DESIGN OF AN AUDIO-VISUAL DISPLAY FOR CROSS-MODAL EXPERIMENTATION

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Sciences

By

Enrique Alvarez Vazquez

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department: Electrical and Computer Engineering

May 2012

Fargo, North Dakota

North Dakota State University

Graduate School

Title

DESIGN OF AN AUDIO-VISUAL DISPLAY FOR CROSS-MODAL EXPERIMENTATION

By

ENRIQUE ALVAREZ VAZQUEZ

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

SUPERVISORY COMMITTEE:

DR. JACOB GLOWER

Chair

DR. CHAO YOU

DR. BENJAMIN BRAATEN

DR. MARK McCOURT

Approved:

5/31/2012

Date

DR. RAJENDRA KATTI

Department Chair

ABSTRACT

The present paper shows the characteristics of a hardware/software display capable of presenting, at different spatial locations, multiple audio-visual stimuli in real-time. The system has been developed to provide a tool in neuroscience in event-related potential (ERP) studies, with presentation delays of less than one millisecond. The high time precision achieved makes it possible to overcome problems from previously used technologies in order to create a realistic environment of cross-modal stimulation.

The design presented here has been built using a modular approach, which makes the system flexible and adaptable to different experiments or studies.

The capability of presenting different stimuli either in audition, vision, or both combined in real-time, facilitates the study of realistic cross-modal attention experiments.

ACKNOWLEDGMENTS

First of all, I want to thank my family for all the support I have had during my life. To everyone one of them, thank you very much.

I also want to thank Dr. Mark McCourt and Dr. Wolfgang Teder for giving me the opportunity to work in the Center of Visual and Cognitive Neuroscience. Thanks for the support you have given me and for enabling me to apply my engineering skills in this project as well as others.

Also, thanks to Dr. Jake Glower for being my graduate advisor and for helping me during this process, I really appreciate it. Also great thanks to all the NDSU Electrical and Computer Engineering faculty and staff.

Thanks to Alyson Saville for helping me review my English and tons of thanks to all the COBRE and Psychology faculty and staff.

DEDICATION

I dedicate this Master's Paper to my parents.

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGMENTS iv

DEDICATION v

LIST OF TABLES ix

LIST OF FIGURES x

LIST OF ABBREVIATIONS xii

1. INTRODUCTION 1

    1.1 Vision and Audition 1

    1.2 Project Requirements 2

    1.3 Organization of this Paper 2

2. PREVIOUS WORK 4

3. HARDWARE DESIGN 9

    3.1 Hardware Requirements 10

    3.2 Communications Bus 10

    3.3 Microcontroller 11

    3.4 LED Circuitry 12

        3.4.1 LED Selection 12

        3.4.2 LED Circuit Arrangement 13

        3.4.3 LED Grey Levels 14

    3.5 Audio Circuitry 15

3.5.1 Speakers 16

3.5.2 The Audio Driver 16

3.6 External Memory 16

3.7 PCB Stack 17

3.7.1 Layer 1: LEDs 17

3.7.2 Layer 2: Transfer Connections 19

3.7.3 Layer 3: Microcontroller 19

3.7.4 Layer 4: External Memory 21

3.8 Summary 22

4. SOFTWARE DESIGN 23

4.1 Introduction 23

4.1.1. Software Requirements 23

4.2 Communications Network 24

4.2.1 Format for Serial Commands 24

4.2.2. Processing Times 26

4.3 Command Processing 26

4.3.1 LoadImage() 27

4.3.2 ShowImage2() 30

4.3.3 ShowImageWithAudio() 32

4.3.4 ShowImage5() 33

4.3.5 SetPixelInImage() 34

4.3.6 SetJustOnePixel() 35

4.3.7 SetAudio() 36

4.3.8 ClearDisplay() 37

4.4 Displaying Simultaneous Stimuli 38

4.4.1 Latency for Simultaneous Events 38

4.5 Summary 41

5. VERIFICATION 42

5.1 Validation (take 2) 44

6. FUTURE WORK AND CONCLUSIONS 45

7. REFERENCES 47

APPENDIX A. CS FIRMWARE 50

APPENDIX B. INDIVIDUAL CELL FIRMWARE 65

Header – Rainbow.h 65

Data.c 66

Main Cell Program 74

LIST OF TABLES

Table Page

1. Software Commands 25

2. Command Processing Times 26

3. Bit Assignment for Audio Channels 36

4. Time required for Every Control Instruction 43

5. Comparison of Audio Visual Device's Specifications to Design Requirements 44

LIST OF FIGURES

Figure | Page

1. Auditory / Visual Displays as Used by Spence and Driver [8]. Speakers Are Located Above a Light Display. The Lights and Speakers Can Be Turned On and Off Independently. 4

2. Cross-modal Spatial Attention Experiment with Built-in LEDs on Speakers [1]. 5

3. Sources Attached to a Panel in Different Locations by a Fastener System [9]. 6

4. Speaker Array used in the CVCN [13]. 6

5. Cross-Modal Display Developed in 2006 From “LED / Audio Array” by Jake Glower, 2006, Dept. of Electrical and Computer Engineering – NDSU [14]. 7

6. Hardware System Overview. 9

7. ATMEGA368 Microcontroller Pin Configuration: [16].. 12

8. Time Modulation to Energize Each Column Sequentially. 13

9. Five I/O Pins from the Microcontroller Connect to the Rows, Allowing You to Turn On Any LED Independently. 14

10. Powered Delivered to the LEDs in Row #1 Through Pulse Width Modulation.............. 14

11. Audio Amplifier Circuit.. 15

12. External Memory Circuit. 17

13. Layered Design of the Individual Cells. 17

14. Schematic and Board Layout of Layer 1. 18

15. Schematic and Board Layout of Layer 2. 19

16. Schematic and Board Layout of Layer 3. 20

17. Schematic and Board Layout of Layer 4. 22

18. Format of Serial Commands. 25

19. Delay from an Instruction as Measured Using a Logic Analyzer.................................. 26

20. Command Processing Flow Chart. 27

21. Stored Symbol in Memory. 28

22. Microcontroller and Memory Connections. 29

23. Flow Chart for ShowImage2(). 30

24. Developed Custom Symbols. 32

25. Flow Chart for ShowImageWithAudio(). 33

26. Flow Chart of ShowImage5(). 33

27. Flow Chart for SetPixelInImage(). 34

28. Flow Chart for SetJustOnePixel(). 35

29. Flow Chart for SetAudio(). 36

30. Audio Hardware Schematic. 37

31. Flow Chart for ClearDisplay(). 37

32. Delay from an Instruction as Measured Using a Logic Analyzer. 38

33. Set of Events being Displayed Simultaneously as Measured on a Logic Analyzer. 41

LIST OF ABBREVIATIONS

ANSI	American National Standards Institute
CRT	Cathode Ray Tube
CS	Central system
CSI	Circuit Specialists Inc.
CVCN	Center of Visual and Cognitive Neuroscience
DC	Direct Current
EEG	Electroencephalogram
EEPROM	Electrically Erasable Programmable Read-Only Memory
ERP	Event related potential
GUI	Graphic User Interface
LED	Light emitting diode
MAA	Minimum auditory angle task
MVA	Minimum visual task
NCRR	National Center for Research Resources
NIH	National Institute of Health
PICC	Peripheral Interface Controller Compiler
PWM	Pulse Width Modulation
RAM	Random-access memory
RGB	Red Blue Green
TFT	Thin film transistor

1. INTRODUCTION

1.1 Vision and Audition

Anyone who has ever taken a walk in the woods can attest to the sensitivity of our senses to motion. When we see a squirrel runs across a tree branch, our attention is attracted to the motion. When hear rustling leaves moving around, our attention is likewise attracted. This sensitivity to motion, both audio as well as visual, can be monitored through brain activity.

Some tests indicate that our senses are not independent. For example, when listening to directions on the phone, some people find it easier to concentrate if they close their eyes. Students are often told to turn off the TV or stereo while studying - although this may not happen that often. This suggests that there are 'links' between our senses.

It has been extensively demonstrated that our sensory system utilizes “links” between modalities such as vision and audition [1] [2]. Some of these links are indeed helpful in the context of animal experiments: temporal coincidence of auditory and visual stimuli speeds up reaction time and accuracy [3]. In humans, these findings have been confirmed and extended. It has been demonstrated that both temporal and spatial coincidence plays a crucial role in cross-modal integration [4][5]. Given the speed of information processing, many audio-visual experiments have been flawed by seemingly minute and irrelevant temporal disparities caused by screen refresh rates and spatial inaccuracies such as computer speakers attached to the sides of a monitor [6].

In order to better quantify and understand the links between audio and visual stimuli, a device is needed which allows the operator to precisely control visual and audio signals sent to a test subject. The design, building, and verification of such a device is the focus of this study.

1.2 Project Requirements

In this work, an audio-visual display will be developed. This display will be a flexible tool that will display visual and auditory stimuli in specific physical locations. The design is very precise temporally in order to allow the use of this tool for ERP experimentation. The solution must be able to present visual stimuli and auditory stimuli at the same spatial location.

Specifically, the audio visual display must meet the following requirements:

- 55 cells arranged in a 5 by 11 grid.
- Each cell contains an array of LEDs and a speaker.
- Each array of LEDs are independently controlled with red, green, and blue LEDs, each with 16 levels of grey and a response time of less than 1ms
- Each speaker can be independently turned on and off and able to mix at least 2 audio channels
- There must be no pop noise when switching a channel or speaker on and off.
- The overall system should be programmable for a given experiment taking in consideration that a regular research session in attention behavioral experiments usually takes from 3 to 5 minutes in duration and contains a minimum of 120 trials/events per session.
- Programming time must take less than one minute, and
- The total power required should be less than 3A.

The final design is intended for use as a tool in basic neuroscience research by the Center of Visual and Cognitive Neuroscience, a center sponsored by the National Institute of Health.

1.3 Organization of this Paper

The paper is organized as follows:

Chapter 1 contains an introductory description of the system being developed in this paper and the rationale for this design.

Chapter 2 describes current state-of-the art systems used in the field along with their short comings.

Chapter 3 contains the description of the system hardware requirements, hardware schematics, and board designs.

Chapter 4 describes the software design, outlining software requirements and describing how the software is developed to comply with those requirements.

Chapter 5 verifies the correct operation of the hardware and software components.

Chapter 6 presents conclusions and suggests future work on the design that might benefit the function of the system.

This publication was made possible by grant number 2P20RR020151-06A1 from the National Center for Research Resources (NCRR), a component of the National Institutes of Health (NIH). Its contents are solely the responsibility of the authors and do not necessarily represent the official view of the NCRR or NIH.

2. PREVIOUS WORK

There has been considerable research in visuospatial attention. However, research in audiospatial attention is not as well understood [7]. Moreover, the interaction between sensory data has only recently been considered and studied.

Cross-modal research is a way of learning how the brain assembles information from different sensory systems. Interactions among senses were introduced by Stein and Meridith [3]. This led to investigation into how neural processes determine what interactions take place among the different senses.

Cross-modal research became a very active topic in the early 1990s, with investigators including Martin Eimer, Jon Driver, and Charles Spence [8][9][10].

In order to investigate cross-modal interaction, several auditory and visual display systems were created. Early prototypes, such as the one presented in Figure 1, were composed of a computer monitor with a limited number of speakers placed to the sides of the monitor [8]. Evolution of these prototypes included simpler visual stimuli, such as arrays of light bulbs or light-emitting diodes (LEDs) on top of loud speakers [1].

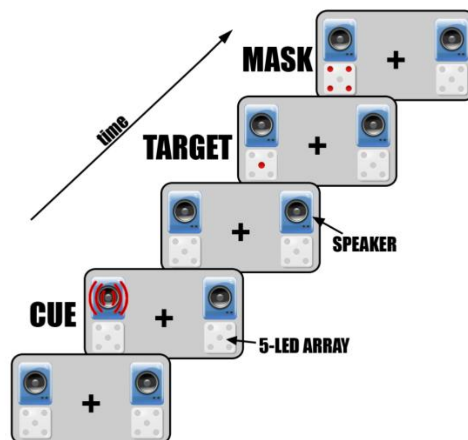

Figure 1. Auditory / Visual Displays as Used by Spence and Driver [8]. Speakers Are Located Above a Light Display. The Lights and Speakers Can Be Turned On and Off Independently.

In this work, it was determined that the localization performance in minimum auditory angle task (MAA) was equal to or better than the results obtained in the minimum visual task (MVA) for all regions in the frontal field with only one exception: presentations at 0 degrees azimuth [11]. The spatial accuracy obtained was about 1-2 degrees of angle [12].

These results suggest that the spatial location of the presented stimuli is critical. Likewise, the effect of having several stimuli occurred in the exact spatial location became of interest, and hardware to test such stimuli were developed.

More recent cross-modal displays included a discrete number of lights, usually no more than six, which are built into loud speakers. A typical configuration of this type is presented in Figure 2 [1]. This setup assures a closer physical location of both stimuli.

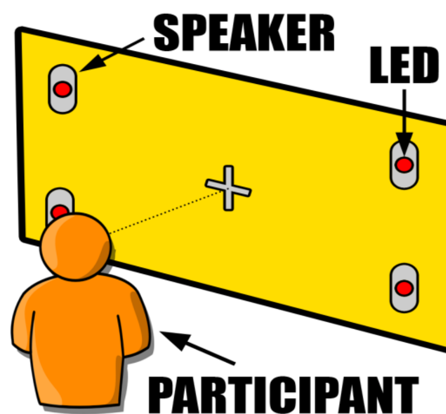

Figure 2. Cross-modal Spatial Attention Experiment with Built-in LEDs on Speakers [1].

A variation on this design allows each speaker / auditory unit to be placed anywhere on a display, using an anchor systems such as Velcro. This variation adds flexibility to the experiments, permitting an infinite number of arrangements. For example, a CVCN model with these characteristics, as developed by Driver and Spence [9] is presented in Figure 3. While the location of each cell is infinitely variable, the total number of cells in this design is very limited:

typically 4 to 8. In addition, changing the location of a cell requires the manual manipulation of the placement of the source device.

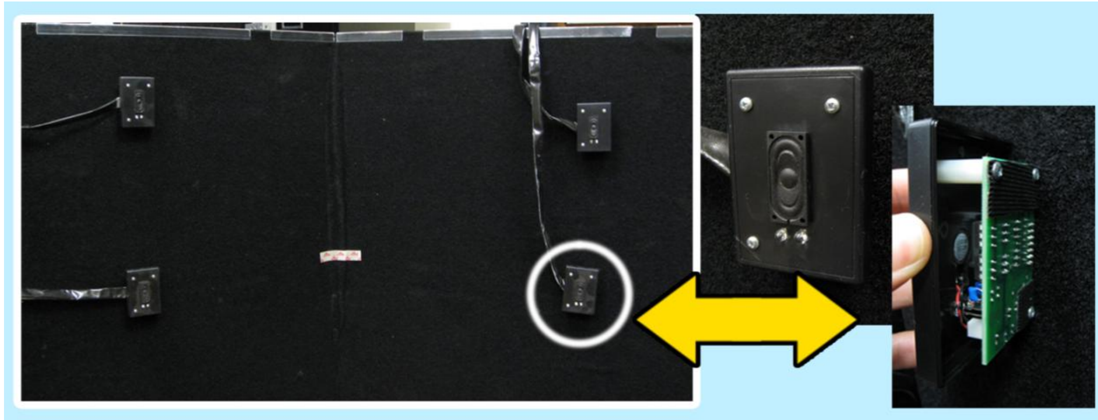

Figure 3. Sources Attached to a Panel in Different Locations by a Fastener System [9].

More recent designs have focused on increasing the number of cells, with the CVCN system was developed at NDSU by Sosa, Teder, & McCourt in 2010 being one example. This system consists of 27 cells, each consisting of one monochrome LED and a single speaker as presented in Figure 4. It was used to deliver auditory stimuli in auditory bisection experimentation [13].

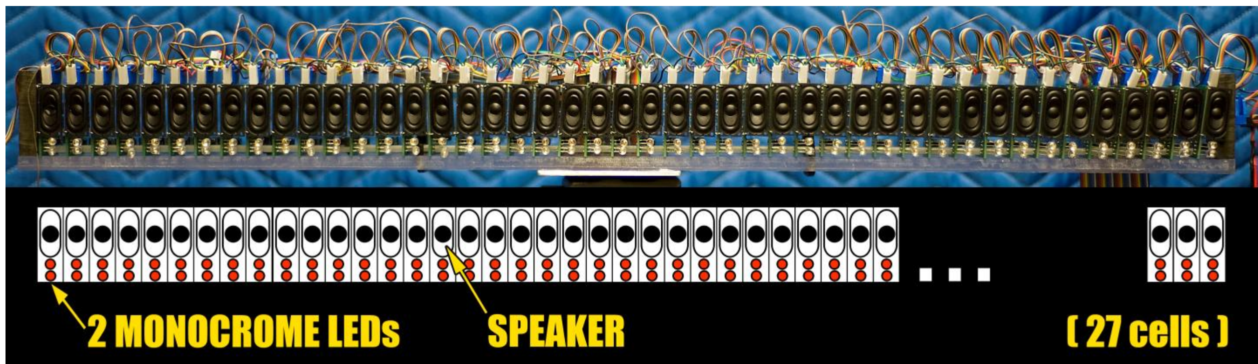

Figure 4. Speaker Array used in the CVCN [13].

In 2006, a 2-dimensional version was developed by the NDSU Center of Visual Neuroscience in collaboration with the Department of Electrical and Computer Engineering, presented in Figure 5. This version of a cross-modal display consisted of 55 cells, arranged in an

11 x 5 grid. For the audio portion, each cell is able to mix four audio signals with 256 volume levels for each channel, which drive an 8-Ohm speaker at up to 3 Watts. For the visual portion, a set of RGB LEDs are driven with 256 levels of grey for each color. Each cell was controlled through a common 0V / 5V line, which synchronized the output for all 55 elements to within one millisecond. On-board EEPROM stored a program, which could be up to 800 lines long. Each line specified the volume for each channel for that cell as well as the RGB brightness. [14]

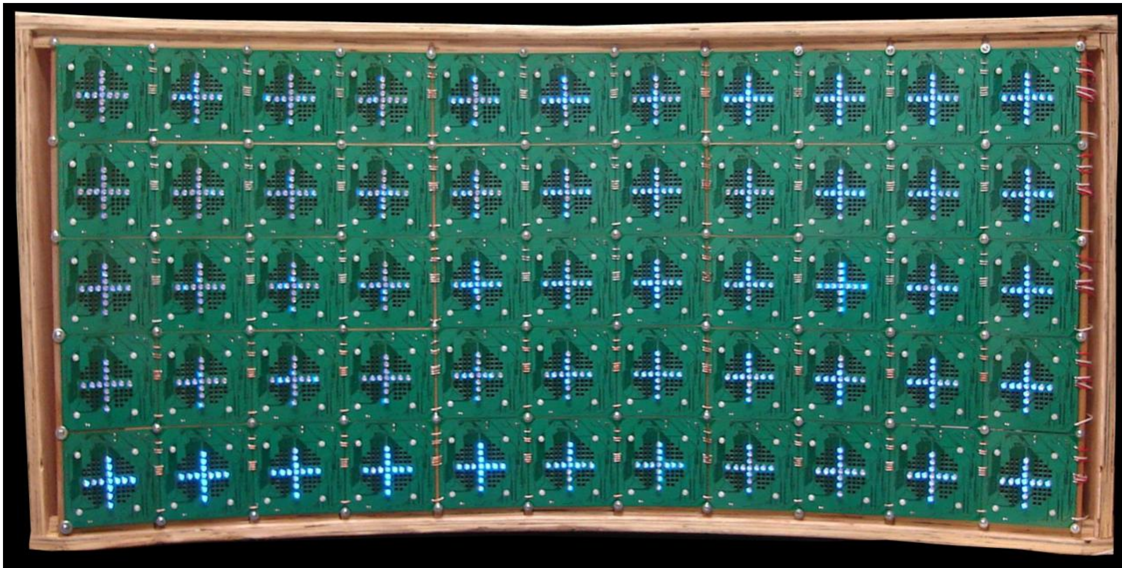

Figure 5. Cross-Modal Display Developed in 2006 From “LED / Audio Array” by Jake Glower, 2006, Dept. of Electrical and Computer Engineering – NDSU [14].

The execution of the program lines was synchronized with a pulse input, which allowed different cells to respond within 1ms of each other. This prototype represented a significant improvement from the devices used in previous research; however it had limitations that had to be solved in order to be usable in basic research:

- The memory in the cells limited the experiment to 800 sequential events,
- It required approximately 15 minutes to download a new experiment to the display, and

- Cells have independent control of the centered led and the cross edge led only, which didn't allow the creation of custom symbols, only crosses and points.

The design presented in this paper overcomes the limitations of the previously mentioned system as well as it adds new attractive features.

3. HARDWARE DESIGN

The Audio Visual Display developed in this paper is a device which allows both visual and audio signals to be presented to a subject in an 11 x 5 array. To simplify the design of this display, each cell will have an identical design - including both software and hardware. Only the identifier for each cell will differ from cell to cell.

