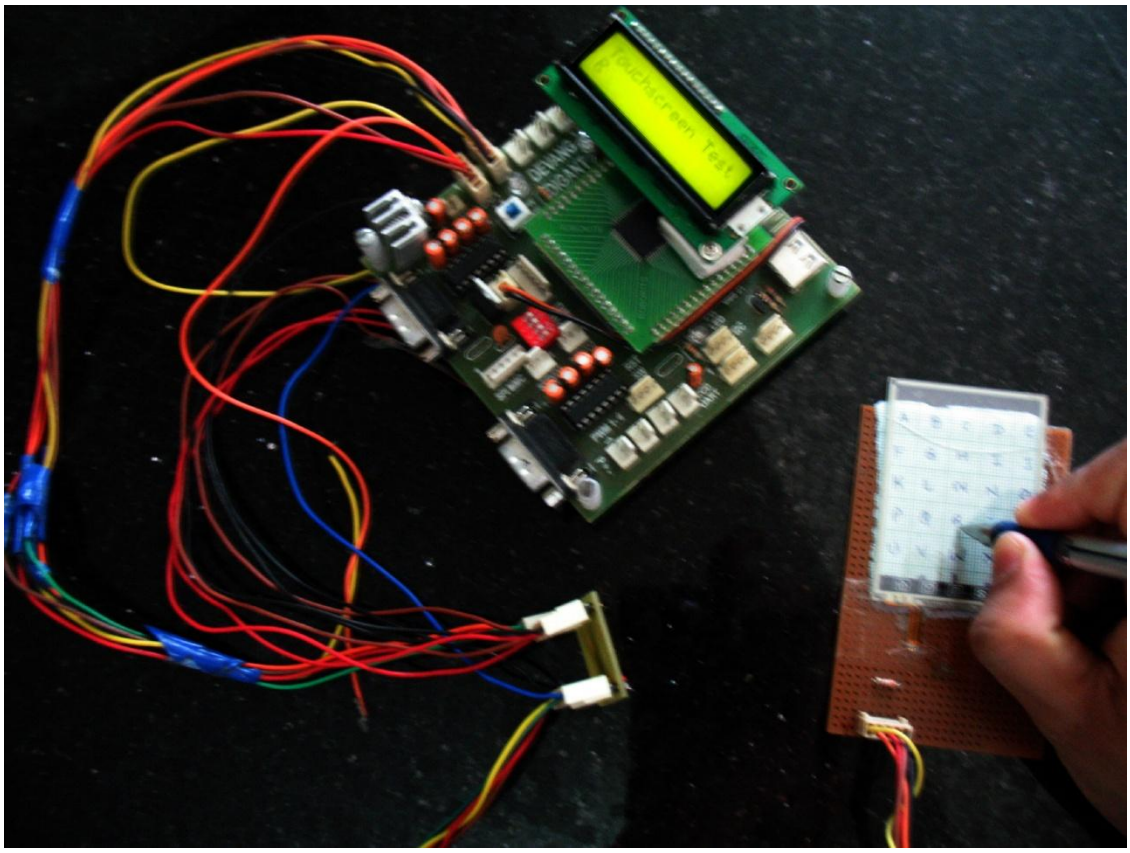
B. Hardware Photos

1. Color LCD Booting

2. Memory Card Interfacing



3. Touch Screen Interfacing



APPENDIX III

Tools Used for the Embedded System Development

1. AVR Studio 4
2. BASCOM- AVR
3. Eagle
4. DipTrace
5. xVi32 Hex Editor
6. Sony Sound Forge
7. Audacity
8. MatLab
9. Ubuntu
10. AVR32 Studio
11. Buildroot
12. Robokits USB Programmer
13. DPKG
14. Turbo C

