A REPORT
ON

Featherweight Electronic Book for Rural Education

By

Name                      ID No.(s)
Rajkumar Singh Parihar    2002A3PS013

At

Microsoft Research India, Bangalore
A Practice School II Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE PILANI (RAJ.)

December, 2005
A REPORT
ON

Featherweight Electronic Book for
Rural Education

By
Rajkumar Singh Parihar 2002A3PS013 B.E. (Hons.) Electrical & Electronics

Prepared in the partial fulfillment of the
Practice School II course
BITS C412

AT
Microsoft Research India, Bangalore
A Practice School II Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,
PILANI (RAJ.)

December, 2005
Title of project: Featherweight Electronic Book for Rural Education

ID No.(s)/Name(s): 2002A3PS013 Rajkumar Singh Parihar
Discipline(s) of the Student(s): Electrical and Electronics

Name(s) and Designations(s): Mr. Vibhore Goyal - Assistant Researcher
Dr. Kentaro Toyama - Assistant Managing Director

Of the Expert(s)

Name(s) of the PS Faculty: Brig. M. R. Narayanan

Key Words: One Time Programmability, Content update, 8-Bit RISC Processor, MMC, PWM, Active Low Pass filter, AVR Microcontroller

Project Ares: Trends in Emerging markets, Featherweight Computing, Socio-Economic Development

Abstract:
The constrained features of conventionally implementation Electronic Book by LeapFrog and Fisher-Price were One Time Programmability, Fixed Components, Less and Non Accessible Memory and difficulties and limited scope with new Content update. This simple and cost effective implementation of Featherweight Electronic Book is addressing and incorporating the solutions of various potential problems which have been noticed in similar products. The goal of content development and updating the Book with new content which was the central theme behind developing an Architecture using ATmega 8-Bit RISC Processor has been achieved fully and comprehensibly using MMC card which is compatible with PC and also with a Microcontroller like ATmega-162 using SPI mode of data transfer. PWM Signal Generation capability microcontroller has brought down the cost by skipping the use of DAC for audio generation. Generation of Audio analog wave from Pulse Width Modulated signal has been achieved by a 5th order Active low Pass filter which has been realized using Low power op-amps and Discrete components like resistors and capacitors. The Remarkable features of design are LOW POWER Consumption (working on 3.3 V), COMPACT Size of system and capability of SERIAL Interface through RS-232 Port with PC for Debugging Purpose whenever required. The quality of audio is as good as AM radio.

Signature(s) of Student(s) Signature of PS Faculty
Date: Dec 13th, 2005
**Response Option Sheet**

**Station:** Microsoft Research India  
**Centre:** Bangalore  
**ID No.(s)/Name(s):** 2002A3PS013  
**Rajkumar Singh Parihar**

**Title of project:** Featherweight Electronic Book for Rural Education.

Usefulness of the project to the on-campus courses of study in various disciplines. Project should be scrutinized keeping in view the following response options. Write Course no. and course name against the option under which the project comes.

Refer Bulletin for Course No. and Course Name.

<table>
<thead>
<tr>
<th>Code No.</th>
<th>Response Option</th>
<th>Course No.(s) &amp; Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>A New Course can be designed out of this project</td>
<td>NO</td>
</tr>
<tr>
<td>2.</td>
<td>The Project can help modification of the course content of some of the Existing Course</td>
<td>NO</td>
</tr>
</tbody>
</table>
| 3.      | The Project can be used directly in some of the existing Compulsory Discipline Courses (CDC)/ Discipline courses other than compulsory (DCOC)/Emerging Areas (EA), etc. Courses. | Can be used in one of the EEE - DCOC  
Course No.: **EEE G512**  
Course Title: **Embedded System Design** |
| 4.      | This project can be used in preparatory courses like Analysis and Application oriented Courses(AAOC)/ Engineering Sciences(ES)/Technical Art (TA) and Core Science | NO                                   |
| 5.      | This Project also relates to the Professional Work of the host organization.   | YES                                  |

______________________________  
Signature of Student  
Date: December 13th, 2005  
______________________________  
Signature of PS Faculty
ACKNOWLEDGEMENTS

I am very thankful to my project Mentors, Dr. Kentaro Toyama, Assistant managing Director and Mr. Vibhore Goyal, Assistant Researcher, Microsoft Research Lab, India. The love and faith that they showered on me helped me to go on and complete this project.

I would also like to thank Dr. P. Anandan, Managing Director, Mr. Sean Blagsvedt, Lead Manager of Hardware and Communication Group, Mr. Joseph Joy, Development Manager and all other members of Microsoft Research Lab for their Cooperation, Support and Encouragement.

I also extend my sincere gratitude to Prof. (Dr.) L. K. Maheshwari, Director, BITS – Pilani, Dr. V. S. Rao, PSD Dean, BITS - Pilani for giving me an opportunity to work in such a wonderful organization like Microsoft Research and Prof. (Dr.) Anu Gupta, EEE, BITS - Pilani who has been my constant source of inspiration and motivation.

I also thank to all my Practice school Mates and Brig. M. R. Narayanan Instructor In charge of Practice School Session – II for their wonderful advices and guidance throughout the whole Practice School Session at Microsoft Research Lab India.

Last but not the least I would like to thank my Friends, Parents, Family members and invisible force which provided moral support and spiritual strength, which helped me completing this project successfully.
December 13th, 2005

To Whomsoever It May Concern

This is to certify that Mr. Rajkumar Singh Parihar has successfully completed a project titled “Featherweight Electronic Book for Rural Education” for a period of nearly 6 months i.e. from 4th July to the 14th of December, 2005 under the guidance of Mr. Kentaro Toyama, Assistant Managing Director of our organization under the PS II program of BITS, Pilani.

He has been a valuable contributor to the project.

For Microsoft Research India,

(Kentaro Toyama)
Assistant Managing Director
Microsoft Research India, Bangalore
<table>
<thead>
<tr>
<th>Content</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract:</td>
<td>i</td>
</tr>
<tr>
<td>Response Option Sheet:</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgements:</td>
<td>iii</td>
</tr>
<tr>
<td>Certificate:</td>
<td>iv</td>
</tr>
<tr>
<td>1. Introduction .............................................................................1</td>
<td></td>
</tr>
<tr>
<td>2. Implementation of Electronic book: with ATMEGA-162 .....................2</td>
<td></td>
</tr>
<tr>
<td>2.1. Generic Block Diagram ................................................................2</td>
<td></td>
</tr>
<tr>
<td>2.2. Basic Building Blocks ................................................................3</td>
<td></td>
</tr>
<tr>
<td>2.2.1. Micro-Controller ....................................................................4</td>
<td></td>
</tr>
<tr>
<td>2.2.1.1. AVR Microcontroller ......................................................4</td>
<td></td>
</tr>
<tr>
<td>2.2.1.2. ATmega162 (AVR family) ...................................................5</td>
<td></td>
</tr>
<tr>
<td>• High Performance .................................................................5</td>
<td></td>
</tr>
<tr>
<td>• Low Power Consumption .........................................................5</td>
<td></td>
</tr>
<tr>
<td>• High code density ...............................................................6</td>
<td></td>
</tr>
<tr>
<td>• In-System development ........................................................6</td>
<td></td>
</tr>
<tr>
<td>2.2.1.3. Pin Descriptions ..........................................................7</td>
<td></td>
</tr>
<tr>
<td>2.2.2. Multi Media Card ....................................................................9</td>
<td></td>
</tr>
<tr>
<td>2.2.2.1. PIN Diagram .....................................................................9</td>
<td></td>
</tr>
<tr>
<td>2.2.2.2. Interfacing with ATmega -162 ..........................................10</td>
<td></td>
</tr>
<tr>
<td>2.2.3. Touch panel and Keypad ......................................................11</td>
<td></td>
</tr>
<tr>
<td>2.2.3.1. Theory of Operation .......................................................12</td>
<td></td>
</tr>
<tr>
<td>2.2.3.2. Keypad Structure ............................................................12</td>
<td></td>
</tr>
<tr>
<td>2.2.3.3. Interfacing with ATmega-162 ............................................12</td>
<td></td>
</tr>
<tr>
<td>2.2.3.4. Decoding Algorithm .......................................................13</td>
<td></td>
</tr>
</tbody>
</table>
2.2.4. Active LPF: PWM signal to Audio Analog wave Generation………..14
  2.2.4.1. 5th-Order Low Pass Filter..................................................14
  2.2.4.2. Low Power Single Supply Op-Amp (LM324)........................15
2.2.5. Audio Amplifier.................................................................16
  2.2.5.1. Features of LM386.............................................................16
  2.2.5.2. Interfacing.................................................................16
2.2.6. Speaker............................................................................17
2.3. Prototype of Electronic Book......................................................18
  2.3.1. Electronic book’s user Interaction.........................................18
  2.3.2. Execution of events.........................................................19
  2.3.3. Complete Schematic........................................................20
  2.3.4. Description.....................................................................21
    2.3.4.1. Multi Media and SD Flash Card Interface and Function…….22
    2.3.4.2. Timer PWM generation function....................................23
    2.3.4.3. 4 by 4 keypad Interface function......................................24
  2.3.5. Program Structure............................................................24
3. C – Program: Electronic Book........................................................25
4. Summary.................................................................................31
Appendix:
AD/DA converters.........................................................i
  • Analog Signal.......................................................i
  • Digital Signal......................................................i
PWM (Pulse Width Modulation)........................................ii
  • Basic Concepts....................................................ii
  • PWM to analog Conversion.......................................iii
Wave files format.....................................................iv
AVR programming Tips................................................vi
  • Basic Include Files.............................................vi
  • Basic I/Os.........................................................vi
C-Program.............................................................vii
  • MMC Read........................................................vii
  • FAT Implementation and Reading the LIST of FILES.....ix
  • Key Press Decoding.............................................xii