The hardware organization of each of the 55 cells is presented in Figure 6. The heart of each cell is a microcontroller. This monitors a communications bus for commands as well as drives the LEDs and mixer for the audio channels. Attached to the microcontroller is external RAM, which stores data for the experiment being executed. Visual output is provided by LEDs arranged in a 5x5 grid, controlled by the microcontroller. Finally, audio signals are mixed and amplified, driving a 2 Watt speaker, with the mixing and volume controlled by the microcontroller.

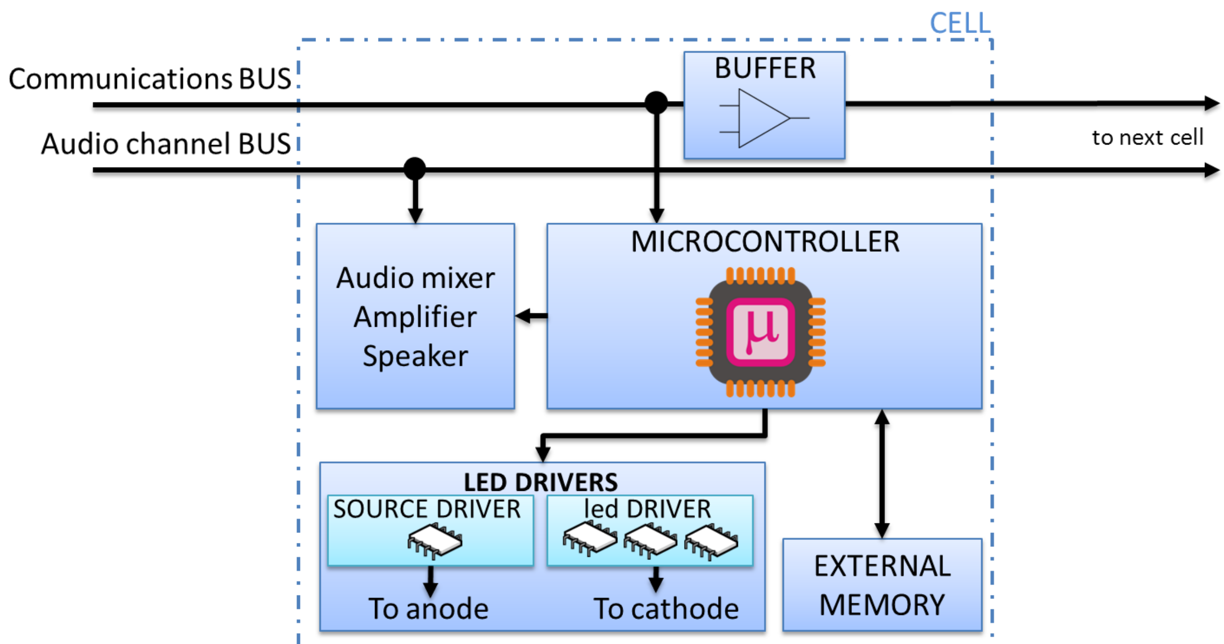

Figure 6. Hardware System Overview.

In this chapter, a brief description of the hardware requirements and the hardware for each of these sections is presented.

3.1 Hardware Requirements

To meet the overall system requirements, the hardware must be capable of the following:

- Display different colors and patterns, with 16 levels of grey
- Change the display settings in less than 1ms
- Drive the LEDs with an on-off cycle exceeding 16Hz
- Mix at least two audio channels
- Have no pop-noises when turning a channel on and off
- Drive an 8 Ohm speaker at up to 2 Watts
- Be tied to a network, which allows programming of a sequence in less than 1 minute as well as controlling the system (discussed in Chapter 4: Software Design), and
- Be simple to debug and repair.

A brief description of the circuitry which meets these requirements is presented in the following sections.

3.2 Communications Bus

In order to be compatible with previous designs used in the Center of Visual and Cognitive Neuroscience Lab at NDSU, the communications bus is required to operate as follows:

- 0V / 5V TTL data levels
- 300 kbps serial data
- 8 data bits, 1 start bit, 1 stop bit, no parity bit.

- Communications only take place from the CS to the cell. Hence, hardware to allow handshaking is not necessary.

3.3 Microcontroller

The microcontroller for the AVD needs to have the following:

- 20 binary outputs to allow each of the 75 LEDs in a cell to be turned on and off independently,
- An SPI output with three chip-select lines to drive the audio mixer,
- An ability to read and write 512 Mb of external memory, used to store data for the experiment being executed, and
- An asynchronous serial bus capable of 300 kbps communications.

The microcontroller selected was an ATMEGA368. This microcontroller has the following specifications:

- 16 Kbytes of in-system self-programmable flash program memory
- 512 bytes EEPROM
- 1k bytes internal SRAM
- Six PWM channels
- Programmable serial USART
- Master/slave SPI serial interface
- Byte-oriented 2-wire serial interface (Philips I2C compatible)
- 23 programmable I/O lines

The Atmega 8/168/328 has enough resources to generate 4096 color for 64 dots while providing complete I2C and UART communication. More importantly they are a popular MCU

among open source hardware community, making it compatible to Arduino IDE and the vast knowledge pool. [15]

The schematic for the microcontroller attached to the system is shown in Figure 7.

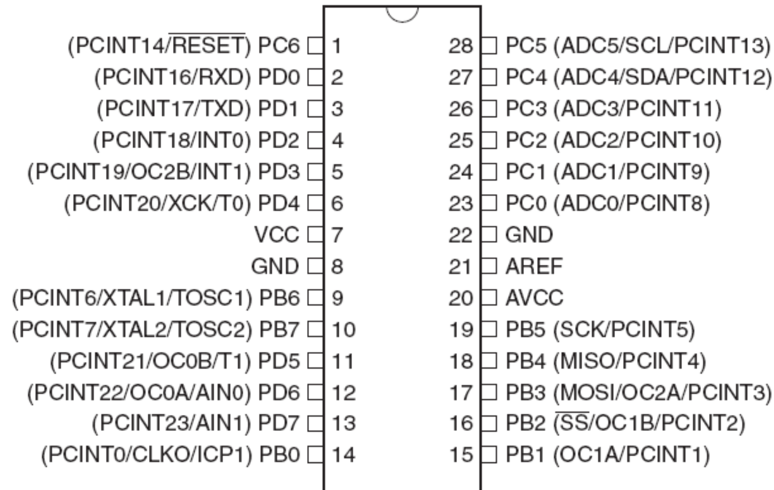

Figure 7. ATMEGA368 Microcontroller Pin Configuration: [16].

3.4 LED Circuitry

Each cell in the AVD is to have 25 RGB LEDs arranged in a 5x5 grid. Each of these 75 LEDs must be able to be turned on and off independently, with 16 levels of grey. In order to limit the number of I/O lines and to simplify the software required to drive the LEDs, the following circuits were used.

3.4.1 LED Selection

First, the LEDs need to be selected. Full-color LEDs offering high intensity light output with a wide viewing angle (120°) were selected in order to present crisp visual symbols at different brightness levels. The modular approach utilized in this design not only allows auditory and visual stimulus presentation at different spatial locations but also allows different patterns to be displayed in each cell's 5x5 grid. Additionally, this approach allows the

experimenter or engineer to simply correct or repair any potential malfunction that could occur in a particular cell.

3.4.2 LED Circuit Arrangement

In order to limit the number of I/O lines required to drive the 75 LEDs (25 red, green, and blue LEDs), the following circuit was used. Five outputs from the microcontroller determine which column is being energized. It is intended that only one of these five is high at any given time - with the column being energized sequenced as presented in Figure 8.

The refresh rate for the LEDs is 500μs per cycle (2000 Hz), which is considerably less than other visual displays such as thin film transistor liquid crystal display (TFTs) and cathode ray tube screens (CRTs) [6].

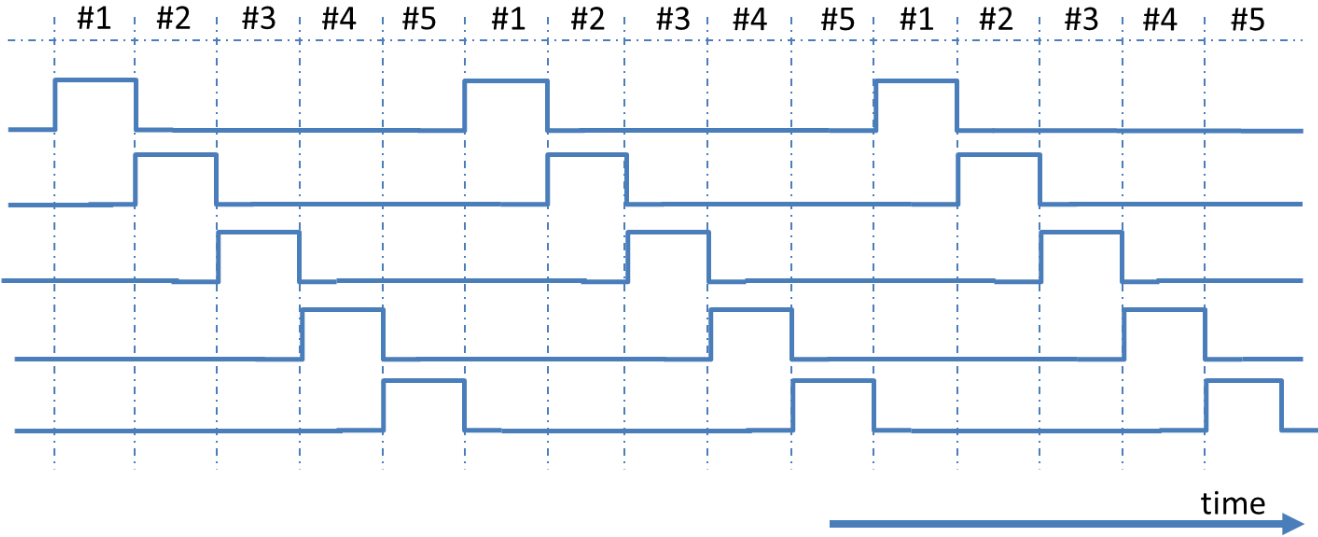

Figure 8. Time Modulation to Energize Each Column Sequentially.

In order to turn on each LED separately, five different output pins from the microcontroller are connected to each row as presented in Figure 9.

A similar set up is used for each color: red, green, and blue, resulting in 20 I/O pins from the microcontroller dedicated to controlling all 75 LEDs.

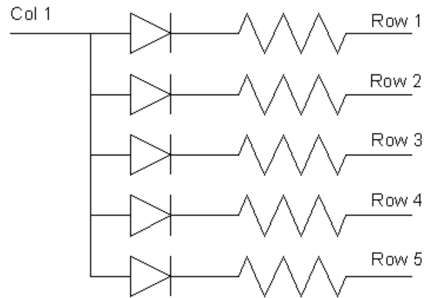

Figure 9. Five I/O Pins from the Microcontroller Connect to the Rows, Allowing You to Turn On Any LED Independently.

3.4.3 LED Grey Levels

In order to adjust the brightness of the cell in its entirety, the signals presented in Figure 10 are modified using pulse-width modulation with 16 levels of grey. For 100% brightness, each LED will be on 20% of the time (the remaining 80% of the time other columns are being energized.) This on-time can be reduced to 0% in 16 steps using the TIMER2 and the TCNT2 value as shown in Figure 10.

Every LED in the matrix pulses at 2KHz. Therefore, considering that the human eye behaves as a low pass filter [17] for a signal over 16 Hz [18][19] the amount of energy contained by this wave will be directly correlated to the duty cycle, and also, to the amount of intensity perceived by human vision.

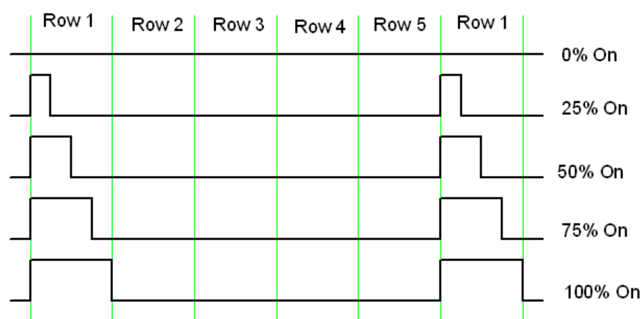

Figure 10. Powered Delivered to the LEDs in Row #1 Through Pulse Width Modulation.

3.5 Audio Circuitry

In addition to driving LEDs, each cell must mix, amplify, and send an audio signal to a 2 Watt speaker. To add flexibility, three channels are mixed, with the circuit used presented in Figure 11.

The microcontroller has 3 digital outputs that control the pass of the audio channels #1, #2 and #3 to the audio amplifier and consequently, the speaker. Every sub-cell that controls the connection between the audio bus and the summing audio amplifier is based on 2 optical isolators. While one of the isolators is open the other isolator is closed and vice versa, this allows eliminating any spurious noise contained in the channel that would cause an undesired noise in the output audio from the speaker. [21]

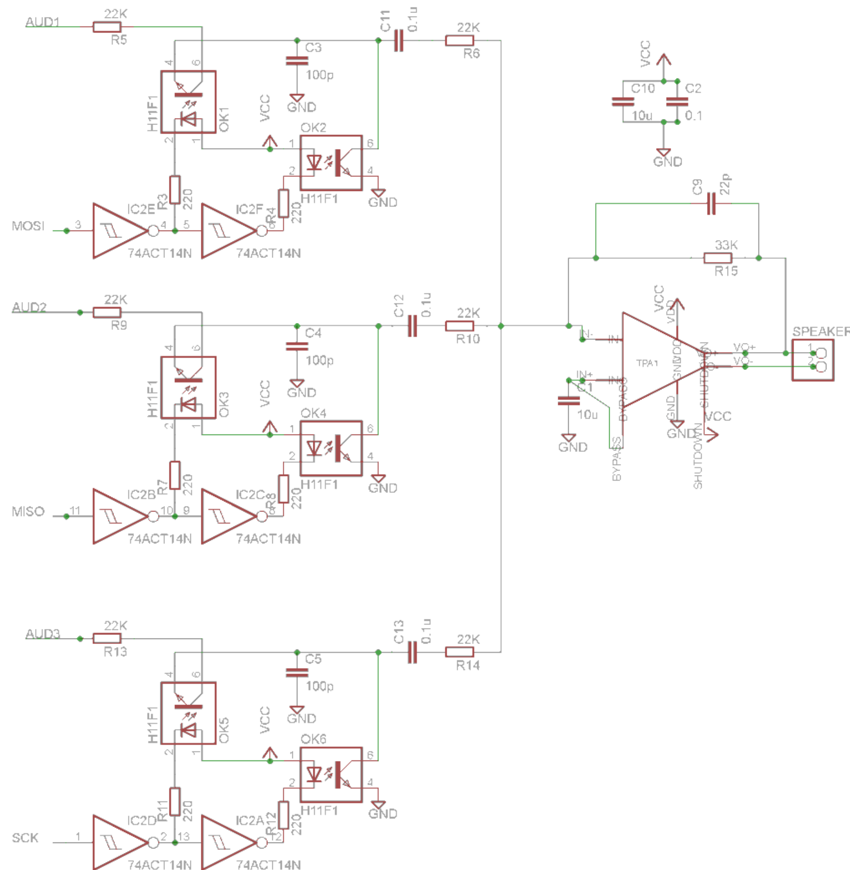

Figure 11. Audio Amplifier Circuit.

3.5.1 Speakers

The current system approach allows three audio channels to be played in every cell, exceeding by one the hardware requirements. Three double-photo transistor based anti-pop switches provide a fast response as well as a satisfactory elimination of any clicking noise when opening or closing audio channels.

3.5.2 The Audio Driver

The audio driver has three main parts: the pink noise built-in generators, the audio amplifier, and the audio switches.

The Audio Visual Device contains three built-in pink noise generators, each with a different audio spectrum. The generators are connected to the cells using 3 audio channels that are part of a communications bus with all three channels available to each cell. Every cell manages the audio channels using 3 anti-pop audio switches. The anti-pop audio connects or disconnects the audio channels to the audio amplifier. The audio amplifier then amplifies the signal and drives the 2 Watt speaker. [Figure 11]

The anti-pop audio switches are based on a double-photo transistor structure, the double structure eliminates the transitory 'click' noise produced in any conventional audio switch, the latency of the on/off ramp is as low as 200ns [21].

3.6 External Memory

The external memory is a 512Mb EEPROM chip connected to Vcc, GND and to an input/output inter-integrated circuit (I2C) port, the pin indicated as Serial Data Line (SDA) on Figure 12 allows the bidirectional communication between the microcontroller and the EEPROM. Serial clock (SCL) is pulled up with a 10K resistor.

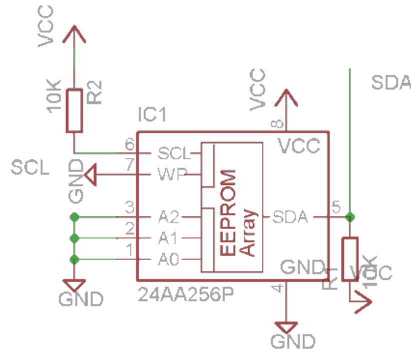

Figure 12. External Memory Circuit.

3.7 PCB Stack

Each cell of the display contains a set of LEDs, a speaker, hardware to drive these, and a microcontroller to coordinate the activities of the Audio Visual Device. Since spatial location of the cells is of paramount importance, it is desired to minimize space between cells. Because of this lack of space, a stack of boards is used for each cell as presented in Figure 13.

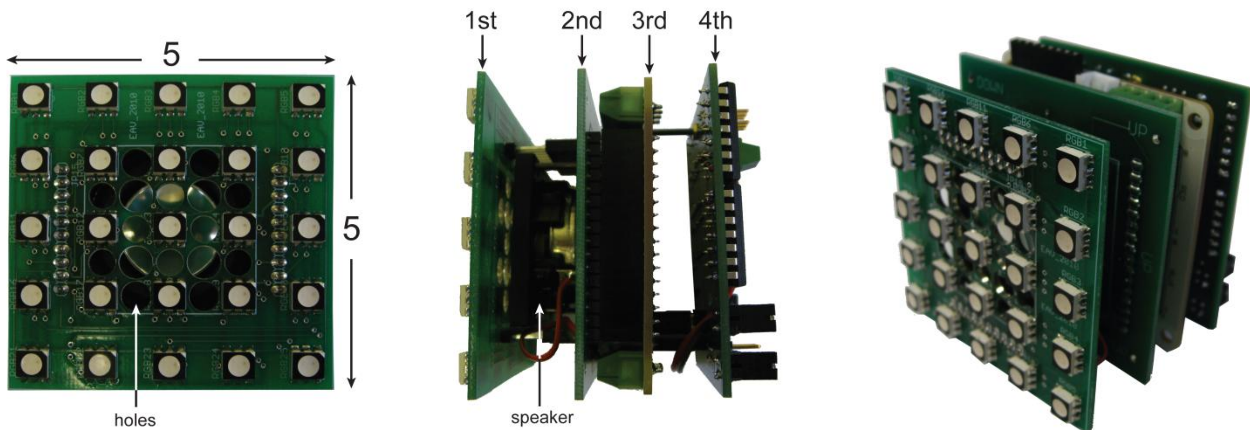

Figure 13. Layered Design of the Individual Cells.

The function of each of these four layers is as follows:

3.7.1 Layer 1: LEDs

The first layer consists of 25 RGB LEDs (5x5), and it also contains several holes in-between the LEDs [Figure 7 and Figure 8]. These holes allow the sound from the speaker to be

able to go through the board and transmit properly at exactly the same spatial location as the visual stimuli.