ABBREVIATIONS

ACK	Acknowledgement
ADC	Analog to Digital Converter
AP	Application Processor
ARM	Advance RISC Machine
CF Card	Compact Flash Card
CODEC	Coder Decoder
CRC	Cyclic Redundancy Check
DAC	Digital to Analog Converter
DMA	Direct Memory Access
DMIPS	Dhrystone Million Instruction Per Second
DSP	Digital Signal Processing
EBI	External Bus Interface
EEPROM	Electrically Erasable Programmable Read Only Memory
FAT	File Allocation Table
FS	File System
HSB	High Speed Bus
I2C	Inter-Integrated Circuit
iPKG	Itsy Package Management System
JFFS	Journaling Flash File System
JTAG	Joint Test Access Group

LCD	Liquid Crystal Display
LSB	Least Significant Bit
MAC	Medium Access Control
MCI	Multimedia Card Interface
MCU	Micro Controller Unit
MII	Media Independent Interface
MIPS	Million Instructions Per Second
MMC	Multi Media Card
MMCA	Multi Media Card Association
MP3	MPEG-1 Audio Layer 3
MPEG	Motion Picture Expert's Group
MSB	Most Significant Bit
OMAP	Open Multimedia Application Processor
OS	Operating System
PCM	Pulse Code Modulation
PDA	Personal Digital Assistants
PDC	Peripheral DMA Controller
PIC	Peripheral Interface Controller
PICO	Pixel Co-Processor
PROM	Programmable Read Only Memory
PWM	Pulse Width Modulation
QVGA	Quarter Video Graphics Array

RISC	Reduced Instruction Set Computer
RMII	Reduced Media Independent Interface
RTC	Real Time Clock
SDC	Secure Digital Card
SDCA	Secure Digital Card Association
SDR	Sample Data Register
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SSC	Serial Synchronous Controller
TCB	Tightly Coupled Bus
TDM	Time Division Multiplexing
TDMI	Thumb Instruction, Debugger, Multiplier, ICE
TFT	Thin Film Transistor
TTL	Transistor Transistor Logic
TWI	Two Wire Interface (I2C)
USART	Universal Synchronous Asynchronous Receiver/Transmitter
USB	Universal Synchronous Bus
VGA	Video Graphics Array
VMU	Vector Multiplication Unit
WDT	Watch Dog Timer

REFERENCES

1. *ATMEL Corp.,* 'ATmega128 Product Manual'.

2. *ATMEL Corp.,* 'AT32AP7000 Product Manual'.

3. 'Interfacing alphanumeric LCD 16x2' , [online document], available at: http://lcdinterfacing.googlepages.com/

4. *James Lynch.,* 'Color LCD Display Driver', [online document], available at: http://gnuarm.alexthegeek.com/lcd/Nokia_6100_LCD_Display_Driver.pdf

5. *Texas Instruments,* 4-Wire and 8-Wire Resistive Touch-Screen Controller Using the MSP430.

6. *Texas Instruments,* ADS7843 Product Manual (September 2000 – Revised May 2002).

7. *Sandisk Corp.,* 'Multimedia Card Product Manual v 1.3'.

8. *Microsoft Corp.,* 'FAT: general overview of on-disk format'- Microsoft Hardware Whitepaper.

9. *ATMEL Corp.,* 'Using the AVR's High Speed PWM- Application Note', [online document], available at: http://www.avrfreaks.net/index.php?module=Freaks%20Tools&func=viewItem&item_id=498

10. *STMicroelectronics,* STA013 Product Manual (2004).

11. *Paul Stoffregen,* 'MP3 Player, Using The STA013 Chip', [online document], available at: http://www.pjrc.com/tech/mp3/sta013.html (Last updated: February 23, 2005).

12. *Thomas Petazzoni,* 'Buildroot Usage and Documentation', [online document], available at: http://buildroot.uclibc.org/buildroot.html (Last updated: March 25, 2009).

13. *Raghvan, Lad et al.,* 'Embedded Linux System Design and Development', Auerbach Publications.

14. *Holtmann, Kleen et al.,* '`git clone`', [online document], available at: http://git.kernel.org/