References.......................................................................xv
1. Introduction:

Electronic Book is the one among few devices which can play an important role to surge the goal of rural education rapidly which is crucial for developing countries like India. The intuition and motivation behind developing such products like “Featherweight Electronics Book” is to provide a particular kind of awareness to the particular age group people. Cost effectiveness, High reliability and Easy availability are some of the Secondary goals which can assist this device to become a global accessory. Same time it should be also user friendly and should require a very less knowledge about how to operate such devices. The reason behind this is very simple and straight forward. As these products aims to the backward communities, children and illiterate adults who have a very little awareness about the external world. Less complexity in understating the interaction will provide an ample opportunity in success and popularity. There are numerous alternatives by which the same goal can be achieved. This is one such attempt to prototype an Electronic Book using embedded system techniques.

Two educational toy companies in the United States (LeapFrog and Fisher-Price) have electronic books on the market in which a paper book provides the visible interface and a pen or finger interface allows children to hear audio feedback from the book. These devices fit some criteria for a rural education device perfectly – they don’t require literacy to use, they have extremely simple interfaces, they’re physically robust, and they’re inexpensive. They are very specifically for a particular age group of learners. Once the content embedded in book there is a very small possibility to change that. This design is also an attempt to incorporate the possibility of changing and adding new content to the existing book through some external device like PC.

The main work done is to create the flexible content and technique by which it can be loaded to device over time so that after some point of time as user grows still it should be able to provide him some knowledge which is relevant for him.
2. Implementation of Electronic book: with ATMEGA-162

2.1. Generic Block Diagram:
According to simple architecture plenty of wave files are stored in Memory (MMC). Address of a particular wave file will be decided by the decoded value after a key press of 4x4 keypad. To generate the particular address of wave file which has to be played we may require some more information incase where more Book can also have flexibility in terms of various exercise like spell checking, Pronunciation, small games etc.

Figure 1: Generic Block Diagram

The above implementation is fairly complex and costly. I have chosen a simple architecture with atmega-162 and very few external devices which is quite simple and utilizes very simple concepts of electronic circuits.
2.2. Basic Building Blocks:

After considering a number of possibilities finally we decided the following components which are more advantageous than others in because of these reasons.

- Cheap
- Easily available
- Easy interfacing
- Easy programming for configuration

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Functional Block</th>
<th>Component Used</th>
<th>Chip or Part No.</th>
<th>Cost (INR) per Part Approx.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Micro-controller</td>
<td>8-Bit RISC</td>
<td>ATmega162</td>
<td>60/-</td>
</tr>
<tr>
<td>2.</td>
<td>Memory</td>
<td>MMC Card</td>
<td>128MB, NCP</td>
<td>200/- (16MB)</td>
</tr>
<tr>
<td>3.</td>
<td>5\textsuperscript{th} Active LPF</td>
<td>LP Op-Amp, R,C</td>
<td>LM324</td>
<td>10/-</td>
</tr>
<tr>
<td>4.</td>
<td>Audio Amplifier</td>
<td>Low Power Amp.</td>
<td>LM386</td>
<td>10/-</td>
</tr>
<tr>
<td>5.</td>
<td>Speaker</td>
<td>8-ohm Speaker</td>
<td></td>
<td>20/-</td>
</tr>
<tr>
<td>6.</td>
<td>Touch PAD</td>
<td>4x4 Key Pad</td>
<td></td>
<td>20/-</td>
</tr>
</tbody>
</table>

Total → 320 (INR)

The tasks which this Architecture is able to do are following:

- Store the Files in wav format in MMC through PC.
  Wav Format Specifications
  1. Bit Rate : 64kbps
  2. Audio Sample Size: 8 bit
  3. Channels : 1 (mono)
  4. Audio Sample Size : 8 KHz
  5. Audio Format : PCM
- Sending data from MMC to of the MuC Buffer.
- Receiving Data from key Pad to MuC.
- Generate the PWM signal corresponding to sample values which have been received from MMC.
2.2.1. Micro-Controller:

A microcontroller is an integrated circuit (IC) that is programmable and contains all the components comprising a controller. When one turns on the power to the microcontroller it goes through a series of commands. These commands are put in the chip by users. Different tasks can be performed by changing the commands (usually called the program). To change the commands or program we need a device which lets one download the program from computer to the microcontroller in HEX or Binary format. Microcontrollers are very rich in terms of components and basic building blocks attached with them. Typically, they include a CPU, RAM, some form of ROM, I/O ports, and timers.

Unlike a general-purpose computer, which also includes all of these components, a microcontroller is designed for a very specific task – to control a particular system. As a result, the parts can be simplified and reduced, which cuts down on production costs. A microcontroller is a computer-on-a-chip optimized to control electronic devices. It is a type of microprocessor emphasizing self-sufficiency and cost-effectiveness, in contrast to a general-purpose microprocessor, the kind used in a PC. A typical microcontroller contains all the memory and I/O interfaces needed, whereas a general purpose microprocessor requires additional chips to provide these necessary functions.

2.2.1.1. AVR Microcontroller:

AVR® microcontrollers have a RISC core running single cycle instructions and a well-defined I/O structure that limits the need for external components. Internal oscillators, timers, UART, SPI, pull-up resistors, pulse width modulation, ADC, analog comparator and watch-dog timers are some of the features one can find in AVR devices. AVR instructions are tuned to decrease the size of the program whether the code is written in C or Assembly. With on-chip in-system programmable Flash and EEPROM, the AVR is a perfect choice in order to optimize cost and get product as quickly as needed.
2.2.1.2. ATmega162 (AVR family):

The ATmega162 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega162 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed. ATmega162 is an 8-bit RISC core AVR controller with 16KB of flash memory. Here is the list of key features of AVR controllers. This is to understand why we choose to work on them.

1. High Performance
2. Low power consumption
3. High code density
4. In-System development

1. High Performance:

AVR controllers have RISC core. They can operate with clock rates up to 20 MHz, achieving close to 20 MIPS. With 32 general purpose registers, the AVR delivers unmatched performance and flexibility, especially when you program in high-level languages, like C, Pascal or Basic.