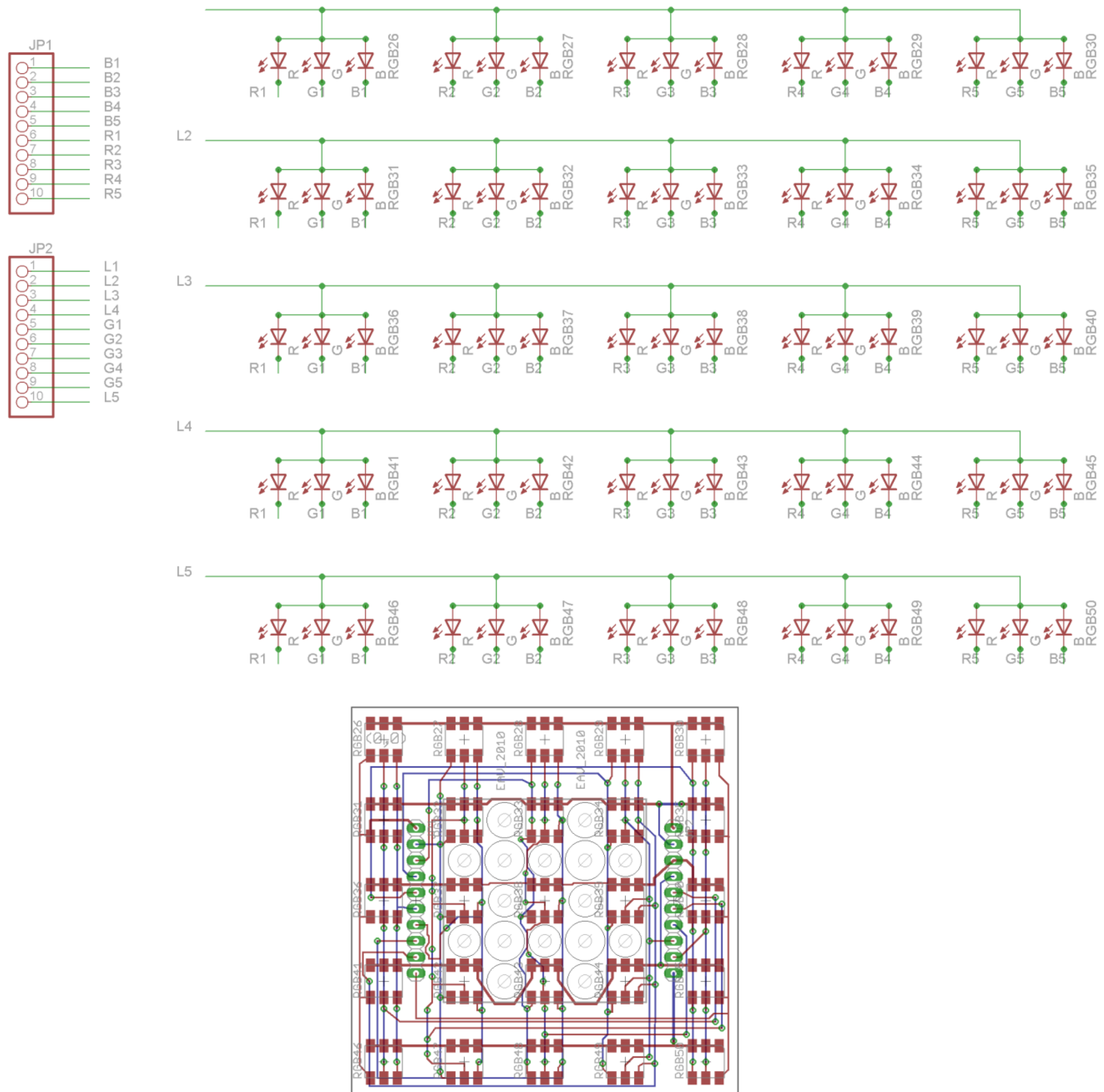

Figure 14. Schematic and Board Layout of Layer 1.

Each of the 25 RGB LEDs in the matrix of 5x5 can be driven individually. Two parameters can be set, intensity and color (16 levels of intensity and 4096 colors per dot), which

are controlled using three pulse-width modulated (PWM) driver integrated circuits (ICs), one for each color, in the present case: red, green and blue.

The matrix is controlled powering sequentially one row at a time. Every time a row is enabled the microcontroller synchronizes the PWM circuit to output the necessary energy to every column. The PWM circuitry has three units, one unit per color, and every unit has 5 outputs, one for each column [Figure 14].

3.7.2 Layer 2: Transfer Connections

The second layer holds the speaker and transfers the connections from the LEDs to the 3rd layer. [Figure 15]

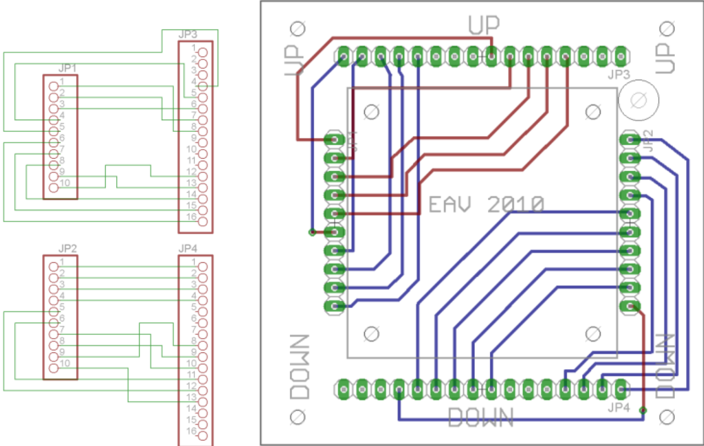

Figure 15. Schematic and Board Layout of Layer 2.

3.7.3 Layer 3: Microcontroller

The third layer contains the microcontroller (center of operations) of the cell, the LED driver circuitry and the power management [Figure 16].

This third layer is based on the platform Rainbowduino ((Albert Miao, 2009)). It contains an Atmega 168/328, one super source driver (M54564P) based on 8 circuit output-sourcing Darlington transistor arrays, and three constant current LED drivers (MBI5168) that are designed for LED display applications, the drivers include with PrecisionDrive™ technology to

enhance its output characteristics. Moreover, MBI5168 contains data latches a serial buffer, that allows converting serial input data into parallel output format. The system provides uniform and constant current sinks for driving LEDs within a large range of Vf variations. [15]

The layout is implemented on a two sided board, providing compact size in addition to accessing different components from both sides. This last characteristic is essential since the microcontroller layer is #3, between layer #2 and #4.

Power and ground planes are present in the design in order to avoid noise interferences in the LED drivers as well as making the routing simpler. The design was implemented using Easily Applicable Graphical Layour Editror (EAGLE) version 6.1.0 for Windows. The boards were designed and purchased from Seedstudio (Rainbowduino), then they were modify to accommodate the layer pattern used in the overall design.

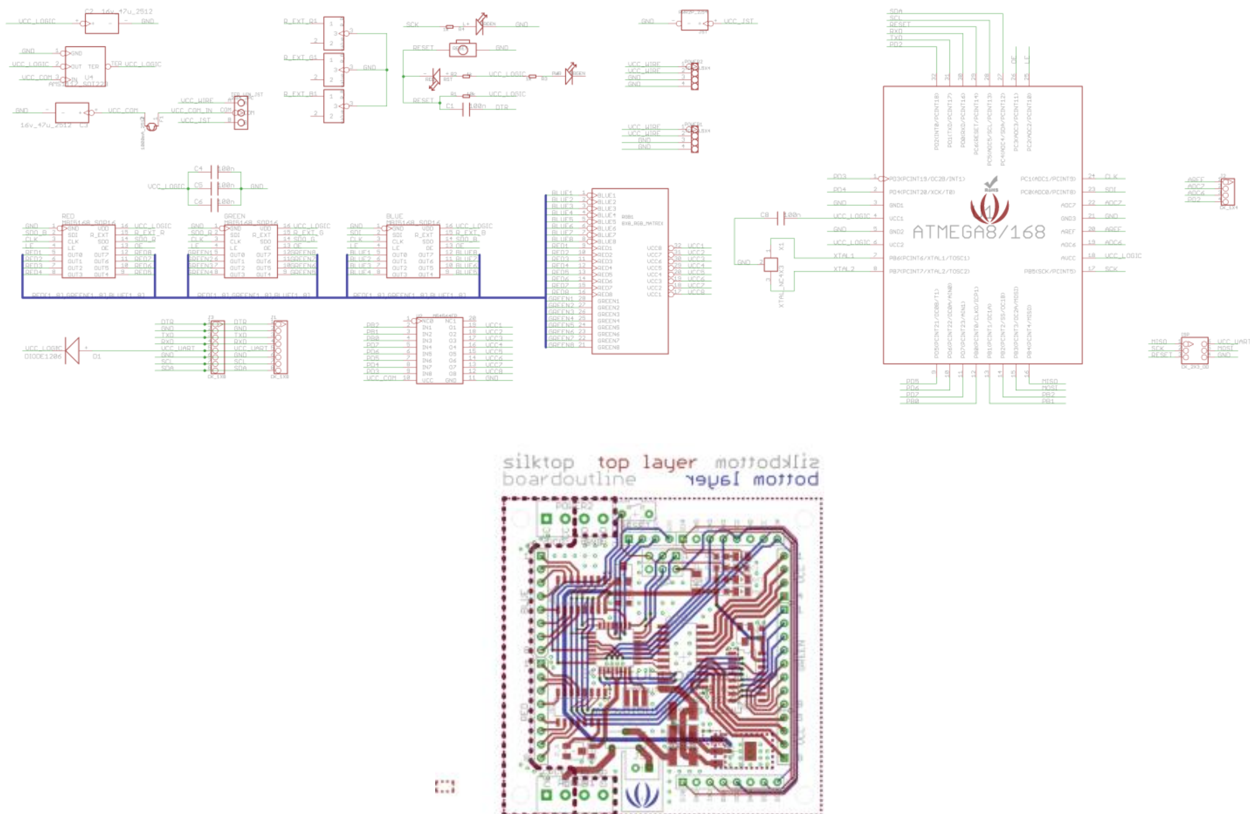

Figure 16. Schematic and Board Layout of Layer 3.

3.7.4 Layer 4: External Memory

Finally, the fourth layer contains the external memory, the audio amplifier, the audio switching circuit and the communications interface. [Figure 17]

Layer #4 is implemented using a two sided board. This approach allows a compact size as well as eases the connection between layer #3 and #4, and layer #4 and the wire connectors.

Ground plane and power plane are included to reduce noise especially in the audio amplifier subsystem. The audio layout was implemented following the indications from the TPA751 datasheet [22] and the user's guide [23] provided by Texas Instruments.

The upper section of the board allocates the communication connectors as wells as the buffer components. On the left upper corner a hole is included to allow the pass of the speaker wire to be connected to the audio amplifier. Moreover, the lower right corner contains the audio switch subsystem and test-pads, and the lower left corner includes the external memory and the pinhead connector for layer #3.

The design was implemented using EAGLE cadsoft, and was custom made to match the length and the width of the rest of the layers in the system.

The inclusion of an external EEPROM in combination with the 2KB of SRAM memory from the ATMEGA328 gives sufficient memory to pre-store the necessary symbol/sequences for a particular experimentation session.

The values of intensity and color are read by the system from memory using a double-buffer (also known as ping-pong buffer) (Multiple_Buffering, 2010). In order to display a particular stimuli, the microcontroller will read from one of the buffers of the double-buffer, then, if the stimuli have to be changed, the second buffer of the double-buffer is modified with the new values of intensity and color, and all the new values are finally written and processed

upon which the buffers are switched. This double-buffer technique allows the system to display stimuli, and at the same time, to modify the stimuli without corrupting the system.

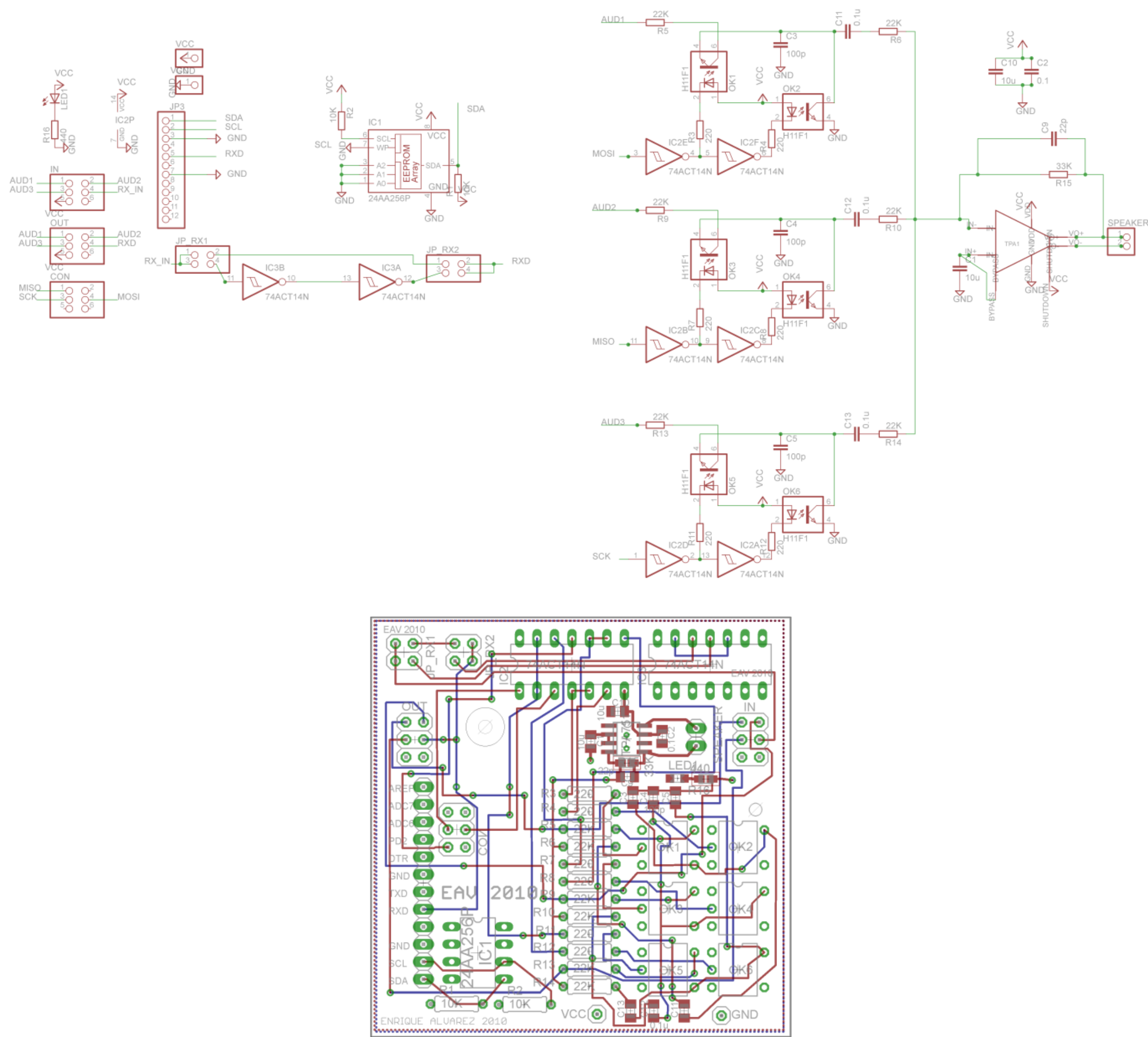

Figure 17. Schematic and Board Layout of Layer 4.

3.8 Summary

In this chapter, the hardware for each cell in the Audio Visual Display device was presented. This hardware allows a microcontroller to drive each of 25 LEDs with 16 levels of grey as well as the on/off state of the audio speaker. In the following chapter, the software which controls the functioning of each cell as well as monitoring the data bus will be presented.

4. SOFTWARE DESIGN

4.1 Introduction

The Audio Visual Device is a tool to facilitate investigation into spatial and time-related issues pertaining to audio and visual stimuli. Specifically, it is designed to allow investigation into Event Related Potential (ERP) triggering.

ERPs are time-locked samples buried in the continuous electroencephalogram (EEG). One of the key issues in ERP research has always been the pursuit of perfect chronometry with a precision in the millisecond range. ERPs inform theories about cognitive operations and have distinct deflections that emerge as early as 20 ms after stimulus presentation. For this reason, the timing of stimulus presentation and recording of time-locked activity is of paramount importance and instrumentation based delays of more than 1-2 ms are unacceptable.

4.1.1. Software Requirements

In order to facilitate ERP investigation, the software for the AVD must meet the following requirements:

- Each of the 55 cells must be controllable separately.
- The time lag between receiving a command and changing the audio or visual output must be less than 1ms,
- The delay between an ERP trigger and changes in the audio or visual output must be less than 1ms
- If two cells are to respond at the same time, there must be no more than a 1ms time skew between the audio or visual outputs.
- Trials of at least 5 minutes must be supported, and
- The system shall be able to store at least 10 visual symbols

To meet these requirements, the software for the AVD is divided into two portions:

- First, each cell is connected to a network to a central Control System (CS). From this network, commands are received, defining how each cell is to behave at each step in the experiment to be conducted.
- Second, from this network, synchronizing commands are sent. These commands address all cells and allow all cells to respond synchronously to a command.

Each cell has its own control software. This software drives the LEDs and the speakers as dictated from the CS.

In this chapter, the software which controls each cell is presented.

4.2 Communications Network

The Audio Visual Display contains two principal components: the control system (CS), which coordinates the different stimuli to be presented at different times, and the audio-visual display (AVD), which is a device that combines software and hardware and physically displays the stimuli.

The CS is intended to be a computer with an experimental control software such as Presentation from neurobs.com, which is widely used in the Center of Visual and Cognitive Neuroscience. The developed design is a unidirectional system where the control system is the transmitter and the audio-visual display is the receiver. The control system sends serial commands at 300 Kbps, (8 bits of data, none parity, 1 stop bit) using a high performance National Instruments PCI serial interface.

4.2.1 Format for Serial Commands

Every serial command consists of 8 bytes (64 bits). The format of each command is presented in Figure 18:

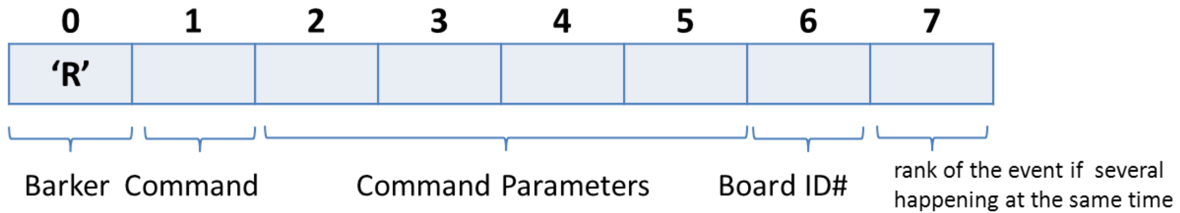

Figure 18. Format of Serial Commands.

There is a distinct command for every task the audio-visual display can perform. Examples of the tasks sent from the CS to the audio-visual display are shown in the following table:

LoadImage()	Loads a symbol from the EEPROM to the microcontroller RAM
ShowImage2()	Displays a visual symbol reading from the EEPROM
ShowImageWithAudio()	Displays a visual symbol and opens an audio channel at the same time
ShowImage5()	Displays a symbol reading from a RAM loaded symbol
SetPixelInImage()	Writes a pixel in a symbol which is going to be saved in the EEPROM
SetJustOnePixel()	Displays just one pixel in the 5 x5 matrix
SetAudio()	Set ON or OFF a combination of audio channels
ClearDisplay()	Displays no symbol

Table 1. Software Commands.

For more detailed information about the functions of the CS and the firmware of the individual cells that compose the audio-visual display array, please refer to APPENDIX A and APPENDIX B.

Some of the commands are general and apply to all the cells in the display, such as: LoadImage(). The rest of the commands only affect a concrete number of cells in the matrix. These individual commands contain a unique identifier of the cell that is sent to. Therefore, once the control system inputs the command in the communications bus, only the specific cell that is sent to reacts to the command.

4.2.2. Processing Times

Once a command is received by a cell, it must be decoded and executed. Each command takes 233 μs to transmit:

(2bits +8 bits/byte) x (8 bytes / message) / (300 kbps) = 266 μs

Once received, each command is processed. The processing times were measured as follows. First, a command was sent from the CS to a cell. Next, a bit (output pin) was set at the start of the routine which processed the command, and cleared upon return. The resulting execution time could then be measured on a digital analyzer with a result presented in Figure 19.

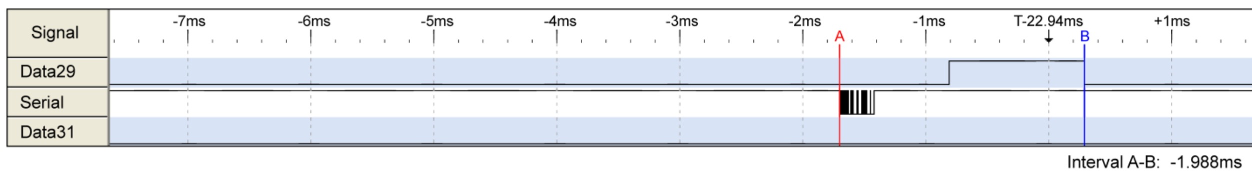

Figure 19. Delay from an Instruction as Measured Using a Logic Analyzer.

The resulting execution times are presented in Table 2.

A detailed explanation of each command and how the software drives the hardware for each command is presented in the following section.

Command	Processing Time
LoadImage() –buffer	3.21 ms
ShowImage2()	3.9 ms
ShowImageWithAudio()	1.36 ms
ShowImage5()	1.35 ms
SetPixelWithAudio()	1.21 ms
SetJustOnePixel()	1.18 ms
SetAudio()	0.95 ms
ClearDisplay()	0.95 ms

Table 2. Command Processing Times.

4.3 Command Processing

The commands are received using the microcontroller's serial port. When the microcontroller detects an ASCII 'R' byte it starts storing the next following 7 bytes. Once

stored, the firmware selects the assigned process depending on the command code received in the message as presented in Figure 20. Every command has a function assigned.

4.3.1 LoadImage()

This function loads a symbol from the EEPROM to the microcontroller's RAM.

The EEPROM is paginated in 128 byte pages. TheLoadImage() function has two input parameters: the symbol's ID number to be uploaded into RAM and the symbol's number in EEPROM. As a result, the LoadImage() function allocates the samples obtained from the EPROM into a multidimensional variable in the microcontroller RAM memory.

Every symbol is represented as a three-dimensional matrix with 3 colors, 5 rows, and 5 columns as presented in Figure 21.