2. Low Power Consumption:

A wide operating range (1.8V-5.5V) ensures great flexibility when it comes to power supplies. AVR Flash microcontrollers have up to six different sleep modes. This assures that between real-time events the better part of the AVR does not consume power while ensuring the fastest possible wake-up time even when using an external crystal. Software controlled frequency makes it possible to use maximum speed when needed, saving power the rest of the time.
3. High code density:

High code density ensures that a minimum of instructions and clock cycles are needed to execute a task, thereby reducing the power consumption significantly. Even if you write your application in high-level languages like C they can fit into small flash memory. AVR is optimized for high level programming languages such as C and offers an unmatched code density. Low cost On-Chip Debug tools are also available and the whole tool chain is an easy and efficient development environment to designers.

4. In-System development:

In-System programming allows programming and reprogramming of any AVR Microcontroller positioned inside the end system. Using a simple 3-wire SPI interface, The In-System programmer communicates serially with the AVR microcontroller, reprogramming all nonvolatile memories on the chip. In-System programming eliminates the physical removal of chips from the system. This will save time, and money, both during development in the lab, and when updating the software or parameters in the field.

Apart from these features it can programmed in system. It supports In-System programming and debugging. And they have lock-bits to protect code from being inappropriately read or copied. The AVR devices also support self-programming, allowing them to upgrade their own firmware. AVR® Flash microcontrollers - a large family of processors that share a single core architecture. This makes it easy to reuse code in next project, and the one after that with devices handling from 1 Kbyte to 256 Kbyte of code and packages with 8 to 100 pins.
2.2.1.3. Pin Descriptions: Below is the most Common use of all pins of Atmega162 with Basic interfacing.

- **VCC**: Digital supply voltage
- **GND**: Ground
- **Port A (PA7--PA0)**: Port A is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated.

- **Port B (PB7 --PB0)**: Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated.
- **Port C (PC7 --PC0):** Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PC7(TDI), PC5(TMS) and PC4(TCK) will be activated even if a Reset occurs.

- **Port D (PD7 -- PD0):** Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

- **Port E (PE2 --PE0):** Port E is an 3-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port E output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port E pins that are externally pulled low will source current if the pull-up resistors are activated.

- **RESET:** Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset.

- **XTAL1:** Input to the Inverting Oscillator amplifier and input to the internal clock operating circuit.

- **XTAL2:** Output from the Inverting Oscillator amplifier.
2.2.2. Multi Media Card:

Multimedia Cards are one of the smallest Flash cards available, about the size of a postage stamp. A Multimedia Card (MMC) is a tiny memory card that uses flash memory to make storage portable among various devices. Multimedia Cards are much more rugged than traditional storage media. MMC features encryption capabilities for protected content, to ensure secure distribution of copyrighted material. High Speed SD and miniSD cards offer a combination of high storage capacity, great flexibility and security. As an extension of the existing SD card standard, the miniSD card allows cell phone and consumer electronics developers to design much more compact devices.

2.2.2.1. PIN Diagram:

![MMC pin out Diagram](image)

Figure 3: MMC pin out Diagram

<table>
<thead>
<tr>
<th>PIN #</th>
<th>NAME</th>
<th>Type / Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CS</td>
<td>IN</td>
<td>Chip Select</td>
</tr>
<tr>
<td>2</td>
<td>DI</td>
<td>IN</td>
<td>DATA In</td>
</tr>
<tr>
<td>3</td>
<td>VSS</td>
<td>S</td>
<td>Supply Voltage Ground</td>
</tr>
<tr>
<td>4</td>
<td>VDD</td>
<td>S</td>
<td>Supply Voltage</td>
</tr>
<tr>
<td>5</td>
<td>SCLK</td>
<td>IN</td>
<td>Clock</td>
</tr>
<tr>
<td>6</td>
<td>VSS2</td>
<td>S</td>
<td>Supply Voltage Ground</td>
</tr>
<tr>
<td>7</td>
<td>DO</td>
<td>OUT</td>
<td>DATA Out</td>
</tr>
</tbody>
</table>
2.2.2.2. Interfacing with ATmega-162:

It is easy to interface a MMC (Multimedia Card) with an Atmel ATmega162 (AVR series) via the SPI (Serial Port Interface). The MMC is connected to the SPI pins of the ATmega16 via simple resistor voltage dividers to transform the +5V high levels to about 3.3V used by the MMC. If the Atmega-162 is working on 3.3 V power supply then all the MMC pins can be directly connected to Microcontroller (as in this design). The data-out pin from the MMC goes directly to the ATmega162, because 3.3V is high for the ATmega162 anyway. The schematic of the MMC interfacing is given below.

![MMC Atmega-162 Interfacing](image)

Figure 4: MMC Atmega-162 Interfacing
2.2.3. Touch panel and *Keypad*:

Infrared touch panels are designed to fit most flat panel display technologies. The touch panels are a matrix of infrared diodes and detectors, scanning electronics, micro-processor controller, bezel/optical filter assembly and hardware to mount the touch panel and controller to the flat panel display. They have RS-232 interface through which it can send the coordinates as in where it is touched. So these sorts of touch panels with in-built controller are easy to interface with ATmega162.

The keypad supposed to be most widely used I/O devices of the microcontrollers. This one at the lowest level is organized in rows and columns. The controller can access both the columns and rows through the I/O ports.

In the present design a touch sensitive panel has been replaced by a 4x4 keypad. This particular keypad is easily available and far cheaper than touch sensitive pad. It is also fruitful to learn the decoding techniques of mechanical switches by the use of microcontrollers. These kind of simple keypad consist of n by n keys with n column and n row connections. The popular ones are 4x4 and 8x8. Basic operation and interfacing with microcontroller is similar. A 4X4 matrix keypad is shown below.

These kind of keypad don’t have power supply in them so they need to be decoded using a microcontroller as to which key is pressed. This is one of the very Basic keypad whose operation is based on the contact between row and column wires.

![4x4 Keypad Diagram](image-url)

*Figure 5: 4 x 4 Keypad*
2.2.3.1. Theory of Operation:

There are various ways to decode these kinds of keypads as to which key has been pressed at the current state. Seemingly all of them are same with a minor difference. The decoding technique presented here has a slight variation with mostly used previously by a significant number of designers.

2.2.3.2. KEYPAD STRUCTURE:

There are basically 4 row wires and 4 column wires in a 4x4 keypad. All the Row wires are connected to one of the port of Microcontroller and column wires to any other Port which is for general purpose. In this particular case (4x4 keypad) all the 8 wires can be connected to even a single port, but it will close the doors for future possibility of any kind of extension in connections. Keeping this thing in mind the PORTC has been dedicated to column and PORTA to row lines. Whenever a key is pressed the key makes a contact with that particular row line to Column line and appropriate dominant voltage appears across the row or column.

2.2.3.3. INTERFACING WITH ATMega-162:

![Diagram of Atmega-162 and 4 by 4 Keypad interfacing]

Figure 6: Atmega-162 and 4 by 4 Keypad interfacing
2.2.3.4. DECODING Algorithm:

All the column lines are pulled high through appropriate resister (typically 20K). The column port (PORTC) has been configured as input port and row port (PORTA) as an output port.

- All the row lines have been pulled low through microcontroller by sending 0x00 to that port.
- The data coming from PORTC will be always 0xFF when no key is pressed. As a key press will take place immediately the column lines which were 0xFF will no more be same. At least one of them will go LOW because all the any of the one row will make contact to any one of the column line.
- As soon as data scanned through column port is other than 0xFF, the value of current column lines will be stored in one 8-Bit long variable and simultaneously one of the Rows will be made HIGH through microcontroller.
- Just after sending the data to row lines the column line will be scanned if they are not all high the other row line will be made high and as all the column line go HIGH the value sent through Row port will be stored in some other 8-Bit wide variable.
- The combination of these two 8-Bit word (we will call them ROW word and COLUMN word) will give a unique key detection which can be mapped into n by n unique keys as per the connivance.