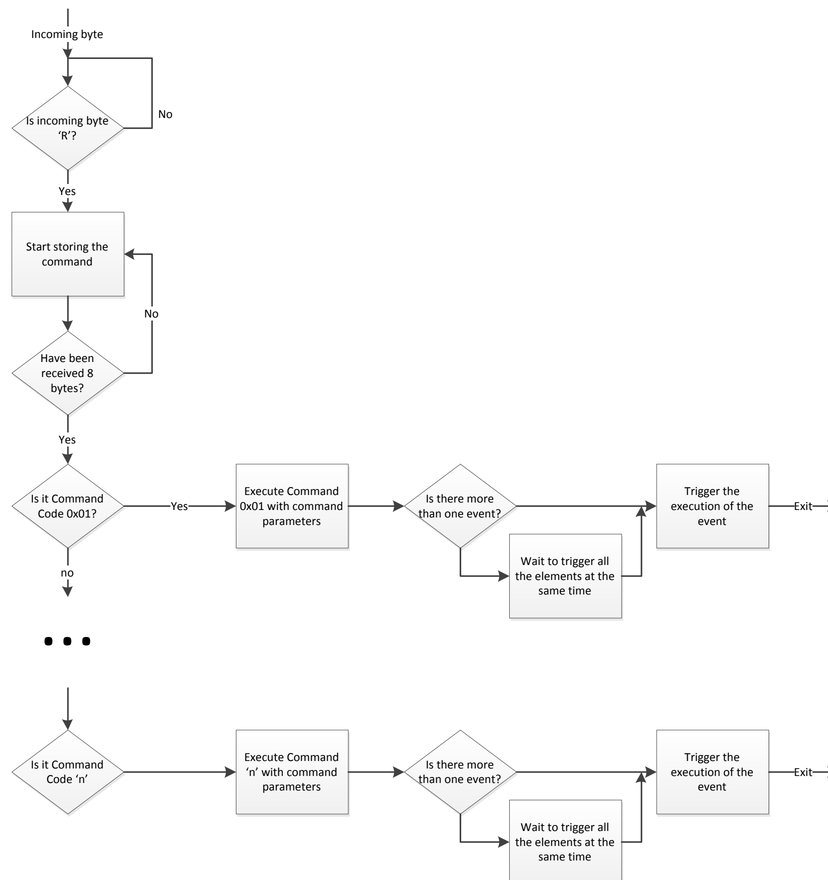

Figure 20. Command Processing Flow Chart.

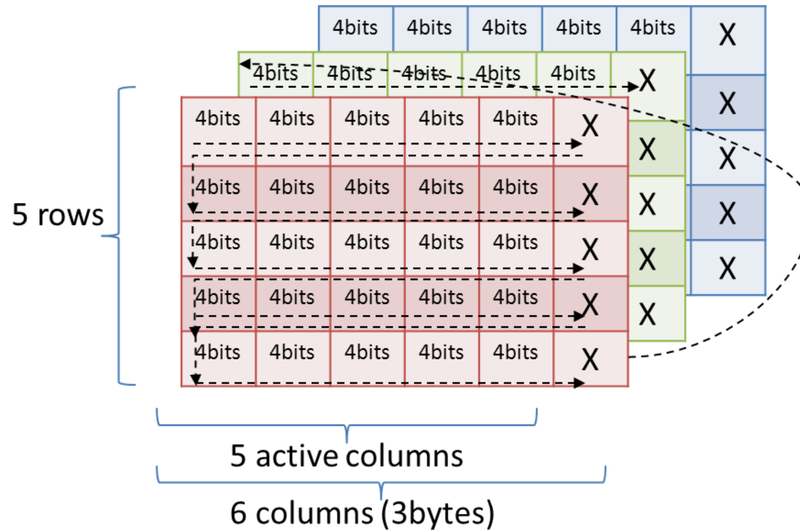

Figure 21. Stored Symbol in Memory.

One pixel is defined with 3 colors with 4 bits each color. The total number of bites required to store one image is then:

$$\# \, Total \ of \ bits = 3 \ colors * 5 \ rows * 5 \ columns * 4 \ bits = 300 \ bits$$

The reading and writing functions related to the EEPROM are defined in bytes. Hence, there must be an integer number of bytes in the defined matrix. In this particular case the number of columns had to be rounded up to 6 to complete 3 bytes. Likewise, the actual number of bits used in memory to store an image is:

$$\# \, Total \ of \ bits = 3 \ colors * 3 \ bytes * \frac{8 \ bits}{1 \ byte} * 5 \ rows = 360 \ bits$$

Due to the EEPROM used having pages predefined as 128 bytes, each page stores one image – meaning that 128 bytes are available for an image even though only 45 bytes are required to store 360 bits. While dedicating an entire page to one image is somewhat wasteful, it simplifies the memory system. Moreover, with 512Mb of memory, over 500,000 symbols can be stored. This is considered sufficient at present.

An address is assigned to every byte memory. When reading or writing an entire symbol, memory addresses are accessed sequentially, in other words, address #0, then address #1, then address #2, etc. The three-dimensional array is transposed in memory color by color by color:

- address #0 = first 8 bits – red color, …, address #15 = last 8 bits –red color
- address#16 = first 8 bits –green color, …, address #30 = last 8 bits –green color
- address#31= first 8 bits –blue color, …, address #45 = last 8 bits-blue color

The direction is graphically displayed in Figure 21 by the dotted black line.

In addition, the EEPROM comes paginated with 128 bytes page-size, so, in order to simplify the design, a 3-color symbol was assigned to an entire 128-byte page.

Figure 22 shows how the pin #27 from the ATMEGA 328 is used as the serial data line (SDA) for bidirectional communication with the EEPROM. The I2C address on the EEPROM by default is 0x50 and can be found in APPENDIX B.

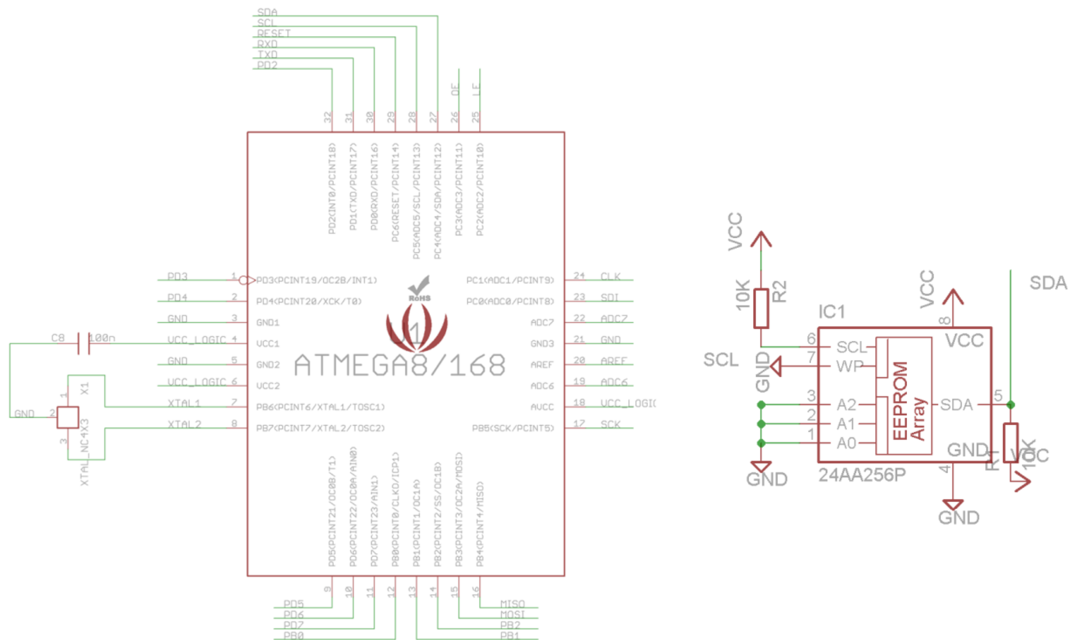

Figure 22. Microcontroller and Memory Connections.

4.3.2 ShowImage2()

ShowImage2() displays a visual symbol reading from the external EEPROM memory. This function takes 2 inputs: first, the number of the symbol to be displayed, and second, a shift value indicated in 4 bits.

The shift value is used to display a symbol shifted horizontally in an integer number of dots. Figure 23 shows how the subroutine performs: at first the microcontroller detects that a new command with the code 0x04 was sent, which triggers the execution of ShowImage2(). Once triggered, the parameters of shift value and symbol # are taken into effect, the EPROM is accessed with the symbol #, and the shift is applied to the recovered data. Finally the subroutine checks if there are more events queued and triggers the display of the symbols in the 5x5 led matrix.

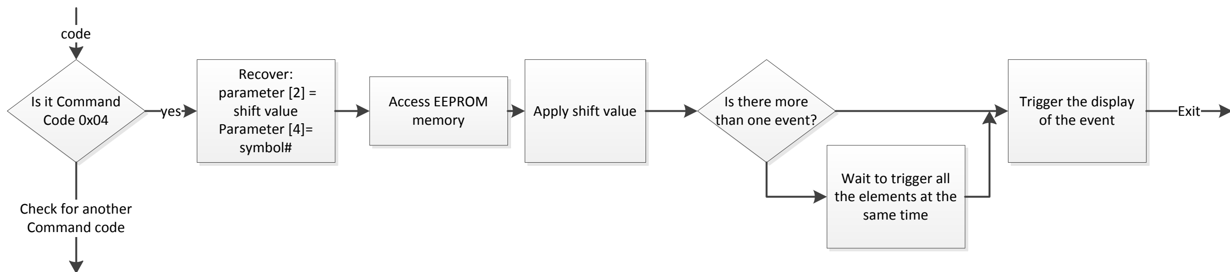

Figure 23. Flow Chart for ShowImage2().

The code of ShowImage2() is included in APPENDIX B - Main Cell Program.

4.3.2.1. Symbol Processing

When presenting a symbol, where a symbol is a set of points in a particular order with a specific color and intensity [Figure 24], the system does not need to send a command for every point in the display. If this were the case, the communications protocol would incur non-acceptable delays in ERP studies.

Instead, just one single command (56 bits) is necessary for symbol presentation. Symbols are pre-loaded into memory before the experiment starts, and every symbol is stored in memory using a unique identifier. To transfer a symbol from the CS and save it in the EEPROM of the recipient cells takes 1.2 sec / symbol.

$$1\ command \times \frac{7\ bytes}{1\ command} \times \frac{(2+8)\ bits}{1\ byte} \times \frac{1sec}{300000bits} + 20\ ms \quad delay\ input\ CMDslow + EEPROM \quad \cong 20.2\ ms$$

$$20.2\ ms \times \frac{60\ bytes}{symbol} \cong 1.2\ sec$$

This represents a great advantage considering the previous model that could take up to 5 to 10 min of preloading time. The individual cells, with the current configuration of 512KB of ROM, can store a maximum 4000 symbols and 44 symbols can be loaded in RAM. The loading time from ROM to RAM consumes 3.7ms per symbol.

4.3.2.2 RAM Access for Displaying a Symbol

While the experiment is running, the microcontroller simply accesses the RAM memory using the mentioned identifier and displays the particular symbol. For example, three possible symbols are presented in Figure 24. These can be selected by sending the command ShowImage2(0), ShowImage2(1), or ShowImage2(3). Accessing the internal memory takes 460µs.

There are cases in ERP experiments where several stimuli have to be presented at the same time in different locations; the actual design allows this feature, even though it is based on a serial communications protocol.

The present design allows for different events to occur at the same time. When this happens, due to the sequential nature of the solution, the system maintains a count of the number

of commands sent and then stacks them in a queue until the last command is sent. After the last command, all the events get processed.

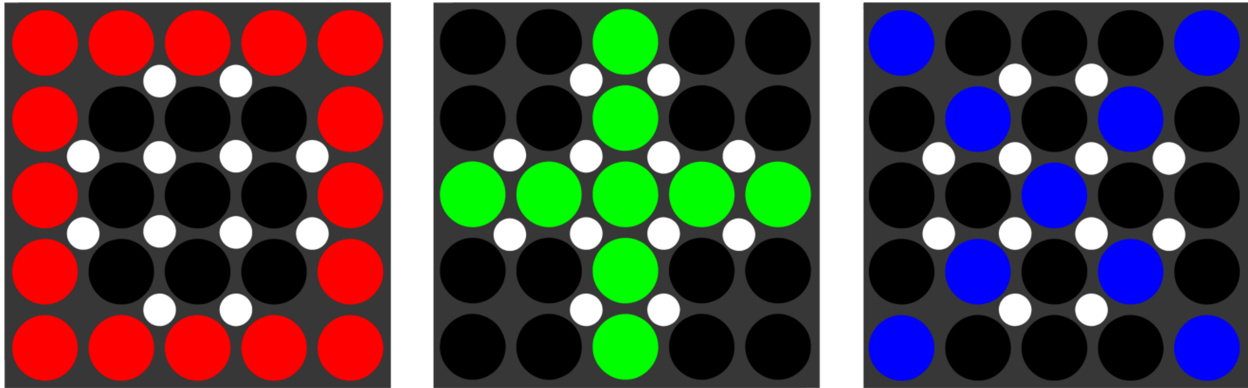

Figure 24. Developed Custom Symbols.

The control system writes the order in one of the bytes of the command, and then the audio-visual display applies a variable delay proportional to this order [Figure 33].

4.3.3 ShowImageWithAudio()

The ShowImageWithAudio() command displays a visual symbol and opens an audio channel at the same time. The symbol is accessed from RAM memory.

The subroutine takes the following inputs:

- The symbol number ID to be displayed
- The horizontal shift value if any
- The audio channels to be activated

These are then decided with the algorithm presented in Figure 25.

The microcontroller receives a command with the ID 0x09 which corresponds to ShowImageWithAudio(). Once received, it recovers the three input parameters, gets the symbol from a multidimensional variable in RAM making use of the symbol ID (the entire code is attached in APPENDIX B- Main Cell Program), after that it applies whatever horizontal shift has

been specified and checks if there is more than one events happening at the same time. If so, it waits to trigger all the events at the same time. If not, it triggers the audio and visual stimuli. The microcontroller displays the audio and the visual stimuli at the same time.

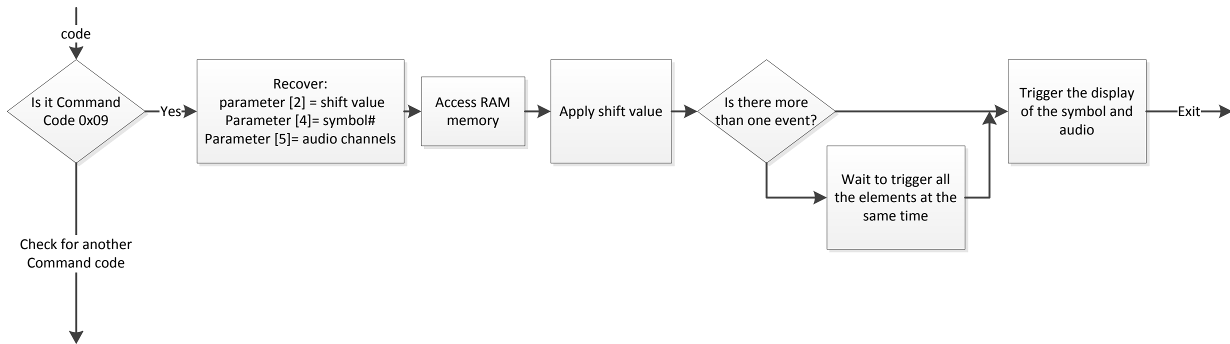

Figure 25. Flow Chart for ShowImageWithAudio().

4.3.4 ShowImage5()

Command ShowImage5() displays a visual symbol read from a RAM loaded symbol. This function utilizes two input parameters: the symbols ID and the horizontal shift value. The structure of ShowImage5() is similar to ShowImageWithAudio(), with the only difference being only visual stimuli are presented. Hence, the code accommodating the audio enabling/disabling in ShowImageWithAudio() is discarded as presented in Figure 26.

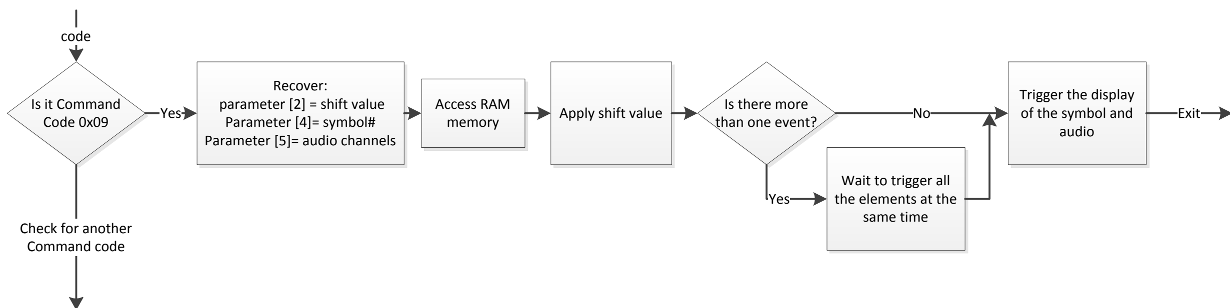

Figure 26. Flow Chart of ShowImage5().

4.3.5 SetPixelInImage()

Command SetPixelInImage() writes a pixel in a symbol which is going to be saved in the EEPROM. When this function is called there are 3 subroutines that take effect: ShowItsReceivingON(), setPixel() and clearDisplay (please refer to APPENDIX B- Main Cell Program for accessing the code).

ShowItsReceivingON() and ClearDisplay() are used to blink the central led of the 5x5 matrix leds while the microcontroller is saving data in the EEPROM. This is used as a simple tool for debugging.

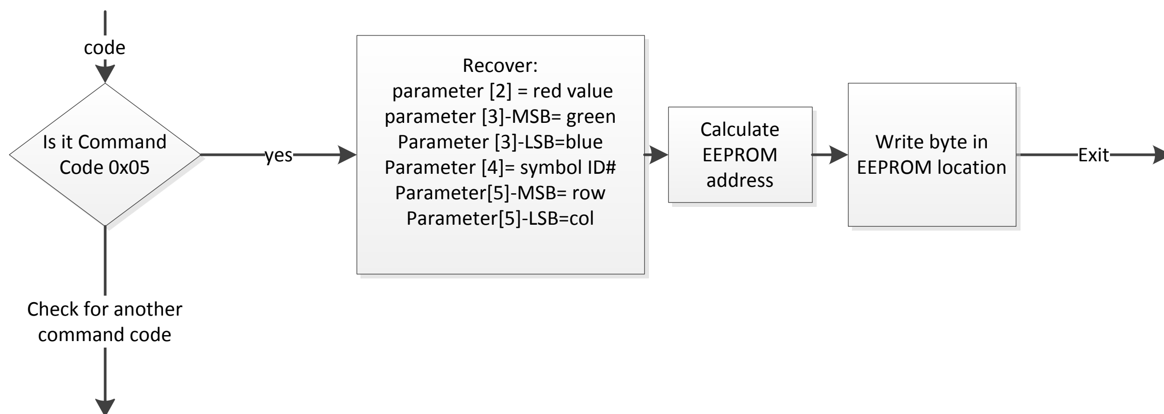

Figure 27. Flow Chart for SetPixelInImage().

SetPixel() is the actual routine that saves the pixel data in the EEPROM, it takes the following input parameters:

- 4 bits for red color
- 4 bits for green color
- 4 bits for blue color
- ID symbol number
- 4 bits for the row number
- 4 bits for the column number

The function saves the RGB color data of a particular pixel in a determined symbol with a specific row and column coordinates as presented in Figure 4-9. When the corresponding command code, 0x05, is detected by the microcontroller, it retrieves the data from the message. With this data it calculates the proper address in EEPROM memory. The memory follows the structured previously shown in Figure 27.

4.3.6 SetJustOnePixel()

Command SetJustOnePixel() displays just one single pixel in the 5 x5 matrix led display. The subroutine takes the following input parameters:

- 4 bits for red color
- 4 bits for green color
- 4 bits for blue color
- 4 bits for the row number
- 4 bits for the column number

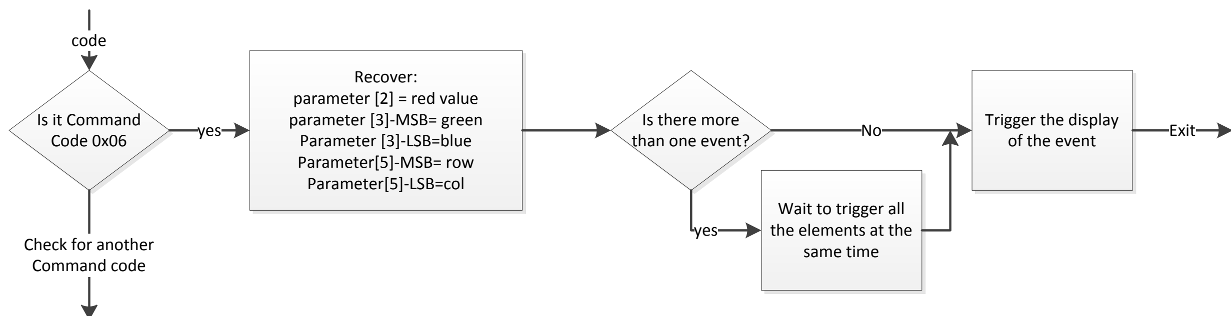

Figure 28. Flow Chart for SetJustOnePixel().

Note that in this case the pixel is not saved in memory; it is just displayed on the LED matrix as presented in Figure 28. When the corresponding code is received, 0x06, the input parameters are retrieved and the function checks to see if there is more than one event happening

at the same time. If so, it waits to trigger all the events at the same time. If not, it immediately triggers the event. A single pixel in the led indicated by the row and column coordinates is light on.

4.3.7 SetAudio()

Command SetAudio() sets ON or OFF a combination of audio channels. It takes only one input parameter, which is 3 bits following the assignation on table where the least significant bits are assigned to every audio channel in Table 3.

If the bit is 0 then the audio channel in disabled, and when the assigned bit is 1 then the audio channel is enabled. Therefore with 3 channels, there are $2^3 = 8$ combinations on open/closed audio channels.

Bit #0 -LSB	Audio Channel #0
Bit #1	Audio Channel #1
Bit #2	Audio Channel #2

Table 3. Bit Assignment for Audio Channels.

Figure 29 displays the flow chart for the setAudio() subroutine. When the command code 0x07 is detected the setAudio() routine begins. Once begun, it retrieves the input parameter with the indicated audio channels, checks for multiple events, and triggers the pass or cut of the audio signal.

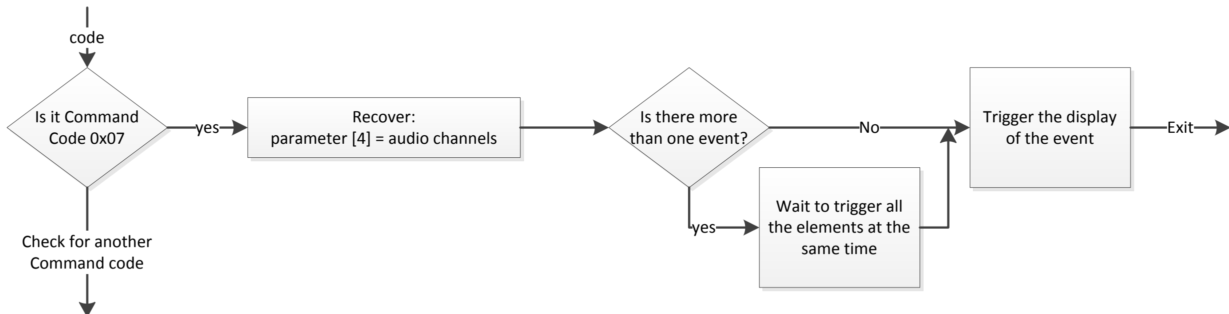

Figure 29. Flow Chart for SetAudio().

Regarding the hardware connections, the microcontroller uses the digital outputs MOSI, MISO and SCK (pins 15, 16, and 17 respectively) in order to control the opto-audio control sub-cells as presented in [Figure 30].

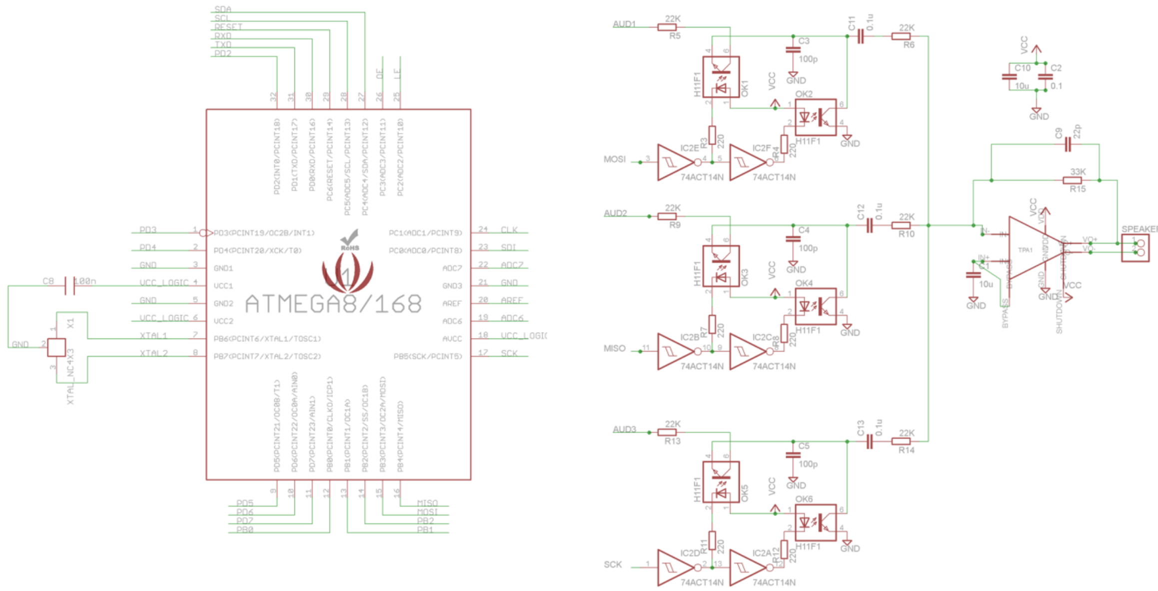

Figure 30. Audio Hardware Schematic.

4.3.8 ClearDisplay()

Command ClearDisplay() clears whatever symbol is being displayed in the 5x5 led matrix. This subroutine is used in SetPixelInImage() but can also be used standalone.

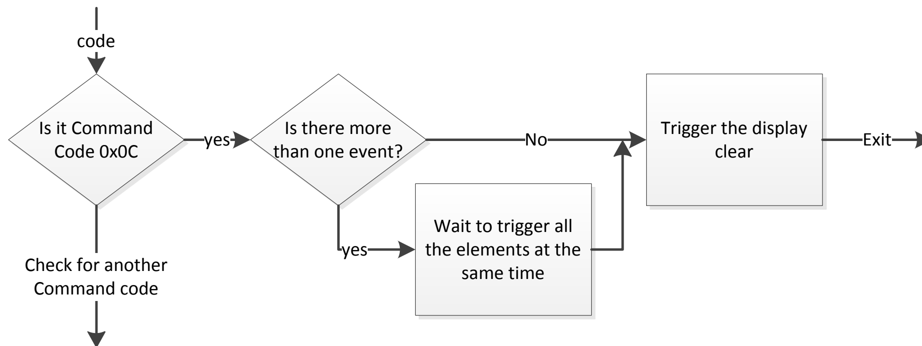

Figure 31. Flow Chart for ClearDisplay().

The function doesn't take any input parameters: it simply sets every pixel in the LED matrix with zeros as presented in Figure 31. If there is more than one cell that has to be cleared, several events can be programmed and triggered to clear the display at the same time.

4.4 Displaying Simultaneous Stimuli

The process of displaying the stimuli requires a variable time depending on the task. Complex tasks such as showing a symbol and audio stimuli will take more time than simple tasks, such as closing an audio channel. In order to keep consistency in the ERP triggering, all these variable latencies are normalized to an integer number of milliseconds, in this case 2 ms. Therefore from the start of sending the command until the command is fully processed the elapsed time will be 2 ms. Knowing this consistent time we can program the ERP trigger at specifically 2 ms, within a temporal error range of 0-70 µs presented in [Figure 32].

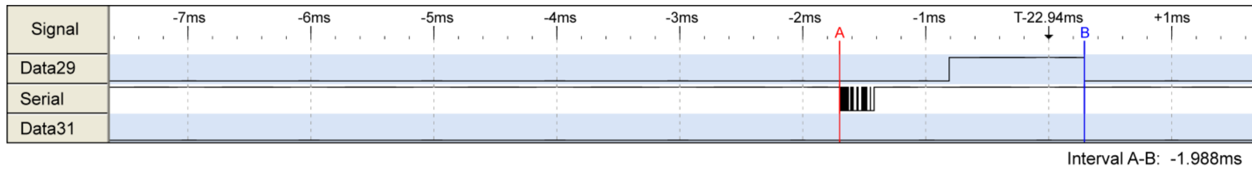

Figure 32. Delay from an Instruction as Measured Using a Logic Analyzer.

4.4.1 Latency for Simultaneous Events

In the case of an experiment requiring six events happening at the "same" time, in a traditional sequential approach can be used to compute the total delay:

$$Delay_{command\#6} = Delay_{command\#5} = Delay_{command\#k} = T_p \quad \text{(Eq. 1)}$$

Where $T_p = T_{proccessing}$ = Time required by the system receives a command and process it. This time is variable with every command.

Therefore after each of the sequential commands (command#1, command#2, etc.) it will take T_p to show every symbol after each command. This creates a delay between the display of the first

symbol in the sequence and the consecutive symbols. The greatest value of this delay is achieved considering the first symbol and the last, so for a ‘n’ number of symbols we would have:

$$\min(Delay_{1st\ vs\ last}) = (n-1) * T_p \quad \text{(Eq. 2)}$$

However, using the approach proposed in this paper, we have eliminated this uncertain time delay. Again, in the case of an experiment requiring six events happening at the same time, the delays can be estimated using the following formula:

$$Delay_{command\#6} = T_p + K \approx 2\ msec \quad \text{(Eq. 3)}$$

$$Delay_{command\#5} = \Delta_{bC} + T_p + K \approx 2\ msec + \Delta_{bC} \quad \text{(Eq. 4)}$$

$$Delay_{command\#4} = 2 * \Delta_{bC} + T_p + K \approx 2\ msec + 2 * \Delta_{bC} \quad \text{(Eq. 5)}$$

...

$$Delay_{command\#1} = 6 * \Delta_{bC} + T_p + K \approx 2\ msec + 6 * \Delta_{bC} \quad \text{(Eq. 6)}$$

Where:

$\Delta_{bC} = \Delta_{betweenCommands}$ = Time between two consecutive commands.

$T_p = T_{proccessing}$ = Time required by the system receive a command and process it. This time is variable with every command.

K = Constant time

Considering ‘n’ commands can be expressed as follows:

$$Delay_{command\#n} = T_p + K \approx 2\ msec \quad \text{(Eq. 7)}$$

$$Delay_{command\#(n-1)} = \Delta_{bC} + T_p + K \approx 2\ msec + \Delta_{bC} \quad \text{(Eq. 8)}$$

$$Delay_{command\#(n-2)} = 2 * \Delta_{bC} + T_p + K \approx 2\ msec + 2 * \Delta_{bC} \quad \text{(Eq. 9)}$$

…

$$Delay_{command\#1} = n * \Delta_{bC} + T_p + K \approx 2\ msec + n * \Delta_{bC} \quad \text{(Eq. 10)}$$

And the sum of all of them can be summarized as:

$$Total_{Delay_{command}} = \sum_{n=1}^{k} Delay_{command\#n} \approx 2\,msec + n * \Delta_{bC} \qquad \text{(Eq. 11)}$$

$$Total_{Delay_{command}} \propto n \qquad \text{(Eq. 12)}$$

There is still a dependency on the number of commands sent, however, with the delay being proportional to the number of involved cells (which is equal to the number of commands sent). [Eq.12] However, in our approach, all the stimuli will be presented 2ms after the last serial command is sent [Eq.7], allowing highly precise ERP triggering.

The practical approach of this algorithm can be seen in [Figure 33]: Data29 and Data31 are control pins of cell ID#29 and cell ID#31 (two different cells), where the control pins indicate the processing time of the event. A 0 level indicates the event is not being processed, whereas a 1 level indicates the event is currently being processed.

To synchronize events, the CS specifies for every command the number of subsequent commands. Once all commands have been issued, all nodes respond simultaneously. For example, in Section #1, three commands are sent from the CS to the audio-visual display. The first one is sent with the ID# 31, and in consequence, only cell number 31 will react to this message. Command#2 sends a message to cell ID#30 (not represented in the diagram), whereas command #3 sends a message to cell ID#29. Once all the third and final command is sent, all three cells execute their commands. The execution delay dictates the time each cell's display is updated, and is governed by [Eq. 7]. Worst case, the timing difference of the displayed events is 70 µs – significantly less than the 2ms design requirements. Note that the time after the last command [Eq. 7] (Interval A-B in sections 2 and 4) is independent of the number of events presented at the desired time as presented in [Figure 33].

4.5 Summary

In this chapter, the software for the AVD was presented. An explanation of the 8-byte command sent from the Control System to the AVD was presented, along with the command set currently implemented. Finally, the algorithm for implementing each command was presented. The next chapter verifies that the software and hardware work together and meet the overall system requirements.

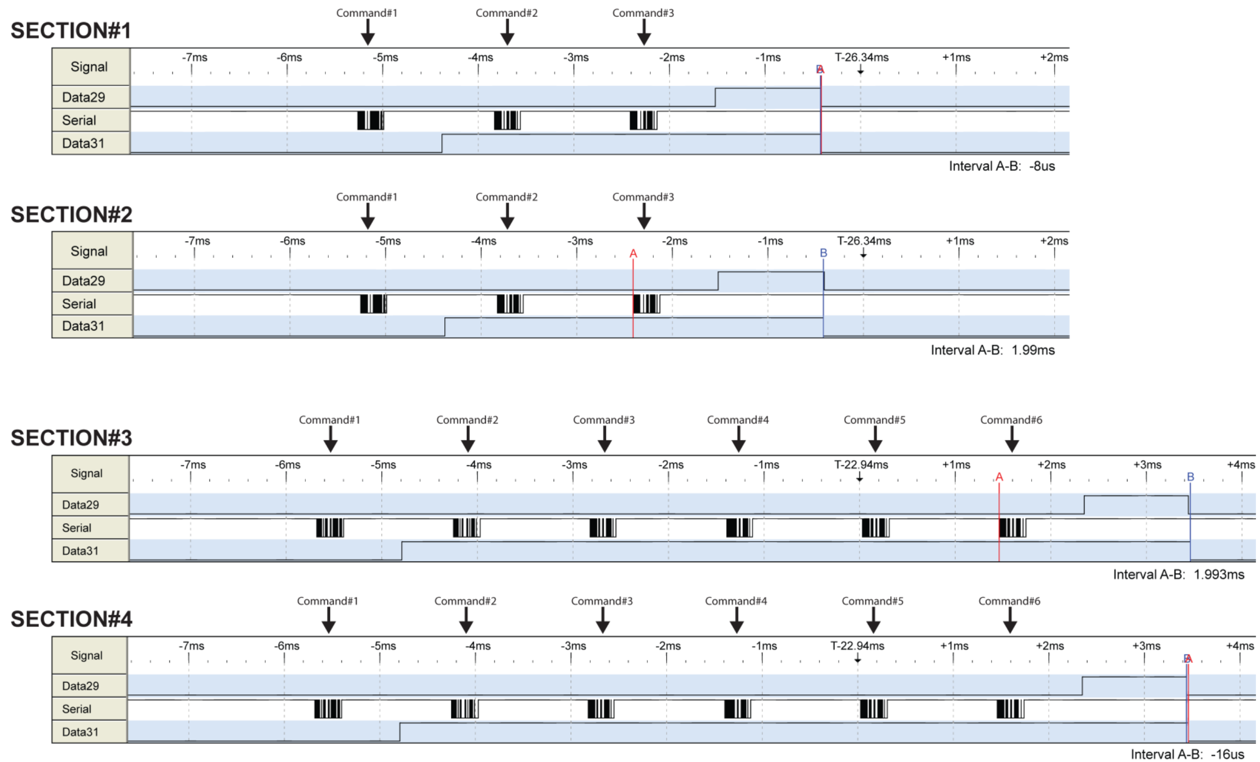

Figure 33. Set of Events being Displayed Simultaneously as Measured on a Logic Analyzer.

5. VERIFICATION

An initial set of 11 cells were implemented, where layers #1, #2, and #4 were designed in the lab. Layer 3 was adopted from Rainbowduino platform, but was modified in software and hardware in order to accommodate the 5 by 5 LED matrix and the requirements of the cross-modal display. After the PCBs were implemented, they were populated and electrically tested for continuity.

Layer #1, which contains 25 RGB surface-mount technology LEDs, was first tested with a pre-programmed rainbow moving pattern in standalone mode. This procedure showed at first glance which components were incorrectly soldered.

Regarding layer #1, it was found that 1 in 10 boards had some minor defect; these defects were fixed, and the modular approach taken for the design made it very easy to interchange layers in the different cells. Layer #2 and #3 didn't show any soldering defects.

Layer #4 contains the audio switches and audio amplifier, as well as the external EEPROM. The communication buffers were checked using a series of tests. A C software routine was developed to test the EEPROM, where the behavior of reading and writing at different speeds and operations over single byte and blocks of arrays were investigated.

The serial port was used to verify the correct functionality of the operations regarding the EEPROM. The results of the operations were satisfactory but it was appreciated that the tasks were time consuming. The measurement to write a pixel from the CS to a particular EEPROM's cell takes approximately 13.4ms if done pixel by pixel and 3.21ms if done by blocks of data.

The timing was measured using an Intronix LOGICPORT 34 channel logic analyzer.

A specific routine was used to test the 3 audio channels and the nine possible combinations.

The communications speed was set to a maximum of 300 Kbps based on speed tests from 500 Kbps in decremented steps of 5Kbps. 300Kbps was found to not have data loss in the communication of symbols.

The basic commands shown in Table 4 were tested in the prototype for timing in every cell, and pins PB3, PB4 and PB5 were used as flag indicators as to when the different tasks were finalized. It was proven that complex tasks (such as show a symbol) were more time consuming than simple tasks.

LoadImage() –buffer	3.21 ms
ShowImage2()	3.9 ms
ShowImageWithAudio()	1.36 ms
ShowImage5()	1.35 ms
SetPixelWithAudio()	1.21 ms
SetJustOnePixel()	1.18 ms
SetAudio()	0.95 ms
ClearDisplay()	0.95 ms

Table 4. Time Required for Every Control Instruction.

After every single cell was tested, they were put together in a connected series which formed a row. The communications protocol was satisfactorily tested sending a set of instructions from the CS to the individual cells in the array.

Power consumption was checked to verify that there wouldn't be any further issues in the scalability of the design. The power consumed by every cell by the specifications manual of Rainbowduino, has a typical value of 600mA [15]. In the tests developed in the CVCN lab, the average milliamps measured in a single cell with the rainbow-pattern lighting up was approximately 130 mA. The power consumed by the cells is directly proportional to the number of LEDs emitting, and with simple symbols the power was expected to be on the 60 mA range.

Summing up, in the cross-modal display system considering there is symbol lighting in every cell, a row will draw 660 mA, and the total system (5 rows) will draw 3300mA. However

based on the nature of the cross-modal experiments that are going to be run with the device, it is more likely that the number of elements lighting at the same time would be in the range 8-10 maximum, which carries a total max of 600mA.

The power measurements were done using a CSI DC regulated bench power supply with a built in digital voltmeter and ampere-meter.

In conclusion, the CS and cell firmware were verified developing several experimentations and stimulation sequences with satisfactory results. However, it was found that the initial calculations of the symbol capacity were lower than the quantity calculated in first place. Instead of a maximum of 44 symbols with 2KB in RAM, it was tested that only a maximum of 20 symbols can be loaded without interfering with the correct functioning of the cell firmware.

5.1 Validation (take 2)

In summary, the Audio Visual Display Device described herein has met all design requirements as presented in [Table 5]:

Design Requirement	Requirement	Actual Design
# cells	55	55
LEDs	1 speaker, 55 RGB LEDs	1 speaker, 55 RGB LEDs
Grey Levels	16	16
Response Time	< 1ms	< 700µs
# Audio Channels	2	3
Volume Levels	2 (on / off)	2
Latency	< 1ms	< 70µs
Programming time	< 1 minute	< 1.5 sec/symbol (typical 6 to 8 symbols < 12 sec)
Session duration	> 3-5 minutes (min. 120 events)	0 to ∞ (0 to ∞ events)
Total Power	< 3A	60mA / cell (typical) 1.5A total (typical)

Table 5. Comparison of Audio Visual Device's Specifications to Design Requirements.

6. FUTURE WORK AND CONCLUSIONS

The cross-modal device proposed in this paper makes it possible to conduct studies where several visual and audio stimuli can be presented in different physical locations providing an excellent temporal and spatial coincidence. The design is structured as a modular/cellular design which allows the system to grow with minor software changes, making it flexible in arranging the cells in different shapes (circular, square, etc.), and facilitating troubleshooting and cell replacement.

In addition, the timing accuracy in the order of microseconds makes this display ideal for ERP studies. The high precision time control can be summarized as:

- 500 μs refresh rate
- 100 ns circuit response
- < 0.7 ms delay for single instruction/command
- < 70 μs difference in response times for simultaneous events
- High speed communications

Furthermore, the audio system already contains three audio generators capable of creating pink noise in different spectrum bands; very desired characteristic for ERP studies. In addition, the audio sub-system contains three sets of opto-audio switches which effectively eliminate all of the unwanted clicks/pops produced when different audio signals are turned on and off. Optical infrared MOSFET switching devices provide isolation between digital control circuitry and the audio signal path, and (due to their high transit frequency of 2 MHz) may be likened to an extremely fast fader instead of an on-off switch. [21]

The system also contains a high capacity memory to store symbols; (approximately 4000 symbols can be stored) which improves the system response in order to display a complex

symbol. This memory is easily accessible and can be upgraded to a greater memory size should highly complex presentation scenarios require such expanded features.