For Example: let the column is always 0xFF and rows are always 0x00. Someone has pressed a key ‘7’ then 3rd row and 3rd row will make a contact. In this case the words will be 0x02 (Row word) and 0xFD (Column word) and the n * n unique combination will help to recognize the key press.
2.2.4. Active LPF: PWM signal to Audio Analog wave Generation:

A sampled sound system may require low-pass filtering to remove the harsh, hissing sound of high-frequency switching transients caused by the transition from one sample to the next. As with a PWM output, a high sample rate makes filtering easier. If someone using a PWM output already, the low-pass filter that averages the variations on duty cycle of the PWM pulse train will also even out the steps in the sample values.

A simple RC circuit, fourth-order low pass filter and audio amplifier are used to convert the PWM signal into audio signal. The output sound quality is as good as an AM radio. The audio generation circuit consists of a RC circuit, four-order low pass filter, audio amplifier and speaker. The RC circuit and Low Pass Filter is used for harmonics and carrier frequency generated by PWM signals. The audio amplifier will drive the speaker to generate the audio.

![LPF stage in audio generation](image)

**Figure 7: LPF stage in audio generation**

### 2.2.4.1. 5th-ORDER LOW PASS FILTER

Figure shows the schematic of the low pass filter. This four-order low pass filter is consisted of LM386 Low Power Op-Amp. It removes most of the harmonic generated by the PWM output waveform. The cut-off frequency of the circuit is about 21kHz.
2.2.4.2. Low Power Single Supply Op-Amp (LM324):
The design requirement is single power operation and low supply voltage as well. This IC consists of four independent, high gain, internally frequency compensated operational amplifiers. They operate from a single power supply over a wide range of voltages. Operation from split power supplies is also possible and the low power supply current drain is independent of the magnitude of the power supply voltage.

![5th Order LOW Pass Filter](image)

Figure 8: 5th Order active low pass filter
- R values are in K ohm and C value in pF.

![LM324 Pin out](image)

Figure 9: LM324 Pin out
2.2.5. **Audio Amplifier:**

Audio Amplifier is a circuit with one or more stages designed to amplify the audio frequency range (about 20 to 20,000 Hz). The output signal at the output of low Pass filter stage is too week to drive a 8-ohm speaker. To drive a typical 8 – ohm speaker the rating is 500 mWs to 1W. Audio amplifiers are specially designed circuit which can meet the requirement to drive a speaker.

The LM386 is a power amplifier designed for use in low voltage consumer applications. The gain is internally set to 20 to keep external part count low, but the addition of an external resistor and capacitor between pins 1 and 8 will increase the gain to any value from 20 to 200. The inputs are ground referenced while the output automatically biases to one-half the supply voltage.

### 2.2.5.1. Features of LM386:

1. Wide supply voltage range: 4V–12V or 5V–18V
2. Voltage gains from 20 to 200
3. Self-centering output quiescent voltage
4. Low distortion: 0.2% (AV = 20, VS = 6V, RL = 8W, PO = 125mW, f = 1kHz)

### 2.2.5.2. Interfacing:

![Audio Amplifier Stage and basic connections](image)

Figure 10: Audio Amplifier Stage and basic connections
2.2.6. Speaker:

To hear the full richness of sound from our electronic devices, we have to move air so our ears respond to the frequencies in the range we can interpret, roughly 20-20,000Hz. To move the air is the job of the loudspeaker. For low frequencies, we use a large speaker cone called a woofer and for high frequencies we use a smaller one called a tweeter. This is because the cone which works best for low frequencies does not work best for high frequencies and vice versa. Mid-range single speakers are good for voice range frequencies but not so good for booming bass or high pitched sounds. Loudspeakers are analog devices and use an electro-mechanical process to try to reproduce sound.

Figure 11: 8-ohm Speaker

A light voice coil is mounted so that it can move freely inside the magnetic field of a strong permanent magnet. The speaker cone is attached to the voice coil and attached with a flexible mounting to the outer ring of the speaker support. Because there is a definite "home" or equilibrium position for the speaker cone and there is elasticity of the mounting structure, there is inevitably a free cone resonant frequency like that of a mass on a spring. The frequency can be determined by adjusting the mass and stiffness of the cone and voice coil, and it can be damped and broadened by the nature of the construction, but that natural mechanical frequency of vibration is always there and enhances the frequencies in the frequency range near resonance. Part of the role of a good enclosure is to minimize the impact of this resonant frequency.
2.3. Prototype of Electronic Book:

Electronic book is a device looking very similar to a regular book but equipped with some excellent capabilities also. Generally a typical Electronic book provides a simple text UI (user interface) to users which may contain awareness oriented information for a particular age group. For example alphabets, poems and stories for kids, health awareness for adults and women etc. Just below the text a efficient sensing mechanism is installed in such a way so that whenever users interact with the book (Interaction means touches the text written on Pad with fingers or some conducting device i.e. stylus based on Magnetic effect or capacitance changing) a signal goes to central Controlling unit (in this case microcontroller) which will determine the correct address of audio information stored related with the text in memory device attached with it with the help of data which was sent from sensing element array(generally touch pad but 4x4 keypad in this case).

2.3.1. Electronic book’s user Interaction: (External: for users)

![Diagram of User Interaction and Responses from Device](image)

Figure 12: User Interaction and Responses from Device
2.3.2. Execution of events: (Internal: System Level)

Assumptions:

- All the accessories required for design (touch pad, Memory device and clock generation circuits) are interfaced with microcontroller in right manner with individual power supplies and grounds.

- In case of Re programmable Microcontrollers or CU (AVR RISC processors or FPGA) the program for the application is pre compiled and burnt in proper format with all required calibrations in the CU using appropriate loading tool.

- The program and microcontrollers support the interfacing with external devices.

Figure 13: Flow Chart of System Events
2.3.3. Complete Schematic:

![Complete Schematic of the Electronic Book](image)

Figure 14: Complete Schematic of the Electronic Book

The value which are not mentioned are according to following convention

- all the R values are in K-ohm
- all the C values are in pF
2.3.4. Description:

This design application is based on some basic tasks which will be explained step by step in this description. The sub tasks can be performed independently to each other and after verification of proper functioning merging with parameters considerations will make it a complete application. (Complete code written in C is attached in appendix section)

- Multi Media and SD Flash Card Interface and Function
- Timer PWM generation function
- 4 by 4 keypad Interface function

Micro-controller Atmega-162 is the central unit used in this design whose jobs in sequence is Detect and decodes the key press and store the value in an 8-bit wide variable. With the help of this value the name, size and starting cluster of mapped file will be read from Root Directory entry sector. After getting the starting cluster the data of file which are in form of PCM audio sample will be read from DATA sector of MMC to the 512 Byte buffer of Atmega-162. Using PWM techniques of timer one byte will be read from the buffer and corresponding PWM signal will be generated at the output of timer. As soon as the timer finishes 512 byte next sector will be read to the buffer whose address can be get from FAT table. In fact here the file data is arranged in chuck of 4 sectors known as cluster. Each time the size of file will be checked. As the end of file reaches the process stops and microcontroller look for a key press.

The PWM signal is a digital signal with varying duty cycle. The analog sample values are the parameters which cause the variation in duty cycle. For example in 8 – Bit PCM a high sample value around 250 will make the duty cycle close to 100% and 128 close to 50%. This high frequency wave has to be filtered out with LOW PASS FILTER to get the analog audio waveform. Feeding of this analog audio waveform to 8-ohm Speaker will generate the audio voice.
2.3.4.1. Multi Media and SD Flash Card Interface and Function

Accomplishing the communication between microcontroller and MMC card we need to know about the Protocols supported by devices and interface. Here for MMC we are using SPI interface. All the SPI commands and functions are written into spi.c library file.

Including of this file to source code will enable us to perform MMC related task.

To work with this library, the card must be wired to the SPI port of the Atmel microcontroller as described below. Typical cards can operate at up to 25MHz maximum SPI clock rate (thus faster than most AVR's maximum SPI clock rate).

```
/  1 2 3 4 5 6 7 8           |<- view of MMC/SD card looking at contacts
/ 9                                  |Pins 8 and 9 are present only on SD cards
|    MMC/SD Card        |
/YYYY/YYYY/YYYY/YYYY/YYYY\n
1 - CS : (chip select) - wire to any available I/O pin(*)
2 - DIN: (data in, card<-host) - wire to SPI MOSI pin
3 - VSS: (ground) - wire to ground
4 – VDD: (power, 3.3V only) - wire to power (MIGHT BE 3.3V ONLY!)
5 – SCLK: (data clock) - wire to SPI SCK pin
6 – VSS: (ground) - wire to ground
7 – DOUT: (data out, card->host) - wire to SPI MISO pin
```

Function used for this application form this library and their brief description.

```c
void mmcInit( void )
Initialize AVR<->MMC hardware interface. Also prepares hardware for MMC access.

u08 mmcRead ( u32 sector, u08 * buffer )
Read 512-byte sector from card to buffer Returns zero if successful.

u08 mmcReset ( void )
Initialize the card and prepare it for use. Returns zero if successful.
```
2.3.4.2. **Timer PWM generation function**

As it is universally known that PWM signal can be generated using a simple running timer with interrupts and a comparator. That’s why it’s easy to implement PWM in microcontroller applications. The sample frequency will decide the timer top count. As counter will reach to this value it will be rolled back and a interrupt will be generated which will take next byte from buffer. (A detailed description can be getting in AVR Atmal application notes.)

Function used from timer.h library for PWM generation.

```c
void timer1PWMInit ( u08 bitRes )

Enter standard PWM Mode on timer1.

Parameters: bitRes indicates the period/resolution to use for PWM output in timer bits. Must be either 8, 9, or 10 bits corresponding to PWM periods of 256, 512, or 1024 timer tics
```

```c
void timer1PWMInitICR ( u16 topcount )

Enter PWM Mode on timer1 with a specific top-count value.

Parameters: topcount indicates the desired PWM period in timer tics. Can be a number between 1 and 65535 (16-bit)
```

```c
void timer1PWMAOn (void)

Turn on/off Timer1 PWM outputs. Turn on timer1 Channel A (OC1A) PWM output
```

```c
void timer1PWMOff (void)

Turn off all timer1 PWM output and set timer mode to normal.
```

```c
void timerAttach (u08 interruptNum, void(*userFunc)(void))

Attach a user function to a timer interrupt.
```

```c
void timerDetach (u08 interruptNum)

Detach a user function from a timer interrupt.
```
2.3.4.3. **4 by 4 keypad Interface function**

The keypad decoding will need basic set and reset port commands which are defined in io.h library. In the AVR-GCC C language, ports can be accessed using two kinds of commands:

- `inb()` and `outb()` - in-byte and out-byte
- `cbi()` and `sbi()` - clear-bit and set-bit

```
outb(DDRC, 0xFF);  // set all port A pins to input
a = inb(PINA);     // read the input state of all pins on port A
```

2.3.5. Program Structure:

A typical program for this application will have the following components:

1. Include all the library files
2. MMC initiation
3. FAT16 Implementation
4. Keypad press detection function
5. As key press occurs get the first entry’s (name, size and starting cluster) for Root Sector
6. Check for valid root entry and Phony entry incase of Long file name
7. Read the first cluster sector by sector
8. Reading one sector generate the PWM signal corresponding to each byte
9. After finishing current cluster get the address of next cluster form FAT sector and read the cluster sector by sector
10. Check the size of file and data read from disk
11. Keep reading the cluster until the data read becomes equal to file size.
3. C – Program: Electronic Book

// Project : Electronic Book for rural Education
// File Name : ebook.c
// Target MCU : Atmel AVR series ATmega 162
// Function : 1. MMC initialization, Reset;
// 2. fat16 implementation;
// 3. 4x4 Keypad decoding
// 4. wav File Read
// 5. PWM signal generation
// 6. timer Interrupt
// Revision History:
// When Who where
// --------- -------------- -----------------------
// 25-NOV-2005 Rajkumar Parihar MSR India

#include <avr/io.h> // include I/O definitions
#include <avr/signal.h> // include "signal" names (interrupt names)
#include <avr/interrupt.h> // include interrupt support
#include "global.h" // include our global settings
#include "uart2.h" // include uart function library
#include "rprintf.h" // include printf function library
#include "timer.h" // include timer function library (timing, PWM, etc)
#include "spi.h" // include SPI interface functions
#include "mmc.h" // include MMC card access functions
#include "fat.h"

//FAT implementation Specification
#define HideSec 32
#define BootSec 1
#define Fat1Size 245
#define Fat2Size 245
#define RootDisp 32
#define StclusDisp 0x1A
#define FatSec 38
#define RootSec 528
#define MRDE 512

//Function declarations
void myPWMtest(void);
void overflow(void);
void fatTest(void);
u8 keypadTest(void);

//global variables
u8 buffer[0x200];
volatile u16 tempc=0;

//----- Begin Code -----------------------------------------------
int main(void)
{
    // initialize the UART (serial port)
    uartInit();
uartSetBaudRate(0,38400);
// make all rprintf statements use uart for output
rprintfInit(uart0SendByte);
sbi(DDRD, 5);
outb(OCR1AH, 0x00);

fatTest();
return 0;
}

void fatTest(void)
{

unsigned char d;
int i,temp=0;
int Fatadd1=0,Fatadd2=0;
u16 tempa;
u32 tempb,sector=0;
u08 k=0xF0;
u08 j,m,l=0x00;
//u08 itre=0;

// initialize MMC interface
mmcInit();
mmcReset();
d = fatInit(buffer);
rprintf("welcome to file system");

// testing loop
while(1)
{
    k=0xF0;
    //rprintf("start of while loop: ");
    k = keypadTest();
    //rprintf(" k ==> %d ",k);
    if(k != 0xF0)
    {
        rprintf("you are reading * %d * file . file name is --> ",k);
        //reading the information of frist file
        m=0x00;
        j=0x00;
        l=0x00;
        //checking the file location
        for(m=0;m<32;m++)
        {
            mmcRead(RootSec+m,buffer);
            for(j=0;j<16;j++)
            {
                if((buffer[11+j*RootDisp] & 0x08)==0)
                {
                    l++;
                    if (l==(k+1))
                    {
                    }
                }
            }
        }
    }
}