The current design can be improved in several ways, some of which are as follows:

- Greater resolution and more complex symbols. The 5x5 array could be increased in size to allow more realistically accurate images.
- Change the architecture to minimize the delay from the EPR trigger to the response. It should be possible to reduce this delay to less than 100 micro seconds.
- Generate EPR triggers that connected to the EEG recording system installed in the Center of Visual and Cognitive Neuroscience recording chambers, particularly BESA.
- Add volume control rather than simple on/off control. Digital potentiometers similar to the ones used in the previous version are one possibility.
- Increase the number of audio channels.
- Incorporate an independent power supply to eliminate the DC regulated bench power supply presently used.
- Add a graphic user interface to allow different users to develop a specific symbol with different colors and shapes. Then the GUI would automatically create the code for the symbol and event would give the possibility to save it in the cell memory.
- Develop a GUI capable of creating an experiment and generate the commands to send to each cell to initialize the experiment. There might include a timeline in the experiment/trial and the researcher could create different events in an intuitive manner and would not be required to know the intrinsic operations of the system.

7. REFERENCES

[1] Mc Donald, J., Teder, W., Di Russo, F., & Hillyard, S. (2003). Neural Substrates of Perceptual Enhancement by Cross-Modal Spatial Attention. *Journal of Cognitive Neuroscience, 15*(1), 10.

[2] Spence, C., & Driver, J. (1996). Audiovisual links in endogenous covert spatial attention. *Journal of Experimental Psychology: Human Perception and Performance, 22*, 1005-1030.

[3] Stein, B., & Meredith, M. (1993). *The merging of the senses.* Cambridge, MA: MIT Press.

[4] Teder-Sälejärvi, W., Di Russo, F., McDonald, J., & Hillyard, S. (2005). Effects of Spatial Congruity on Audio-Visual Multimodal Integration. *Journal of Cognitive Neuroscience, 17*, 1396-1409.

[5] Teder-Sälejärvi, W. M. (2002). An analysis of audio-visual crossmodal integration by mean of event-related potential (ERP) recordings. *Cognitive Brain Research, 14*, 106-114.

[6] Plant, R. R., & Turner, G. (2009). Millisecond precision psychological research in a world of commodity computers: New hardware, new problems? *Behavior Research Methods*, 41 (3), 598-614.

[7] Sosa, Y., Clarke, A., & McCourt, M. (2011). Hemifield asymmetry in the potency of exogenous auditory and visual cues. *Vision Research*, 1207-1215.

[8] Spence, C., & Driver, J. (1994). Covert spatial orienting in audition: exogenous and endogenous mechanisms facilitate sound localization. *Journal of Experminetal Psychology: Human Perception and Performance, 20*, 555-574.

[9] Driver, J., & Spence, C. (1998). Crossmodal links in spatial attention. *Philosophical Transactions of the Royal Society*, 1319-1331.

[10] Eimer, M., Van Velzen, J., & Driver, J. (2004). ERP Evidence for Cross-Modal Audiovisual Effects of Endogenous Spatial Attention within Hemifields. *Journal of Cognitive Neuroscience, 16*(2), 272-288.

[11] Perrot, D., Costantino, B., & Cisneros, J. (1993). Auditory and visual localization (performance in a sequential discrimination task. *Journal of the Acoustical Society of America, 93*, 2134-2138.

[12] Perrott, D., & Pacheco, S. (1989). Minimum Audible Angle Thresholds for Borad-Band Noise as a Function of the Delay BEtween the Onset of the "Lead" and "Lag" Signals. *Journal of the Acoustical Society of America, 85*, 2669.

[13] Sosa, Y., Teder, W., & McCourt, M. (2010). Biases of spatial attention in vision and audition. *Barin and Cognition*, 229-235.

[14] Glower, J. (2006). *LED / Audio Array.* Fargo, ND: Dept of Electrical and Computer Engineering - NDSU.

[15] Albert Miao, F. X. (2009, 6 23). *RAINBOWDUINO- LED controller manual and datasheet v1.1.* Retrieved from http://www.seeedstudio.com/depot/images/product/Rainbowduino_Manualv1.1.pdf

[16] *ATMEL data sheet* (2011, May). Retrieved from http://www.atmel.com/Images/8271S.pdf

[17] Thibos, L. N. (1989). Image processing by the human eye. *Visual Communications and Image Processing IV, William A. Pearlman (ed), Proc SPIE 1199*, 1148-1153.

[18] *Flicker fusion threshold*. (2010, June 8). Retrieved from http://en.wikipedia.org/wiki/Flicker_fusion_threshold

[19] *Flicker fusion threshold*. (2010, June 8). Retrieved from http://en.wikipedia.org/wiki/Flicker_fusion_threshold: http://en.academic.ru/dic.nsf/enwiki/231030

[20] *Multiple_Buffering*. (2010, June 20). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Multiple_buffering

[21] Teder, W. (1987). Field-effect Optocoupler. *Elector Electronics Great Britain, No. 142, Vol. 13*, 58-61.

[22] *TPA751 data sheet* (2002, October). Retrieved from http://www.ti.com/lit/ds/symlink/tpa751.pdf

[23] *TPA 751 users guide* (2001, February). Retrieved from http://www.ti.com/lit/ug/slou112/slou112.pdf

[24] Mazza, V., Turatto, M., Rossi, M., & Umilta, C. (2007). How automatic are audiovisual links in exogenous spatial attention? *Neuropsuchologia*(3), 514-522.

APPENDIX A. CS FIRMWARE

```
#include <Wire.h>
#define EEPROM_ADDR 0x50        // I2C Buss address of 24LC256 256K EEPROM

unsigned char RainbowCMD[8];
unsigned char State = 0;
unsigned long timeout;
int ledPin=13;
int first=0;
int timeDelaySend=15;

void setup(){
  Wire.begin();                // join I2C bus (address optional for master)
  Serial.begin(300000);
   pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
}

void loop(){
  int test;


  // SHOW ONE COLOR TEST - blue
  /*for(test=15;test>0;test=test-4)
  {
   ShowColor(4,test, 0, 0);delay(500);
   ShowColor(4,0, 0, 0);delay(500);
  }*/
  // END FOF SHOW ONE COLOR TEST - blue

  // EEPROM COMMUNICATION TEST
  /*Serial.println("Writing Test:");
  for (int i=0; i<32; i++){        // loop for first 20 slots
   i2c_eeprom_write_byte(EEPROM_ADDR,i,77-i);
// write address + 65 A or 97 a  77-i
   Serial.print(".  ");
   delay(5);                   // NEED THIS DELAY! (tests suggest it can be as
                            // small as delay(3) -- BBR
  }
  Serial.println("");
  //delay(500);
  Serial.println("Reading Test:");
  for (int i=0; i<32; i++){        // loop for first 20 slots
   Serial.print(i2c_eeprom_read_byte(EEPROM_ADDR, i),BYTE);
   Serial.print(" ");
  }
  Serial.println(" ");
  Serial.println((int)3/2, DEC);
  Serial.println(0%2, DEC);
  Serial.println(" ");*/
  //END OF EEPROM COMMUNICATION TEST

  //BROADCAST SYMBOLS TO EVERY CELL
  //SetPixelInImage(0, 0x0F,0, 1,0,3, 3);  //show image 1
```

```
  if (first==0){
   int symbol=0;

   SetPixelInImage(0, 0,0, symbol,0, 0);
   for(int i=0; i<5;i++){
    for (int j=0; j<5; j++){
  //    if (i==3&& j==3)      SetPixelInImage(4, 0, 0,0x03, 0,i, j);
//SetPixelInImage(Address,blue,red,green,number,rowI,colI)
  //    else if (i==0&& j==0)   SetPixelInImage(4, 0, 0,0x03, 0,i, j);
      if (i==2)      SetPixelInImage(0, 0,0x03, symbol,i, j);  //row 3!
      if (j==2)  SetPixelInImage(0, 0,0x03, 0,i, j);  //colum 3!
      if (i!=2 && j!=2)  SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
      //else SetPixelInImage(4, 0, 0,0, 0,i, j);
      delay(timeDelaySend);
    }
   }

   symbol=1;
   SetPixelInImage(0, 0,0, symbol,0, 0);
   for(int i=0; i<5;i++){
    for (int j=0; j<5; j++){
      if (i==1 || i==3 || j==1 || j==3)  SetPixelInImage(0, 0x03, 0,
symbol,i, j);  //row 3!
      //if (j==1 || j==3)  SetPixelInImage(4, 0x03, 0, 0, 1,i, j);  //colum
3!
      else SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
      //else SetPixelInImage(4, 0, 0,0, 0,i, j);
      delay(timeDelaySend);
    }
   }

   symbol=2;
   SetPixelInImage(0, 0,0, symbol,0, 0);
   for(int i=0; i<5;i++){
    for (int j=0; j<5; j++){//0x00, 0x0D, 0x06  yellow
      if (i==0 || i==4 || j==0 || j==4)  SetPixelInImage(0x03, 0, 0,
symbol,i, j);  //row 3!
      //if (j==1 || j==3)  SetPixelInImage(4, 0x03, 0, 0, 1,i, j);  //colum
3!
      else SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
      //else SetPixelInImage(4, 0, 0,0, 0,i, j);
      delay(timeDelaySend);
    }
   }

   symbol=3;
   SetPixelInImage(0, 0,0, symbol,0, 0);
   for(int i=0; i<5;i++){
    for (int j=0; j<5; j++){//0x00, 0x0D, 0x06  yellow
      if (i==2 && j==0)  SetPixelInImage(0x03, 0, 0, symbol,i, j);  //row 3!
      //if (j==1 || j==3)  SetPixelInImage(4, 0x03, 0, 0, 1,i, j);  //colum
3!
      else SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
      //else SetPixelInImage(0, 0,0, 0,i, j);
      delay(timeDelaySend);
    }
   }
```

```
    symbol=4;
    SetPixelInImage(0, 0,0, symbol,0, 0);
    for(int i=0; i<5;i++){
     for (int j=0; j<5; j++){//0x00, 0x0D, 0x06  yellow
       if (i==2 && j==1)  SetPixelInImage(0x03, 0 , 0, symbol,i, j);  //row 3!
       //if (j==1 || j==3)  SetPixelInImage(0x03, 0, 0, 1,i, j);  //colum 3!
       else SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
       //else SetPixelInImage( 0, 0,0, 0,i, j);
       delay(timeDelaySend);
     }
    }

    symbol=5;
    SetPixelInImage(0, 0,0, symbol,0, 0);
    for(int i=0; i<5;i++){
     for (int j=0; j<5; j++){//0x00, 0x0D, 0x06  yellow
       if (i==2 && j==2)  SetPixelInImage(0x03, 0, 0, symbol,i, j);  //row 3!
       //if (j==1 || j==3)  SetPixelInImage(0x03, 0, 0, 1,i, j);  //colum 3!
       else SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
       //else SetPixelInImage(0, 0,0, 0,i, j);
       delay(timeDelaySend);
     }
    }

    symbol=6;
    SetPixelInImage(0, 0,0, symbol,0, 0);
    for(int i=0; i<5;i++){
     for (int j=0; j<5; j++){//0x00, 0x0D, 0x06  yellow
       if (i==2 && j==3)  SetPixelInImage(0x03, 0x00, 0x00, symbol,i, j);
//row 3!
       //if (j==1 || j==3)  SetPixelInImage(0x03, 0, 0, 1,i, j);  //colum 3!
       else SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
       //else SetPixelInImage(0, 0,0, 0,i, j);
       delay(timeDelaySend);
     }
    }

    symbol=7;
    SetPixelInImage(0, 0,0, symbol,0, 0);
    for(int i=0; i<5;i++){
     for (int j=0; j<5; j++){//0x00, 0x0D, 0x06  yellow
       if (i==2 && j==4)  SetPixelInImage(0x03, 0x00, 0x00, symbol,i, j);
//row 3!
       //if (j==1 || j==3)  SetPixelInImage(0x03, 0, 0, 1,i, j);  //colum 3!
       else SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
       //else SetPixelInImage(0, 0,0, 0,i, j);
       delay(timeDelaySend);
     }
    }

    symbol=8;
    SetPixelInImage(0, 0,0, symbol,0, 0);
    for(int i=0; i<5;i++){
     for (int j=0; j<5; j++){//0x00, 0x0D, 0x06  yellow
       SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
       delay(timeDelaySend);
```

```
    }
   }
   SetPixelInImage(0x03, 0, 0, symbol,1, 1);delay(timeDelaySend);
   SetPixelInImage(0x03, 0, 0, symbol,1, 2);delay(timeDelaySend);
   SetPixelInImage(0x03, 0, 0, symbol,1, 3);delay(timeDelaySend);
   SetPixelInImage(0x03, 0, 0, symbol,2, 1);delay(timeDelaySend);
   SetPixelInImage(0x03, 0, 0, symbol,2, 3);delay(timeDelaySend);
   SetPixelInImage(0x03, 0, 0, symbol,3, 1);delay(timeDelaySend);
   SetPixelInImage(0x03, 0, 0, symbol,3, 2);delay(timeDelaySend);
   SetPixelInImage(0x03, 0, 0, symbol,3, 3);delay(timeDelaySend);

   symbol=9;
   SetPixelInImage(0, 0,0, symbol,0, 0);
   for(int i=0; i<5;i++){
    for (int j=0; j<5; j++){
      if (i==1 || i==3 || j==1 || j==3)  SetPixelInImage(0, 0x03, 0,
symbol,i, j);  //row 3!
      //if (j==1 || j==3)  SetPixelInImage(4, 0x03, 0, 0, 1,i, j);  //colum
3!
      else SetPixelInImage(0, 0,0, symbol,i, j);  //colum 3!
      //else SetPixelInImage(4, 0, 0,0, 0,i, j);
      delay(timeDelaySend);
    }
   }
   //END OF SYMBOL BROADCASTING
   first=1;
   //SetAudio(7, 1);
   //SetAudio(1, 2);
  }
  //SetAudio(7, 1);
  //

  unsigned char symbol=9;

  // TEST #0
  //delay(delayT);
  //SetAudio(0, 1);
  //ClearDisplay(1);
  //ClearDisplay(1);
  //ClearDisplay(1);
  //ShowImage3(9,0,1, 1);
  //ShowImage2(symbol,0,1);
  //SetAudio(audioChannel, 1);

  //delay(delayT);
  //SetAudio(7, 1);
  //setJustOnePixel(0, 15,0,1, 1,1);
  // END OF TEST #0

  //TEST #1
  //ShowImage2(symbol,0,1, 2);  //number, shift, ID_BOARD, numEvents
  //ShowImage2(symbol,0,3, 0);  //number, shift, ID_BOARD, numEvents
  //ShowImage2(symbol,0,2, 0);  //number, shift, ID_BOARD, numEvents
  //ShowImage5(1,0,3,1);  //number of image, shift, ID_BOARD, numEvents
  //ShowImage5(1,0,1,5);  //number of image, shift, ID_BOARD, numEvents
  //ShowImage5(1,0,3,0);  //number of image, shift, ID_BOARD, numEvents
  //ShowImage5(1,0,3,0);  //number of image, shift, ID_BOARD, numEvents
```

```
  //ShowImage5(1,0,3,0);  //number of image, shift, ID_BOARD, numEvents
  //ShowImage5(1,0,3,0);  //number of image, shift, ID_BOARD, numEvents
  //ShowImage5(1,0,2,0);  //number of image, shift, ID_BOARD, numEvents
  //ShowImage5(1,0,3,0);  //number of image, shift, ID_BOARD, numEvents
  //SetAudio(7,1,4);  //number, ID_BOARD,numEvents
  //SetAudio(7,3,0);
  //SetAudio(7,3,0);
  //SetAudio(7,3,0);
  //SetAudio(7,2,0);  //number, ID_BOARD,numEvents
  //setJustOnePixel(0,15, 0, 1, 1, 1, 2); //red, green, blue, rowI, colI,
ID_BOARD, numEvents
  //setJustOnePixel(0,15, 0, 1, 1, 3, 0);
  //setJustOnePixel(0,15, 0, 1, 1, 3, 0);
  //setJustOnePixel(0,15, 0, 1, 1, 3, 0);
  //setJustOnePixel(0,15, 0, 1, 1, 2, 0);

  //ShowImageWithAudio(1,0,1, 1,0);  //number of image, shift,audioChannels,
ID_BOARD, numEvents
  //ShowImageWithAudio(1,0,1, 3,0);  //number of image, shift,audioChannels,
ID_BOARD, numEvents
  //ShowImageWithAudio(1,0,1, 3,0);  //number of image, shift,audioChannels,
ID_BOARD, numEvents
  //ShowImageWithAudio(1,0,1, 2,0);  //number of image, shift,audioChannels,
ID_BOARD, numEvents

  //setJustOnePixelWithAudio(0,15, 0, 1, 1, 1, 1, 0);  //red, green, blue,
rowI, colI, AudioChannels,  ID_BOARD , numEvents
  //setJustOnePixelWithAudio(0,15, 0, 1, 1, 1, 3, 0);  //red, green, blue,
rowI, colI, AudioChannels,  ID_BOARD , numEvents
  //setJustOnePixelWithAudio(0,15, 0, 1, 1, 1, 3, 0);  //red, green, blue,
rowI, colI, AudioChannels,  ID_BOARD , numEvents
  //setJustOnePixelWithAudio(0,15, 0, 1, 1, 1, 3, 0);  //red, green, blue,
rowI, colI, AudioChannels,  ID_BOARD , numEvents
  //setJustOnePixelWithAudio(0,15, 0, 1, 1, 1, 2, 0);  //red, green, blue,
rowI, colI, AudioChannels,  ID_BOARD , numEvents
  //END OF TEST #1


  ClearDisplay(1, 2);// ID_BOARD, numEvents
  ClearDisplay(3, 0);// ID_BOARD, numEvents
  ClearDisplay(2, 0);// ID_BOARD, numEvents

  delay(10);

  // TEST #2
  /*delay(delayT);
  SetAudio(0, 1);
  setJustOnePixel(0, 0,0,0, 0,0x0A);
  //ShowImage3(9,0,1, 1);
  ShowImage2(symbol,0,1);
  SetAudio(audioChannel, 0x0A);

  delay(delayT);
  SetAudio(0, 0x0A);
  setJustOnePixel(0, 0,0,0, 0, 1 );
  //ShowImage3(9,0,1, 2);
  ShowImage2(symbol,0,2);
```

```
SetAudio(audioChannel, 9);

delay(delayT);
SetAudio(0, 9);
setJustOnePixel(0, 0,0,0, 0,2 );
//ShowImage3(9,0,1, 3);
ShowImage2(symbol,0,3);
SetAudio(audioChannel, 8);

delay(delayT);
SetAudio(0, 8);
setJustOnePixel(0, 0,0,0, 0,3 );
//ShowImage3(9,0,1, 4);
ShowImage2(symbol,0,4);
SetAudio(audioChannel, 7);

delay(delayT);
SetAudio(0, 7);
setJustOnePixel(0, 0,0,0, 0, 4 );
//ShowImage3(9,0, 1,5);
ShowImage2(symbol,0,5);
SetAudio(audioChannel, 6);

delay(delayT);
SetAudio(0, 6);
setJustOnePixel(0, 0,0,0, 0, 5 );
//ShowImage3(9,0, 1,5);
ShowImage2(symbol,0,6);
SetAudio(audioChannel, 5);

delay(delayT);
SetAudio(0, 5);
setJustOnePixel(0, 0,0,0, 0, 6 );
//ShowImage3(9,0, 1,5);
ShowImage2(symbol,0,7);
SetAudio(audioChannel, 4);

 delay(delayT);
SetAudio(0, 4);
setJustOnePixel(0, 0,0,0, 0, 7 );
//ShowImage3(9,0, 1,5);
ShowImage2(symbol,0,8);
SetAudio(audioChannel, 3);

 delay(delayT);
SetAudio(0, 3);
setJustOnePixel(0, 0,0,0, 0, 8 );
//ShowImage3(9,0, 1,5);
ShowImage2(symbol,0,9);
SetAudio(audioChannel, 2);

 delay(delayT);
SetAudio(0, 2);
setJustOnePixel(0, 0,0,0, 0, 9);
//ShowImage3(9,0, 1,5);
ShowImage2(symbol,0,0x0A);
SetAudio(audioChannel, 1);*/
```

```
//END OF TEST #2

//*****************************************

//TEST #3
//SetAudio(7, 2);SetAudio(7, 1);

/*delay(50);
ClearAudio(0, 2);SetAudio(1, 1);
//SetAudio(0, 2);SetAudio(0, 1);
delay(100);
ShowImage2(8,0, 1);
ShowImage2(8,0, 2);
//ClearAudio(0, 2);ClearAudio(0, 1);
delay(100);
setJustOnePixel(0x0F , 0,0,0, 1, 1 );
setJustOnePixel(0x0F , 0,0,0, 1, 2 );
delay(100);

ClearAudio(0, 1);SetAudio(1, 2);*/

/*SetAudio(7, 2);SetAudio(7, 1);
delay(1000);*/
/*digitalWrite(11, HIGH);
digitalWrite(12, HIGH);
digitalWrite(13, HIGH);*/
/*
//test corner (0,0)
setJustOnePixel(0x0F , 0,0,0, 0, 1 );
delay(5000);
ShowColor(15 , 15, 15, 1);
delay(5000);
delay(1);
*/
//END OF TEST #3


//TEST #4- SHIFT
/*int symbolX=8;
ShowImage2(symbolX,-5, 2);
ShowImage2(symbolX,0, 1);delay(100);
ShowImage2(symbolX,1, 1);
ShowImage2(symbolX,-4, 2);  delay(100);
ShowImage2(symbolX,2, 1);
ShowImage2(symbolX,-3, 2);  delay(100);
ShowImage2(symbolX,3, 1);
ShowImage2(symbolX,-2, 2);  delay(100);
ShowImage2(symbolX,4, 1);
ShowImage2(symbolX,-1, 2);  delay(100);
ShowImage2(symbolX,5, 1);
ShowImage2(symbolX,0, 2);  delay(100);

//going back
ShowImage2(symbolX,-1, 2);
ShowImage2(symbolX,4, 1);delay(100);
 ShowImage2(symbolX,-2, 2);
ShowImage2(symbolX,3, 1);delay(100);
```

```
    ShowImage2(symbolX,-3, 2);
   ShowImage2(symbolX,2, 1);delay(100);
    ShowImage2(symbolX,-4, 2);
   ShowImage2(symbolX,1, 1);delay(100);
   ShowImage2(symbolX,5, 2);
   ShowImage2(symbolX,0, 1);delay(100); */
   // END ODF TEST #4

   //TEST #5
   /*for (int i=0; i<6;i ++){
    ShowImage2(1,i, 1);  //show image2 number,shift, ID_BOARD
    ShowImage2(1,i, 2);  //show image2 number,shift, ID_BOARD
    delay(500);
   }
   for (int i=5; i>-6; i--){
    ShowImage2(1,i, 1);  //show image2 number,shift, ID_BOARD
    ShowImage2(1,i, 2);  //show image2 number,shift, ID_BOARD
    delay(500);
   }*/
   /*
   for(int s=3; s<8; s++){
    ShowImage2(s,0, 1);  //show image 2; (add , num , shift)
    delay(25);
   }
   for(int s=7; s>2; s--){
    ShowImage2(s,0, 1);  //show image 2; (add , num , shift)
    delay(25);
   }*/
   // END OF TEST #5

   //TEST #6
   /*
   int piv=0;
   int j=0;
   for(int s=0; s<5; s++){
    for(int j=0; j<5; j++){
     setJustOnePixel(0x0F , 0,0,s, j, 1 );
     delay(100);
    }
    s++;
    if(s<5){
     for(int j=4; j>=0; j--){
       setJustOnePixel(0x0F , 0,0,s, j, 1 );
       delay(100);
     }
    }
   }*/
   /*setJustOnePixel(0x0F , 0,0,0, 0, 1 );
   delay(1000);*/
   //END OF TEST #6
}

//----------------------------------------------------------------------------
//Name:LoadImage
//function: Send a command to Rainbowduino for loading a matrix from ROM to R
AM
//parameter: number: #symbol in origin - ROM
```

```
//            numberDestiny:  #symbol in destiny - RAM
//
//---------------------------------------------------------------------------
-
void LoadImage(unsigned char number,unsigned char numberDestiny)
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x01;
   RainbowCMD[4]=number;
   RainbowCMD[5]=numberDestiny;

    SentCMD();
}

//---------------------------------------------------------------------------
//Name:ShowImage2
//function: Send a conmand to Rainbowduino for showing a picture which was pr
e-set in Rainbowduino EEPROM
//parameter:
//            number:  the pre-set picture position
//            shift: the picture  shift bit for display
//            ID_BOARD: board identifier
//            numEvents: number tag for queued events
//---------------------------------------------------------------------------
-
void ShowImage2(unsigned char number,signed char shift, unsigned char
ID_BOARD, unsigned char numEvents)
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x04;
   shift+=5;
   RainbowCMD[2]=(shift<<4);
   RainbowCMD[4]=number;
   RainbowCMD[6]=ID_BOARD;
   RainbowCMD[7]=numEvents;

    SentCMD();
}

//---------------------------------------------------------------------------
//Name:ShowImageWithAudio
//function: Send a conmand to Rainbowduino for showing a picture which was pr
e-set in Rainbowduino RAM with audio added
//parameter:
//            number:  the pre-set picture position
//            shift: the picture  shift bit for display
//            audioChannels: number of channels ON in binary (ex.  0x03-
 channels 1 and 2)
//            ID_BOARD: board identifier
//            numEvents: number tag for queued events
//---------------------------------------------------------------------------
-
void ShowImageWithAudio(unsigned char number,signed char shift,unsigned char
audioChannels, unsigned char ID_BOARD, unsigned char numEvents)
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x09;
```

```
   shift+=5;
   RainbowCMD[2]=(shift<<4);
   RainbowCMD[4]=number;
   RainbowCMD[5]=audioChannels;  //doesnt follow the structure from before
but its more efficient
   RainbowCMD[6]=ID_BOARD;
   RainbowCMD[7]=numEvents;

    SentCMD();
}

//--------------------------------------------------------------------------
//Name:ShowImage5
//function: Send a conmand to Rainbowduino for showing a picture which was pr
e-set in Rainbowduino RAM
//parameter:
//             number:  the pre-set picture position
//             shift: the picture  shift bit for display
//             ID_BOARD: board identifier
//             numEvents: number tag for queued events
//---------------------------------------------------------------------------
-
void ShowImage5(unsigned char number,signed char shift, unsigned char
ID_BOARD, unsigned char numEvents)
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x0D;
   shift+=5;
   RainbowCMD[2]=(shift<<4);
   RainbowCMD[4]=number;
   RainbowCMD[6]=ID_BOARD;
   RainbowCMD[7]=numEvents;

    SentCMD();
}

//--------------------------------------------------------------------------
//Name:SetPixelInImage
//function: Send a conmand to Rainbowduino for setting a pixel in an image pr
e-set in Rainbowduino external EEPROM
//parameter: Address: rainbowduino IIC address
//             red,green,blue:  the color RGB
//             number:  the stored EEPROM picture position (0- first image, 1-
 second image, ...)
//             shift: the picture  shift bit for display
//             rowI and colI: positions of the pixel
//
//             Note: this command doesnt have a ID_BOARD because it is a globa
l command for every board.
//---------------------------------------------------------------------------
-
void SetPixelInImage(unsigned char red, unsigned char green, unsigned char
blue , unsigned char number,unsigned char rowI, unsigned char colI )
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x05;
   RainbowCMD[2]=(red);
```

```
   RainbowCMD[3]=((green<<4)|(blue));
   RainbowCMD[4]=number;
   RainbowCMD[5]=((rowI<<4)|(colI));

    SentCMD_slow();
}

//---------------------------------------------------------------------------
//Name:SetJustOnePixel
//function: Send a conmand to Rainbowduino for setting just one pixel on disp
lay
//parameter:
//            red,green,blue:  the color RGB
//            number:  the stored EEPROM picture position
//            shift: the picture  shift bit for display
//            rowI and colI: positions of the pixel
//            ID_BOARD: board identifier
//            numEvents: number tag for queued events
//---------------------------------------------------------------------------
-
void setJustOnePixel(unsigned char red, unsigned char green, unsigned char
blue,unsigned char rowI, unsigned char colI, unsigned char ID_BOARD, unsigned
char numEvents )
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x06;
   RainbowCMD[2]=(red);
   RainbowCMD[3]=((green<<4)|(blue));
   RainbowCMD[5]=((rowI<<4)|(colI));
   RainbowCMD[6]=ID_BOARD;
   RainbowCMD[7]=numEvents;

    SentCMD();
}

//---------------------------------------------------------------------------
//Name:setJustOnePixelWithAudio
//function: Send a conmand to Rainbowduino for setting just one pixel on disp
lay with audio added
//parameter:
//            red,green,blue:  the color RGB
//            number:  the stored EEPROM picture position
//            shift: the picture  shift bit for display
//            rowI and colI: positions of the pixel
//            AudioChannels: audio channels activated
//            ID_BOARD: board identifier
//            numEvents: number tag for queued events
//---------------------------------------------------------------------------
-
void setJustOnePixelWithAudio(unsigned char red, unsigned char green,
unsigned char blue,unsigned char rowI, unsigned char colI, unsigned char
audioChannels,  unsigned char ID_BOARD ,unsigned char numEvents)
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x0A;
   RainbowCMD[2]=(red);
   RainbowCMD[3]=((green<<4)|(blue));
```

```
   RainbowCMD[4]=audioChannels;
   RainbowCMD[5]=((rowI<<4)|(colI));
   RainbowCMD[6]=ID_BOARD;
   RainbowCMD[7]=numEvents;

    SentCMD();
}

//----------------------------------------------------------------------------
//Name:SetAudio
//function: Send a conmand to Rainbowduino for selecting an audio channel, th
e channel can be activated (1) or deactivated (0)
//parameter:     number: number of the channel to activate/deactivate
//            ID_BOARD: board identifier
//            numEvents: number tag for queued events
//----------------------------------------------------------------------------
-
void SetAudio(unsigned char audioChannels, unsigned char ID_BOARD,unsigned
char numEvents)
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x07;
   RainbowCMD[4]=audioChannels;
   RainbowCMD[6]=ID_BOARD;
   RainbowCMD[7]=numEvents;

    SentCMD();
}

//----------------------------------------------------------------------------
//Name:ClearAudio
//function: Send a conmand to Rainbowduino for closing the audio channels
//parameter:     number: number of the channel to activate/deactivate
//            ID_BOARD: identificator of the board
//            numEvents: number tag for queued events
//
// Notes: dont use this function in future version since its cover by SetAudi
o()
//----------------------------------------------------------------------------
-
void ClearAudio(unsigned char audioChannels, unsigned char ID_BOARD,unsigned
char numEvents)
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x08;
   RainbowCMD[4]=audioChannels;
   RainbowCMD[6]=ID_BOARD;
   RainbowCMD[7]=numEvents;

    SentCMD();
}

//----------------------------------------------------------------------------
//Name:ClearDisplay
//function: Send a conmand to Rainbowduino for clearing the display
//parameter:
//            ID_BOARD: identificator of the board
```

```
//             numEvents: number tag for queued events
//-----------------------------------------------------------------------
-
void ClearDisplay(unsigned char ID_BOARD,unsigned char numEvents)
{
   RainbowCMD[]='R';
   RainbowCMD[1]=0x0C;
   RainbowCMD[6]=ID_BOARD;
   RainbowCMD[7]=numEvents;

   SentCMD();
}

//-----------------------------------------------------------------------
//Name:SentCMD
//function: Send bytes Rainbow conmand out
//parameter: NC
//-----------------------------------------------------------------------
-
void SentCMD()
{  unsigned char OK=0;
    unsigned char i,temp;

   while(!OK)
  {
   switch (State)
   {

   case 0:
    for (i=0;i<8;i++) Serial.write(RainbowCMD[i]);
    State=1;
    break;

   case 1:
    State=2;
    delayMicroseconds(1200);
    break;

   case 2:
    OK=1;
    State=0;
    break;

   default:
    State=0;
    break;

   }
  }
}

//-----------------------------------------------------------------------
//Name:SentCMD
//function: Send bytes Rainbow conmand out -
 slower version to allow the proper writing in EEPROM
//parameter: NC
//-----------------------------------------------------------------------
```

```
-
void SentCMD_slow(){
   unsigned char OK=0;
   unsigned char i,temp;

   while(!OK)
  {
   switch (State)
   {
   case 0:
    for (i=0;i<8;i++) Serial.write(RainbowCMD[i]);

    State=1;
    break;

   case 1:
    State=2;
    delay(5);
    break;

   case 2:
    OK=1;
    State=0;
    break;

   default:
    State=0;
    break;

   }
  }
}


//   External EEPROM - Wathc out this one does work for Microchip EEPROMs
//   -----------------------------------------------------------------------
-

void i2c_eeprom_write_byte( int deviceaddress, unsigned int eeaddress, byte
data ) {
  int rdata = data;
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));    // Address High Byte
  Wire.send((int)(eeaddress & 0xFF));  // Address Low Byte
  Wire.send(rdata);
  Wire.endTransmission();
}

// Address is a page address, 6-bit (63).  More and end will wrap around
// But data can be maximum of 28 bytes, because the Wire library has a buffer
 of 32 bytes
void i2c_eeprom_write_page( int deviceaddress, unsigned int eeaddresspage,
byte* data, byte length ) {
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddresspage >> 8)); // Address High Byte
  Wire.send((int)(eeaddresspage & 0xFF)); // Address Low Byte
  byte c;
```

```
  for ( c = 0; c < length; c++)
   Wire.send(data[c]);
  Wire.endTransmission();
  delay(10);                   // need some delay
}

byte i2c_eeprom_read_byte( int deviceaddress, unsigned int eeaddress ) {
  byte rdata = 0xFF;
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));    // Address High Byte
  Wire.send((int)(eeaddress & 0xFF));  // Address Low Byte
  Wire.endTransmission();
  Wire.requestFrom(deviceaddress,1);
  if (Wire.available()) rdata = Wire.receive();
  return rdata;
}

// should not read more than 28 bytes at a time!
void i2c_eeprom_read_buffer( int deviceaddress, unsigned int eeaddress, byte
*buffer, int length ) {
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));    // Address High Byte
  Wire.send((int)(eeaddress & 0xFF));  // Address Low Byte
  Wire.endTransmission();
  Wire.requestFrom(deviceaddress,length);
  //int c = 0;
  for ( int c = 0; c < length; c++ )
   if (Wire.available()) buffer[c] = Wire.receive();
}
```