}
l=100;
break;
}
}
}
}
if(l==100)
break;
}

// reading actual file
mmcRead(RootSec+m, buffer);

for(temp=0; temp<8; temp++)
    rprintf("%c", buffer[j*RootDisp+temp]);
// here change the no.
rprintf("location *! %d *", j+RootDisp*m);
temp=26+RootDisp*j;
tempa=((u16 *)&buffer[temp]);
// rprintf("Starting Cluster: %d ", tempa);
tempb=((u32 *)&buffer[temp+2]);
// rprintf("Original file size: %d ", tempb);
// address of first chunk of data from starting cluster
sector=RootDisp+RootSec+(tempa-2)*4;

for(i=0; i<4 && tempb>0; i++, tempb=tempb-512)
{// reading and displaying first cluster
    mmcRead(sector+i, buffer);
    // for(temp=0; temp<512; temp++)
    // rprintf("%d", buffer[temp]);
    myPWMtest();
}
// rprintf("file size left: %d ", tempb);
while(tempb>0)
{
    Fatadd1= (u16)(tempa)/256;
    Fatadd2=((tempa)%256)*2;
    sector=FatSec+Fatadd1;
    mmcRead(sector, buffer);
    tempa=((u16 *)&buffer[Fatadd2]);
    sector=RootDisp+RootSec+((tempa)-2)*4;

    for(i=0; i<4 && (tempb)>0; i++, tempb=tempb-512)
    {// reading and displaying first cluster
        if(tempb<512)
            tempb=512;
        mmcRead(sector+i, buffer);
        myPWMtest();
        // rprintf(" sector: %d ", i);
    }
}
if(tempb==0)
{
    // rprintf(" raj kumar: ");
    break;
}
}
void myPWMtest(void)
{
    tempc = 0;
    timer1Init();
    timer1PWMInit(8);
    // FA --> 7.96KHZ, F8 --> 8.03 KHZ so 0x00F9 -->~ 8KHz
    timer1PWMInitICR(0x00F9);
    timer1PWMAOn();
    timerAttach(TIMER1OVERFLOW_INT, overflow);
    for(; tempc < 511;)
    {
    }
    timer1PWMOff();
    // timer1SetPrescaler(TIMER_CLK_STOP);
}

void overflow(void)
{
    outb(OCR1AL, buffer[tempc]);
    tempc++;
    if(tempc > 511)
    {
        timerDetach(TIMER1OVERFLOW_INT);
    }
}

u08 keypadTest(void)
{
    u08 a, b = 0, n;
    u08 key = 0xF0;
    u08 received = 0;
    // PORT A --> ROW (setting, output)
    // PORT C --> COLUMN (reading, input)
    // PA0-PA3 --> row and PC0-PC3 --> column
    while(received == 0)
    {
        outb(DDRA, 0xFF);
        outb(DDRC, 0x00);
        outb(PORTA, 0x00);
        a = inb(PINC);
        // rprintf("port A cleared");
        if (a != 0x0F)
        {
            // rprintf("value of a = %d ", a);
            // column detection
            for(n = 1; n < 0x10; n = n * 2)
            {
                // rprintf(" ** %d ** ", i);
                outb(PORTA, n);
            }
        }
    }
delay_us(500);
if (inb(PINC)==0x0F)
{
    b=n;
    break;
}

//rprintf("the value of b = %d ",b);
// first column
//Row detection
if ((a==0x07) && (b==0x01))
{
    key=0x01;
    received=1;
}
else if ((a==0x07) && (b==0x02))
{
    key=0x04;
    received=1;
}
else if ((a==0x07) && (b==0x04))
{
    key=0x07;
    received=1;
}
else if ((a==0x07) && (b==0x08))
{
    key=0x0A;
    received=1;
}

//SECOND COLUMN
else if ((a==0x0B) && (b==0x01))
{
    key=0x02;
    received=1;
}
else if ((a==0x0B) && (b==0x02))
{
    key=0x05;
    received=1;
}
else if ((a==0x0B) && (b==0x04))
{
    key=0x08;
    received=1;
}
else if ((a==0x0B) && (b==0x08))
{
    key=0x00;
    received=1;
}
// THIRD column
else if ((a==0x0D) && (b==0x01))
{
    key=0x03;
    received=1;
}
else if ((a==0x0D) && (b==0x02))
{
    key=0x06;
    received=1;
}
else if ((a==0x0D) && (b==0x04))
{
    key=0x09;
    received=1;
}
else if ((a==0x0D) && (b==0x08))
{
    key=0x0B;
    received=1;
}

// FOURTH column
else if ((a==0x0E) && (b==0x01))
{
    key=0x0F;
    received=1;
}
else if ((a==0x0E) && (b==0x02))
{
    key=0x0E;
    received=1;
}
else if ((a==0x0E) && (b==0x04))
{
    key=0x0D;
    received=1;
}
else if ((a==0x0E) && (b==0x08))
{
    key=0x0C;
    received=1;
}
else
{
    key = 0xF0;
    received=0;
}

//rprintf("value of key is = %d ",key);
else
{
    key = 0xF0;
}
return key;
4. Summary:

The goal of content development and updating the Book with new content which was the central theme behind developing an Architecture using ATmega-162 has been achieved fully and comprehensively using MMC card which is compatible with PC and also with a Microcontroller like ATmega-162 using SPI mode of data transfer. Storage Device used in this design is NCP MMC card which are available in various range of price and storage capacity. So it is able to provide an obvious choice to developers which would lead to an optimization in design. By the use of MMC card the most potential problem of a product (i.e. Electronic Book) which was Content Development has been reduced to transferring some audio files from PC to MMC and putting back it to slot attached with Microcontroller.

PWM Signal Generation capability of Atmega-162 brought down the cost by skipping the use of DAC for audio generation. Generation of analog wave from Pulse Width Modulated signal has been achieved by a 5\textsuperscript{th} order Active low Pass filter which has been realized using Low power op-amps and Discrete components. Use of a better filter IC will add a new dimension in performance without making too many changes in Basic Architecture of Design. The contents are in form of audio wav files which can be created using one of the very basic windows utility known as sndrec32.exe. The specifications of wave file are as: 8-Bit PCM (Mono), 8 KHz - Sampling Frequency and 64 Kbps – Bit Rate. The capability of Re Programming of AVR Microcontroller enabled this architecture to become one of the most highly flexible designs which one can think of. This design is flexible in not only the way it operates but also the accessories which can be attached in future. For Example 4x4 Keypad (currently used) can be replaced by 8x8 keypad and also by any kind of touch sensitive Pad. All it will require is some changes in wire connections and a small change in piece of code which is written in one of the most Basic Programming Language C.
Appendix:

AD/DA Converters:

**Analog Signal:**

An analog signal is a continuously variable representation of a physical quantity, property, or condition such as pressure, flow, temperature, etc. The signal may be transmitted as pneumatic, mechanical, or electrical energy. An analog audio signal, for instance, is a representation of the pressure waves which make up audible sound.

An Analog to Digital converter (AD or ADC) is an electronic circuit which accepts an analog input signal (usually a voltage) and produces a corresponding digital number at the output.

**Digital Signal:**

Signal which takes only \( n \) number of values, maybe off or on. Typically represented by “zero” and “one”. Digital signals require less power but (typically) more bandwidth than analog. A signal composed of discrete states that can be represented by finite numbers. This is in contrast to analog signals which can vary continuously.

A Digital to Analog converter (DA or DAC) is an electronic circuit which accepts a digital number at its input and produces a corresponding analog signal (usually a voltage) at the output. They also exist as modules, ICs, or fully integrated inside other parts, e.g. \( \mu \)Cs.

![Diagram of Real World Interfacing with Digital Devices](image)

Figure A1: Real World Interfacing with Digital Devices
Pulse Width Modulation (PWM):

Basic Concepts:

There are many forms of modulation used for communicating information. When a high frequency signal has Amplitude varied in response to a lower frequency signal we have AM (amplitude modulation). When the signal frequency is varied in response to the modulating signal we have FM (frequency modulation). These signals are used for radio modulation because the high frequency carrier signal is needed for efficient radiation of the signal. When communication by pulses was introduced, the amplitude (PAM), Phase (PPM) and pulse width (PWM) become possible modulation options.

A variation of the amplitude modulation method is to use pulse-width modulation (PWM). With PWM, a periodic digital pulse's duty cycle is proportional to the desired amplitude of the signal at that point in time. Many microcontrollers available today already have PWM outputs, and those that don't usually have, some timers that may be pressed into service to create one. A PWM output must usually drive a low-pass filter to convert the signal to analog levels suitable for driving an amplifier and speaker. In extremely cost-sensitive applications it may be acceptable to apply the PWM signal directly to a speaker and let the inertia of the speaker cone or diaphragm perform the filtering.

In most cases, however, better sound quality requires a separate filter circuit, which may even be incorporated into the amplifier stage that drives the speaker. In general, the higher the pulse rate of a PWM signals is in relation to the modulating signal, the easier filtering becomes. A PWM frequency of 10 times the highest modulating (tone) frequency is a generally sufficient.
PWM signal to Analog Conversion:

A sampled sound system may require low-pass filtering to remove the harsh, hissing sound of high-frequency switching transients caused by the transition from one sample to the next. As with a PWM output, a high sample rate makes filtering easier. If someone using a PWM output already, the low-pass filter that averages the variations on duty cycle of the PWM pulse train will also even out the steps in the sample values.

The simplest analog form of generating fixed frequency PWM is by comparison with a linear slope waveform such as a saw tooth. This is implemented using a comparator whose output voltage goes to a logic HIGH when input is greater than the other.

Figure A3: Analog waveform and its equivalent PWM wave from Triangular Wave
Wave File Format:

When we store a file in wave format it is organized according to following structure.

The WAVE format starts with the RIFF header:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4 bytes</td>
<td>'RIFF'</td>
</tr>
<tr>
<td>4</td>
<td>4 bytes</td>
<td>&lt;file length - 8&gt;</td>
</tr>
<tr>
<td>8</td>
<td>4 bytes</td>
<td>'WAVE'</td>
</tr>
</tbody>
</table>

(The '8' in the second entry is the length of the first two entries. I.e., the second entry is the number of bytes that follow in the file.). Next, the fmt chunk describes the sample format:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>4 bytes</td>
<td>'fmt '</td>
</tr>
<tr>
<td>16</td>
<td>4 bytes</td>
<td>0x00000010 // Length of the fmt data (16 bytes)</td>
</tr>
<tr>
<td>20</td>
<td>2 bytes</td>
<td>0x0001 // Format tag: 1 = PCM</td>
</tr>
<tr>
<td>22</td>
<td>2 bytes</td>
<td>&lt;channels&gt; // Channels: 1 = mono, 2 = stereo</td>
</tr>
<tr>
<td>24</td>
<td>4 bytes</td>
<td>&lt;sample rate&gt; // Samples per second: e.g., 44100</td>
</tr>
<tr>
<td>28</td>
<td>4 bytes</td>
<td>&lt;bytes/second&gt; // sample rate * block align</td>
</tr>
<tr>
<td>32</td>
<td>2 bytes</td>
<td>&lt;block align&gt; // channels * bits/sample / 8</td>
</tr>
<tr>
<td>34</td>
<td>2 bytes</td>
<td>&lt;bits/sample&gt; // 8 or 16</td>
</tr>
</tbody>
</table>
Finally, the **data** chunk contains the sample data:

<table>
<thead>
<tr>
<th>36</th>
<th>4 bytes</th>
<th>'data'</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>4 bytes</td>
<td>&lt;length of the data block&gt;</td>
</tr>
<tr>
<td>44</td>
<td>bytes</td>
<td>&lt;sample data&gt;</td>
</tr>
</tbody>
</table>

The sample data must end on an even byte boundary. All numeric data fields are in the Intel format of low-high byte ordering. 8-bit samples are stored as unsigned bytes, ranging from 0 to 255. 16-bit samples are stored as 2's-complement signed integers, ranging from -32768 to 32767.

For multi-channel data, samples are interleaved between channels, like this:

```
sample 0 for channel 0
sample 0 for channel 1
sample 1 for channel 0
sample 1 for channel 1
...
```

For stereo audio, channel 0 is the left channel and channel 1 is the right.

![Wave file Format (Microsoft Standard)](image-url)

Figure A4: Wave file Format (Microsoft Standard)
AVR Programming Tips:

Basic header (Include) files:
At starting the entire required library file should be included to accomplish a particular task.

Basic I/Os:

All AVR processors have I/O ports which each contain up to 8 user-controllable pins. From a hardware perspective, these I/O pins are each an actual physical pin coming out of the processor chip. The voltage on the pins can be sensed or controlled via software, hence their designation as Input/Output pins. While I/O pins are actual wires in the real world, they also exist inside the processor as special memory locations called registers. The software-controlled contents of these registers are what determine the states and operations of I/O pins and I/O ports.

Since AVR processors deal naturally with 8 bits at a time, I/O pins are grouped into sets of 8 to form I/O ports. Three registers are assigned to each I/O port to control the function and state of that port's pins. The registers are 8-bits wide, and each bit (#0-7) determines the operation of the corresponding number pin (pin 0-7).

The three registers are:

a. DDRx - this register determines the direction (input/output) of the pins on port[x]
   - A '0' bit in the DDR makes that port pin act as input
   - A '1' bit in the DDR makes that port pin act as output

b. PORTx - this register contains the output state of the pins on port[x]
   - A '0' bit makes the port pin output a LOW (~0V)
   - A '1' bit makes the port pin output a HIGH (~5V)

c. PINx - this register contains the input state of the pins on port[x]
   - A '0' bit indicates that the port pin is LOW (at ~0V)
   - A '1' bit indicates that the port pin is HIGH (at ~5V)
Programs:

MMC READ

//=======================================================================================================
//  Project      : Electronic Book for rural Education
//  File Name    : mmctest.c
//  Target MCU   : Atmel AVR series ATmega 162
//  Function     : MMC initialization, Read, Reset
//  Revision History:
//  When         Who            Where
//  ------------- -------------------- ------------------------
//  20-SEP-2005   Rajkumar Parihar MSR India
//=======================================================================================================

//----- Include Files ---------------------------------------------------------
--------
#include <avr/io.h>    // include I/O definitions
#include <avr/signal.h> // include "signal" names (interrupt names)
#include <avr/interrupt.h> // include interrupt support
#include "global.h"     // include our global settings
#include "uart2.h"     // include uart function library
#include "rprintf.h"     // include printf function library
#include "timer.h"     // include timer function library (timer, PWM, etc)
#include "spi.h"       // include SPI interface functions
#include "mmc.h"       // include MMC card access functions
#include "debug.h"

void mmcTest(void);

//----- Begin Code -----------------------------------------------------------
--------
int main(void)
{
    // initialize libraries
    // initialize the UART (serial port)
    uartInit();
    uartSetBaudRate(0,38400);  // 0 -- > uart0 & 1 --> uart1 ; Baud rate
    // make all rprintf statements use uart for output
    rprintfInit(uart0SendByte);
    // initialize the timer system
    timerInit();
    // initialize vt100 terminal
    //Argumanet in ms
    timerPause(100);

    // print welcome message
    rprintf("***Welcome to the MMC Test Suite!***");
    timerPause(1000);

    mmcTest();
    return 0;
}
void mmcTest(void)
{
    u32 sector=0;
    int temp =0;
    u08 buffer[0x200];
    int c;

    // initialize MMC interface
    mmcInit();
    // print new prompt
    rprintf("cmd>");

    // testing loop
    while(1)
    {
        // check for keypress
        if((c=uart0GetByte()) != -1)
        {
            switch(c)
            {
            case 'i':
                // initialize card
                rprintf("***Resetting MMC/SD Card***");
                mmcReset();
                break;
            case 'r':
                // read current sector into buffer
                rprintf("***Read Sector %d***
                mmcRead(sector, buffer);
                // print buffer contents
                for(temp=0;temp<512;temp++)
                    rprintf(" 0x%x ", buffer[temp]);
                break;
            // it is not recommended to write sth in MMC
            case 'w':
                /*write current sector with data from buffer
                rprintf("***Write Sector %d***", sector);
                for(temp=0;temp<512;temp++)
                    buffer[temp]=i;
                mmcWrite(sector, buffer);
                break;*/
            case '+': sector++; rprintf("***Sector = %d***", sector); break;
            case '-': sector--; rprintf("***Sector = %d***", sector); break;
            case '*': sector+=512; rprintf("***Sector = %d***", sector); break;
            case '/': sector-=512; rprintf("***Sector = %d***", sector); break;
            case '\r':
                default:
                    break;
            }
        // print new prompt
        rprintf("cmd>");
    }
}
FAT Implementation and Reading the LIST of FILES