APPENDIX B. INDIVIDUAL CELL FIRMWARE

Header – Rainbow.h

```
#ifndef Rainbow_h
#define Rainbow_h
//=============================================
//MBI5168
#define SH_DIR_OE   DDRC
#define SH_DIR_SDI  DDRC
#define SH_DIR_CLK  DDRC
#define SH_DIR_LE   DDRC

#define SH_BIT_OE   0x08
#define SH_BIT_SDI  0x01
#define SH_BIT_CLK  0x02
#define SH_BIT_LE   0x04

#define SH_PORT_OE  PORTC
#define SH_PORT_SDI  PORTC
#define SH_PORT_CLK  PORTC
#define SH_PORT_LE  PORTC
//============================================
#define clk_rising  {SH_PORT_CLK&=~SH_BIT_CLK;SH_PORT_CLK|=SH_BIT_CLK;}
#define le_high    {SH_PORT_LE|=SH_BIT_LE;}
#define le_low    {SH_PORT_LE&=~SH_BIT_LE;}
#define enable_oe  {SH_PORT_OE&=~SH_BIT_OE;}
#define disable_oe  {SH_PORT_OE|=SH_BIT_OE;}

#define shift_data_1    {SH_PORT_SDI|=SH_BIT_SDI;}
#define shift_data_0    {SH_PORT_SDI&=~SH_BIT_SDI;}
//============================================
#define open_line0     {PORTB= (~0x07 & PORTB) | 0x04;}  //this is changed to
preserve the rest of the pins in portB
#define open_line1     {PORTB= (~0x07 & PORTB) | 0x02;}
#define open_line2     {PORTB= (~0x07 & PORTB) | 0x01;}
#define open_line3     {PORTD= (~0xf8 & PORTD) | 0x80;}
#define open_line4     {PORTD= (~0xf8 & PORTD) | 0x40;}
#define open_line5     {PORTD= (~0xf8 & PORTD) | 0x20;}
#define open_line6     {PORTD= (~0xf8 & PORTD) | 0x10;}
#define open_line7     {PORTD= (~0xf8 & PORTD) | 0x08;}
#define close_all_line {PORTD&=~0xf8;PORTB&=~0x07;}
//============================================

#define CheckRequest (g8Flag1&0x01)
#define SetRequest (g8Flag1|=0x01)
#define ClrRequest (g8Flag1&=~0x01)

//=============================================
#define waitingcmd 0x00
#define processing  0x01
#define checking  0x02

#define loadPreloadedCode 0x01
#define showChar 0x02
#define showColor 0x03
```

```
#define showPrefabnicate2 0x04
#define setPixelInImage 0x05
#define setJustOnePixel 0x06
#define setAudioChannel 0x07
#define clearAudioChannel 0x08
#define showPrefabnicate3 0x09
#define setJustOnePixelWithAudio 0x0A
#define getPixelInImage 0x0B
#define clearDisplayCode 0x0C
#define showPrefabnicate5 0x0D
#endif
```