//*************************************************************
******
// Project : Electronic Book for rural Education
// File Name : fattest.c
//
// Target MCU : Atmel AVR series ATmega 162
// Function : MMC initialization, Read, Reset; fat16 implementation,
//            printing the files present in Disk with size and
Starting cluster
// Revision History: When Who where
// ------------------ ------------------ ------------------
// 15-OCT-2005 Rajkumar Parihar MSR India
//*************************************************************
******

//----- Include Files -----------------------------------------------------
#include <avr/io.h> //include I/O definitions (port names, pin names)
#include <avr/signal.h> // include "signal" names (interrupt names)
#include <avr/interrupt.h> // include interrupt support
#include "global.h" // include our global settings
#include "uart2.h" // include uart function library
#include "rprintf.h" // include printf function library
#include "timer.h" //include timer function library (timing, PWM, etc)
#include "spi.h" // include SPI interface functions
#include "mmc.h" // include MMC card access functions
#include "fat.h"
#include "debug.h"

//Some Parameter of MMC card
#define HideSec 32
#define BootSec 1
#define Fat1Size 245
#define Fat2Size 245
#define RootDisp 32
#define Fat1Start 5
#define StclusDisp 0x1A
#define MRDE 512

//function Definition
void fatTest(void);

//----- Begin Code --------------------------------------------------------
int main(void)
{
    // initialize the UART library (serial port)
    uartInit();
    uartSetBaudRate(0,38400); // (uart, baudrate)
    // make all rprintf statements use uart for output
    rprintfInit(uart0SendByte);
    // initialize the timer system
    timerInit();
}
// print welcome message
rprintf("***Welcome to the MMC Test Suite!***");
fatTest();
return 0;
}

void fatTest(void)
{
    unsigned char d;
    u32 sector=0;
    int temp =0;
    u08 buffer[0x200];
    int i,c,j;
    u16 tempa;
    u32 tempb;

    // initialize MMC interface
    mmcInit();
    // print new prompt
    mmcReset();
    d = fatInit(buffer);
    rprintf("cmd>");

    // testing loop
    while(1)
    {
        // check for keypress
        if((c=uart0GetByte()) != -1)
        {
            switch(c)
            {
            case 'r':
                // read current sector into buffer
                rprintf("***Read Sector %d*** ", sector);
                mmcRead(sector, buffer);
                // print buffer contents
                for(temp=0;temp<512;temp++)
                    rprintf(" 0x%x ", buffer[temp]);
                break;
            case '+': sector++; rprintf("***Sector = %d***", sector); break;
            case '-': sector--; rprintf("***Sector = %d***", sector); break;
            case '*': sector+=512; rprintf("***Sector = %d***", sector); break;
            case '/': sector-=512; rprintf("***Sector = %d***", sector); break;
            }
        }
    }
case 'l':  
    // read the filename and size from root entry 
    // adress of root directory sector
    sector=HideSec+Fat1Start+Fat1Size+Fat2Size+BootSec;
    
    for(j=0;j<MRDE/16;j++)
    {
        mmcRead(sector+j, buffer);
        // rprintf(" *! \%d *\!", sector+j);
        // rprintf("%d",sector+j);
        for(i=0; i<MRDE/32;i++)
        
            // recognizing difference between a valid root entry and a phoney entry
            // LFN implmentation will create some phoney entry
            // which we need to skip
            
            if((buffer[RootDisp*i]!=0) && ((buffer[RootDisp*i]!='?') &&
            ((buffer[11+i*RootDisp] & 0x08)==0))
            {
                for(temp=0; temp<8; temp++)
                    rprintf("%c",
                        buffer[RootDisp*i+temp]);
                temp=26+RootDisp*i;
                tempa=*((u16 *)&buffer[temp]);
                // tempa = *a;
                rprintf("", Starting
                cluster:"");
                rprintf("",
                    tempu16(tempa));
                temp=temp+2;
                tempb=*((u32 *)&
                buffer[temp]);
                // tempb = *b;
                rprintf("", size:"");
                rprintfu32(tempb);
                // rprintf("@");
            }
    }
    break;

    case 'r':
        default: 
            break;
    }
    // print new prompt
    rprintf("cmd>");
}
}
# Key Press Decoding

//*************************************************************************************
// Project      : Electronic Book for rural Education
// File Name : keypad.c
//
// Target MCU : Atmel AVR series ATmega 162
// Function    : Keypad
// Revision History:
// When          Who            where
// -------------- --------------- -----------------------
// 25-NOV-2005    Rajkumar Parihar    MSR India
//*************************************************************************************

//----- Include Files -------------------------------------------------------------
--------
#include <avr/io.h> //include I/O definitions (port names)
#include <avr/signal.h> // include "signal" names (interrupt names)
#include <avr/interrupt.h> // include interrupt support
#include "global.h" // include our global settings

// keypad testing loop and detection
int keypadTest(void)
{
    u08 a,b=0,n;
    u08 key=0xF0;
    u08 received=0;
    //PORT A --> ROW (setting, output)
    //PORT C --> COLUMN (reading, input)
    // PA0-PA3 --> row and PC0-PC3 --> column
    while(received==0)
    {
        outb(DDRA, 0xFF);
        outb(DDRC, 0x00);
        outb(PORTA, 0x00);
        a=inb(PINC);
        //rprintf("port A cleared");

        if (a!= 0x0F)
        {
            //rprintf("value of a = %d ",a);
            //column detection
            for(n=1; n< 0x10 ;n=n*2)
            {
                //rprintf(" ** %d ** ", i);
                outb(PORTA,n);
                delay_us(500);
                if (inb(PINC)==0x0F)
                {
                    b=n;
                    break;
                }
            }
        }
    }
}
// rprintf("the value of b = %d ",b);
// first column
// Row detection

if ((a==0x07) && (b==0x01))
{    
    key=0x01;
    received=1;
}
else if ((a==0x07) && (b==0x02))
{    
    key=0x04;
    received=1;
}
else if ((a==0x07) && (b==0x04))
{    
    key=0x07;
    received=1;
}
else if ((a==0x07) && (b==0x08))
{    
    key=0x0A;
    received=1;
}

// SECOND COLUMN
else if ((a==0x0B) && (b==0x01))
{    
    key=0x02;
    received=1;
}
else if ((a==0x0B) && (b==0x02))
{    
    key=0x05;
    received=1;
}
else if ((a==0x0B) && (b==0x04))
{    
    key=0x08;
    received=1;
}
else if ((a==0x0B) && (b==0x08))
{    
    key=0x00;
    received=1;
}

// THIRD COLUMN
else if ((a==0x0D) && (b==0x01))
{    
    key=0x03;
    received=1;
}
else if ((a==0x0D) && (b==0x02))
{    
    key=0x06;
received=1;
}
else if ((a==0x0D) && (b==0x04))
{
    key=0x09;
    received=1;
}
else if ((a==0x0D) && (b==0x08))
{
    key=0x0B;
    received=1;
}

// FOURTH column
else if ((a==0x0E) && (b==0x01))
{
    key=0x0F;
    received=1;
}
else if ((a==0x0E) && (b==0x02))
{
    key=0x0E;
    received=1;
}
else if ((a==0x0E) && (b==0x04))
{
    key=0x0D;
    received=1;
}
else if ((a==0x0E) && (b==0x08))
{
    key=0x0C;
    received=1;
}
else
{
    key = 0xF0;
    received=0;
}
rprintf("value of key is = %d ",key);
}
else
key= 0xF0;
}
return key;
References:

DATA SHEET:
1. Atmega-162 Data Sheet
2. MMC MiniSD Data Sheet
3. LM324 Data Sheet
4. LM386 Data Sheet
5. MAX-232 Data Sheet

WEBSITES:
1. http://www.google.com

Research Papers:
2. “Curlybot: Designing a new Class of Computational Toys”, by Phil Frei and teams.
3. “Smart Toys: Brave new World?”, by H D’Hooge and others