Data.c

```
#include <avr/pgmspace.h>

unsigned char GamaTab[16]=
{0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0
xE7};


//=========================================================
unsigned char dots_color[2][3][8][4] =
{
 {
//=====================================================
  {//green
   {0x00,0x00,0x00,0x4b},
   {0x00,0x00,0x04,0xbf},
   {0x00,0x00,0x4b,0xff},
   {0x00,0x04,0xbf,0xff},
   {0x00,0x4b,0xff,0xff},
   {0x04,0xbf,0xff,0xff},
   {0x4b,0xff,0xff,0xff},
   {0xbf,0xff,0xff,0xfd}
  },
//=======================================================
  {//red
  {0xff,0xfd,0x71,0x00},
   {0xff,0xd7,0x10,0x00},
   {0xfd,0xf1,0x00,0x00},
   {0xda,0x10,0x00,0x00},
   {0x71,0x00,0x00,0x01},
   {0x10,0x00,0x00,0x17},
   {0x00,0x00,0x01,0x7e},
   {0x00,0x00,0x17,0xef}
  },
//======================================================
  {//blue
   {0x06,0xef,0xff,0xff},
   {0x6e,0xff,0xff,0xff},
```

```
    {0xef,0xff,0xff,0xfa},
    {0xff,0xff,0xff,0xa3},
    {0xff,0xff,0xfa,0x30},
    {0xff,0xfa,0xa3,0x00},
    {0xff,0xfa,0x30,0x00},
    {0xff,0xa3,0x00,0x00}
   }
  },
  {
//======================================================
   {//green
    {0xFF,0xFF,0xFF,0x4b},
    {0xFF,0xFF,0xF4,0xbf},
    {0x00,0x00,0x4b,0xff},
    {0x00,0x04,0xbf,0xff},
    {0x00,0x4b,0xff,0xff},
    {0x04,0xbf,0xff,0xff},
    {0x4b,0xff,0xff,0xff},
    {0xbf,0xff,0xff,0xfd}
   },
//========================================================
   {//red
   {0xff,0xfd,0x71,0x00},
    {0xff,0xd7,0x10,0x00},
    {0xfd,0xf1,0x00,0x00},
    {0xda,0x10,0x00,0x00},
    {0x71,0x00,0x00,0x01},
    {0x10,0x00,0x00,0x17},
    {0x00,0x00,0x01,0x7e},
    {0x00,0x00,0x17,0xef}
   },
//=======================================================
   {//blue
    {0x06,0xef,0xff,0xff},
    {0x6e,0xff,0xff,0xff},
    {0xef,0xff,0xff,0xfa},
    {0xff,0xff,0xff,0xa3},
    {0xff,0xff,0xfa,0x30},
    {0xff,0xfa,0xa3,0x00},
    {0xff,0xfa,0x30,0x00},
    {0xff,0xa3,0x00,0x00}
   }
  },
};


unsigned char preLoaded[3][3][8][4] =
{
  {
//=========================0=======================
   {//green
   {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
```

```
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
  },
//==========================================================
  {//red
  {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
  },
//=========================================================
  {//blue
  {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
  }
 },
 {
//=========================1=========================
  {//green
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
  },
//==========================================================
  {//red
  {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
  },
//=========================================================
  {//blue
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
```

```
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
   }
  },

   {
//=========================2=========================
   {//green
   {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
   },
//=======================================================
   {//red
   {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
   },
//======================================================
   {//blue
   {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00},
    {0x00,0x00,0x00,0x00}
   }
  }
};


/*unsigned char Prefabnicatel[5][3][8][4] PROGMEM =
{
  {
//==========================0========================
   {//green
    {0x00,0x00,0x00,0x4b},
    {0x00,0x00,0x04,0xbf},
    {0x00,0x00,0x4b,0xff},
    {0x00,0x04,0xbf,0xff},
    {0x00,0x4b,0xff,0xff},
    {0x04,0xbf,0xff,0xff},
    {0x4b,0xff,0xff,0xff},
    {0xbf,0xff,0xff,0xfd}
   },
```

```
//=========================================================
  {//red
  {0xff,0xfd,0x71,0x00},
   {0xff,0xd7,0x10,0x00},
   {0xfd,0xf1,0x00,0x00},
   {0xda,0x10,0x00,0x00},
   {0x71,0x00,0x00,0x01},
   {0x10,0x00,0x00,0x17},
   {0x00,0x00,0x01,0x7e},
   {0x00,0x00,0x17,0xef}
  },
//========================================================
  {//blue
   {0x06,0xef,0xff,0xff},
   {0x6e,0xff,0xff,0xff},
   {0xef,0xff,0xff,0xfa},
   {0xff,0xff,0xff,0xa3},
   {0xff,0xff,0xfa,0x30},
   {0xff,0xfa,0xa3,0x00},
   {0xff,0xfa,0x30,0x00},
   {0xff,0xa3,0x00,0x00}
  }
 },
 {
//===========================1=======================
  {//green
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00}
  },
//=========================================================
  {//red
   {0x0f,0xff,0xff,0xff},
   {0x0f,0xff,0xff,0xff},
   {0x0f,0xff,0xff,0xff},
   {0x0f,0xff,0xff,0xff},
   {0x0f,0xff,0xff,0xff},
   {0x0f,0xff,0xff,0xff},
   {0x0f,0xff,0xff,0xff},
   {0x0f,0xff,0xff,0xff}
  },
//========================================================
  {//blue
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00}
  }
```

```
 },

  {
//=========================2========================
  {//green
  {0xff,0xfd,0x71,0x00},
   {0xff,0xfd,0x71,0x00},
   {0xff,0xfd,0x71,0x00},
   {0xff,0xfd,0x71,0x00},
   {0xff,0xfd,0x71,0x00},
   {0xff,0xfd,0x71,0x00},
   {0xff,0xfd,0x71,0x00},
   {0xff,0xfd,0x71,0x00}
  },
//=======================================================
  {//red
  {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00}
  },
//======================================================
  {//blue
  {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00},
   {0x00,0x00,0x00,0x00}
  }
 },

  {
//=========================3========================
  {//green
   {0xFF,0xFF,0xFF,0x4b},
   {0xFF,0xFF,0xF4,0xbf},
   {0x00,0x00,0x4b,0xff},
   {0x00,0x04,0xbf,0xff},
   {0x00,0x4b,0xff,0xff},
   {0x04,0xbf,0xff,0xff},
   {0x4b,0xff,0xff,0xff},
   {0xbf,0xff,0xff,0xfd}
  },
//=======================================================
  {//red
  {0xff,0xfd,0x71,0x00},
   {0xff,0xd7,0x10,0x00},
   {0xfd,0xf1,0x00,0x00},
   {0xda,0x10,0x00,0x00},
   {0x71,0x00,0x00,0x01},
```

```
    {0x10,0x00,0x00,0x17},
    {0x00,0x00,0x01,0x7e},
    {0x00,0x00,0x17,0xef}
   },
//=======================================================
   {//blue
    {0x06,0xef,0xff,0xff},
    {0x6e,0xff,0xff,0xff},
    {0xef,0xff,0xff,0xfa},
    {0xff,0xff,0xff,0xa3},
    {0xff,0xff,0xfa,0x30},
    {0xff,0xfa,0xa3,0x00},
    {0xff,0xfa,0x30,0x00},
    {0xff,0xa3,0x00,0x00}
   }
  },


  {
//========================4========================
   {//green
    {0xFF,0xFF,0xFF,0x4b},
    {0xFF,0xFF,0xF4,0xbf},
    {0x00,0x00,0x4b,0xff},
    {0x00,0x04,0xbf,0xff},
    {0x00,0x4b,0xff,0xff},
    {0x04,0xbf,0xff,0xff},
    {0x4b,0xff,0xff,0xff},
    {0xbf,0xff,0xff,0xfd}
   },
//========================================================
   {//red
   {0xff,0xfd,0x71,0x00},
    {0xff,0xd7,0x10,0x00},
    {0xfd,0xf1,0x00,0x00},
    {0xda,0x10,0x00,0x00},
    {0x71,0x00,0x00,0x01},
    {0x10,0x00,0x00,0x17},
    {0x00,0x00,0x01,0x7e},
    {0x00,0x00,0x17,0xef}
   },
//=======================================================
   {//blue
    {0x06,0xef,0xff,0xff},
    {0x6e,0xff,0xff,0xff},
    {0xef,0xff,0xff,0xfa},
    {0xff,0xff,0xff,0xa3},
    {0xff,0xff,0xfa,0x30},
    {0xff,0xfa,0xa3,0x00},
    {0xff,0xfa,0x30,0x00},
    {0xff,0xa3,0x00,0x00}
   }
  }
};*/
```

```
//----------------------------------------------------
/* unsigned char ASCII_Char[52][8] PROGMEM =
{

   {0x0,0x44,0x44,0x7C,0x44,0x44,0x28,0x10},//A
   {0x0,0x3C,0x44,0x44,0x3C,0x44,0x44,0x3C},//B
   {0x0,0x38,0x44,0x4,0x4,0x4,0x44,0x38},//C
   {0x0,0x1C,0x24,0x44,0x44,0x44,0x24,0x1C},//D
   {0x0,0x7C,0x4,0x4,0x3C,0x4,0x4,0x7C}, //E
   {0x0,0x4,0x4,0x4,0x3C,0x4,0x4,0x7C}, //F
   {0x0,0x78,0x44,0x44,0x74,0x4,0x44,0x38},//G
   {0x0,0x44,0x44,0x44,0x7C,0x44,0x44,0x44}, //H
   {0x0,0x38,0x10,0x10,0x10,0x10,0x10,0x38} , //I
   {0x0,0x18,0x24,0x20,0x20,0x20,0x20,0x70},  //J
   {0x0,0x44,0x24,0x14,0xC,0x14,0x24,0x44},  //K
   {0x0,0x7C,0x4,0x4,0x4,0x4,0x4,0x4},//L
   {0x0,0x44,0x44,0x44,0x54,0x54,0x6C,0x44}, //M
   {0x0,0x44,0x44,0x64,0x54,0x4C,0x44,0x44},  //N
   {0x0,0x38,0x44,0x44,0x44,0x44,0x44,0x38},  //O
   {0x0,0x4,0x4,0x4,0x3C,0x44,0x44,0x3C},  //P
   {0x0,0x58,0x24,0x54,0x44,0x44,0x44,0x38},//Q
   {0x0,0x44,0x24,0x14,0x3C,0x44,0x44,0x3C},  //R
   {0x0,0x3C,0x40,0x40,0x38,0x4,0x4,0x78},//S
   {0x0,0x10,0x10,0x10,0x10,0x10,0x10,0x7C},// T
   {0x0,0x38,0x44,0x44,0x44,0x44,0x44,0x44},// U
   {0x0,0x10,0x28,0x44,0x44,0x44,0x44,0x44},// V
   {0x0,0x28,0x54,0x54,0x54,0x44,0x44,0x44},// W
   {0x0,0x44,0x44,0x28,0x10,0x28,0x44,0x44},// X
   {0x0,0x10,0x10,0x10,0x28,0x44,0x44,0x44},// Y
   {0x0,0x7C,0x4,0x8,0x10,0x20,0x40,0x7C},// Z

   {0x0,0x78,0x44,0x78,0x40,0x38,0x0,0x0},// a
   {0x0,0x3C,0x44,0x44,0x4C,0x34,0x4,0x4},// b
   {0x0,0x38,0x44,0x4,0x4,0x38,0x0,0x0},// c
   {0x0,0x78,0x44,0x44,0x64,0x58,0x40,0x40},// d
   {0x0,0x38,0x4,0x7C,0x44,0x38,0x0,0x0},// e
   {0x0,0x8,0x8,0x8,0x1C,0x8,0x48,0x30},// f
   {0x38,0x40,0x78,0x44,0x44,0x78,0x0,0x0},// g
   {0x0,0x44,0x44,0x44,0x4C,0x34,0x4,0x4},// h
   {0x0,0x38,0x10,0x10,0x10,0x18,0x0,0x10},// i
   {0x18,0x24,0x20,0x20,0x20,0x30,0x0,0x20},// j
   {0x0,0x24,0x14,0xC,0x14,0x24,0x4,0x4},// k
   {0x0,0x38,0x10,0x10,0x10,0x10,0x10,0x18},// l
   {0x0,0x44,0x44,0x54,0x54,0x2C,0x0,0x0},// m
   {0x0,0x44,0x44,0x44,0x4C,0x34,0x0,0x0},// n
   {0x0,0x38,0x44,0x44,0x44,0x38,0x0,0x0},// o
   {0x4,0x4,0x3C,0x44,0x44,0x3C,0x0,0x0},// p
   {0x40,0x40,0x58,0x64,0x64,0x58,0x0,0x0},// q
   {0x0,0x4,0x4,0x4,0x4C,0x34,0x0,0x0},// r
   {0x0,0x3C,0x40,0x38,0x4,0x38,0x0,0x0},// s
   {0x0,0x30,0x48,0x8,0x8,0x1C,0x8,0x8},// t
   {0x0,0x58,0x64,0x44,0x44,0x44,0x0,0x0},// u
   {0x0,0x10,0x28,0x44,0x44,0x44,0x0,0x0},// v
   {0x0,0x28,0x54,0x54,0x44,0x44,0x0,0x0},// w
   {0x0,0x44,0x28,0x10,0x28,0x44,0x0,0x0},// x
   {0x38,0x40,0x78,0x44,0x44,0x44,0x0,0x0},// y
   {0x0,0x7C,0x8,0x10,0x20,0x7C,0x0,0x0},// z
```

```
};


unsigned char ASCII_Number[10][8] PROGMEM =
{
  {0x0,0x38,0x44,0x4C,0x54,0x64,0x44,0x38},//0
  {0x0,0x38,0x10,0x10,0x10,0x10,0x18,0x10},// 1
  {0x0,0x7C,0x8,0x10,0x20,0x40,0x44,0x38},// 2
  {0x0,0x38,0x44,0x40,0x20,0x10,0x20,0x7C},// 3
  {0x0,0x20,0x20,0x7C,0x24,0x28,0x30,0x20},// 4
  {0x0,0x38,0x44,0x40,0x40,0x3C,0x4,0x7C},//5
  {0x0,0x38,0x44,0x44,0x3C,0x4,0x8,0x30},//6
  {0x0,0x8,0x8,0x8,0x10,0x20,0x40,0x7C},//7
  {0x0,0x38,0x44,0x44,0x38,0x44,0x44,0x38},//8
  {0x0,0x18,0x20,0x40,0x78,0x44,0x44,0x38}//9
};
*/
```

Main Cell Program

```
#include "Rainbow.h"
#include <Wire.h>
#include <avr/pgmspace.h>

#define EEPROM_ADDR 0x50
#define ID_BOARD 0x02

/*
Modifications:

- Transform it to serial ->OK
- Convert 8x8 to 5x5 matrix ->OK
- Send 6 bytes on communication instead of 5 ->OK
- Write in EEPROM  ->OK
- Messages with a board ID (send 7 bytes instead of 5) -> OK
- implement shift for images
- implement light only one point -> OK
*/

//==============================================================
extern unsigned char dots_color[2][3][8][4];  //define Two Buffs (one for
Display ,the other for receive data)
extern unsigned char GamaTab[16];          //define the Gamma value for
correct the different LED matrix
extern unsigned char preLoaded[3][3][8][4];
//extern unsigned char ASCII_Char[52][8];
//extern unsigned char ASCII_Number[10][8];
//==============================================================
unsigned char line,level;
unsigned char Buffprt=0;
unsigned char State=0;
```

```
unsigned char g8Flag1;
unsigned char RainbowCMD[8]={0,0,0,0,0,0, 0,0};
unsigned int
delayVar[14]={30,70,110,170,210,250,290,310,315,317,318,318,318};  //measured
with logic analyzer
unsigned char toggle=0;

void setup()
{
  _init();
}

void loop()
{
  switch (State)
  {
   case waitingcmd:
   if (Serial.available()>0) receiveEvent(1);  //the one doesnt matter here
   break;

   case processing:
      setPD2();  //flag to measure the processing time
        GetCMD();
      clearPD2();  //flag to measure the processing time
   State=waitingcmd;
   break;
  default:
   State=waitingcmd;
   break;
  }

}


ISR(TIMER2_OVF_vect)        //Timer2  Service
{
  //cli();
  TCNT2 = GamaTab[level];    // Reset a  scanning time by gamma value table
  flash_next_line(line,level);  // sacan the next line in LED matrix level by
level.
  line++;
  if(line>4)       // when have scaned all LEC the back to line 0 and add the
level
  {
   line=0;
   level++;
   if(level>15)     level=0;
  }
  //sei();
}

void init_timer2(void)
{
  TCCR2A |= (1 << WGM21) | (1 << WGM20);
  TCCR2B |= (1<<CS22);  // by clk/64
  TCCR2B &= ~((1<<CS21) | (1<<CS20));  // by clk/64
  TCCR2B &= ~((1<<WGM21) | (1<<WGM20));  // Use normal mode
```

```
  ASSR |= (0<<AS2);     // Use internal clock - external clock not used in
Arduino
  TIMSK2 |= (1<<TOIE2) | (0<<OCIE2B);  //Timer2 Overflow Interrupt Enable
  TCNT2 = GamaTab[];
  sei();
}
/*
OCIE2B bit#2
TOIE2 bit#0
*/
void disableTimer2(void){
  TIMSK2 = 0x00;    //set TOIE2 to 0
}

void enableTimer2(void){
  TIMSK2 |= (1<<TOIE2) | (0<<OCIE2B); ;    //set TOIE2 to 1
}

void setPD2(){PORTD |=_BV(PD2);}    //high
void clearPD2(){PORTD &=~_BV(PD2);}  //low

void _init(void)   // define the pin mode
{
  DDRD=0xff;
  DDRC=0xff;
  DDRB=0xff;
  PORTD=0;
  PORTB=0;
  //Wire.begin(4); // join i2c bus (address optional for master)
  Wire.begin();  //i2c eeprom
  Serial.begin(300000);
  //Wire.onReceive(receiveEvent); // define the receive function for
receiving data from master
  //Wire.onRequest(requestEvent); // define the request function for the
request from maseter
  init_timer2();  // initial the timer for scanning the LED matrix
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
}



void receiveEvent(int howMany)
{
  unsigned char i=0;
  //setPD2();
  while(Serial.available()>0)
  {
   RainbowCMD[i]=Serial.read();
   i++;
  }
  if((i==8)&&(RainbowCMD[]=='R')&&(RainbowCMD[1]==setPixelInImage ||
RainbowCMD[1]==loadPreloadedCode || RainbowCMD[6]==ID_BOARD))
State=processing;
  else    State=waitingcmd;
  //clearPD2();
```

```
}

void DispshowChar(void)
{  //not used in this version
}

void DispshowColor(void)
{  //not used in this version
}

//display image reading from the EEPROM
void DispshowPicture2(void)
{
 unsigned char pi;
  signed char shifts;
  unsigned char color=0,row=0,dots=0;
  unsigned char temp;
  unsigned char fir,sec;
  unsigned int veeaddressX0, veeaddressX1, veeaddressX2, eeaddressX0;
  char value_eeaddressX_h1[20];
  char value_eeaddressX_h2[25];
  char value_eeaddressX[45];  //45+null character added in strcat-46

  shifts=((RainbowCMD[2]>>4)&0x0F);   shifts-=5;
  pi=RainbowCMD[4];
  RainbowCMD[1]=0;

  eeaddressX0=(pi*128);
  //setPD2();

i2c_eeprom_read_bufferChar(EEPROM_ADDR, eeaddressX0, value_eeaddressX_h1, 20)
;
  for (int i=0; i<20;i++){
    value_eeaddressX[i]=value_eeaddressX_h1[i];
  }

i2c_eeprom_read_bufferChar(EEPROM_ADDR, eeaddressX0+20, value_eeaddressX_h2,
25);
  for (int i=20; i<45;i++){
    value_eeaddressX[i]=value_eeaddressX_h2[i-20];
  }
  //clearPD2();

  for(color=0;color<3;color++)
  {
   for (row=0;row<5;row++)
   {

      //shift doesnt work now (maybe in the future if we want to add it)
      //eeaddressX=((15*color)+(3*row)+(dots))+(pi*128);  //15 bytes per
color, 3 byte per row.  Pages of 128

      veeaddressX0=value_eeaddressX[((15*color)+(3*row)+(0))];  //15 bytes
per color, 3 byte per row.  Pages of 128
      veeaddressX1=value_eeaddressX[((15*color)+(3*row)+(1))];  //15 bytes
per color, 3 byte per row.  Pages of 128
      veeaddressX2=value_eeaddressX[((15*color)+(3*row)+(2))];  //15 bytes
```

```
per color, 3 byte per row.  Pages of 128

      switch(shifts){
       case 0:
        dots_color[((Buffprt+1)&1)][color][ow][]=  veeaddressX0;
        dots_color[((Buffprt+1)&1)][color][ow][1] = veeaddressX1;
        dots_color[((Buffprt+1)&1)][color][ow][2] = veeaddressX2;
        break;
       case 1:
        dots_color[((Buffprt+1)&1)][color][ow][]= (veeaddressX0>>4)&0x0F;

dots_color[((Buffprt+1)&1)][color][ow][1] =((veeaddressX0<<4)&0xF0 | (veeaddr
essX1>>4)&0x0F);

dots_color[((Buffprt+1)&1)][color][ow][2] =((veeaddressX1<<4)&0xF0 | (veeaddr
essX2>>4)&0x0F);
        break;
       case 2:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =veeaddressX0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =veeaddressX1;
        break;
       case 3:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =(veeaddressX0>>4)&0x0F;

dots_color[((Buffprt+1)&1)][color][ow][2] =((veeaddressX0<<4)&0xF0 | (veeaddr
essX1>>4)&0x0F);
        break;
       case 4:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =veeaddressX0;
        break;
       case 5:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       //negatives
       case -1:

dots_color[((Buffprt+1)&1)][color][ow][]= ((veeaddressX0<<4)&0xF0 | (veeaddre
ssX1>>4)&0x0F);

dots_color[((Buffprt+1)&1)][color][ow][1] =((veeaddressX1<<4)&0xF0 | (veeaddr
essX2>>4)&0x0F);
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       case -2:
        dots_color[((Buffprt+1)&1)][color][ow][]= veeaddressX1;
        dots_color[((Buffprt+1)&1)][color][ow][1] =veeaddressX2&0xF0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       case -3:

dots_color[((Buffprt+1)&1)][color][ow][]=  ((veeaddressX1<<4)&0xF0 | (veeaddr
```

```
essX2>>4)&0x0F);
        dots_color[((Buffprt+1)&1)][color][ow][1] = 0; //because we have 5 
leds per row
        dots_color[((Buffprt+1)&1)][color][ow][2] = 0;
        break;
       case -4:
        dots_color[((Buffprt+1)&1)][color][ow][]= veeaddressX2&0xF0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       case -5:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
      }//end of switch
   }// end of row loop
  }// end of color loop

  if (RainbowCMD[7]>0x00){
   delayMicroseconds(RainbowCMD[7]*600);
   delayMicroseconds(delayVar[ainbowCMD[7]-1]+50);
  }
  delayMicroseconds(130);

  Buffprt++;
  Buffprt&=1;

}

//display image and audio reading from RAM
void DispshowPictureWithAudio(void)
{
  unsigned char pi, audioChannels;
  signed char shifts;
  unsigned char color=0,row=0,dots=0;
  //unsigned char tempDelay;
  unsigned char fir,sec;
  unsigned char veeaddressX0, veeaddressX1, veeaddressX2;
  unsigned long time;

  shifts=((RainbowCMD[2]>>4)&0x0F);   shifts-=5;
  pi=RainbowCMD[4];
  audioChannels=RainbowCMD[5];
  RainbowCMD[1]=0;

  for(color=0;color<3;color++)
  {
   for (row=0;row<5;row++)
   {

      //shift doesnt work now (maybe in the future if we want to add it)
      //eeaddressX=((15*color)+(3*row)+(dots))+(pi*128);  //15 bytes per 
color, 3 byte per row.  Pages of 128

      veeaddressX0=preLoaded[pi][color][ow][];  //15 bytes per color, 3 byte 
per row.  Pages of 128
```

```
      veeaddressX1=preLoaded[pi][color][ow][1];  //15 bytes per color, 3 byte
per row.  Pages of 128
      veeaddressX2=preLoaded[pi][color][ow][2];  //15 bytes per color, 3 byte
per row.  Pages of 128

      switch(shifts){
       case 0:
        dots_color[((Buffprt+1)&1)][color][ow][]=  veeaddressX0;
        dots_color[((Buffprt+1)&1)][color][ow][1] = veeaddressX1;
        dots_color[((Buffprt+1)&1)][color][ow][2] = veeaddressX2;
        break;
       case 1:
        dots_color[((Buffprt+1)&1)][color][ow][]= (veeaddressX0>>4)&0x0F;

dots_color[((Buffprt+1)&1)][color][ow][1] =((veeaddressX0<<4)&0xF0 | (veeaddr
essX1>>4)&0x0F);

dots_color[((Buffprt+1)&1)][color][ow][2] =((veeaddressX1<<4)&0xF0 | (veeaddr
essX2>>4)&0x0F);
        break;
       case 2:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =veeaddressX0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =veeaddressX1;
        break;
       case 3:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =(veeaddressX0>>4)&0x0F;

dots_color[((Buffprt+1)&1)][color][ow][2] =((veeaddressX0<<4)&0xF0 | (veeaddr
essX1>>4)&0x0F);
        break;
       case 4:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =veeaddressX0;
        break;
       case 5:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       //negatives
       case -1:

dots_color[((Buffprt+1)&1)][color][ow][]= ((veeaddressX0<<4)&0xF0 | (veeaddre
ssX1>>4)&0x0F);

dots_color[((Buffprt+1)&1)][color][ow][1] =((veeaddressX1<<4)&0xF0 | (veeaddr
essX2>>4)&0x0F);
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       case -2:
        dots_color[((Buffprt+1)&1)][color][ow][]= veeaddressX1;
        dots_color[((Buffprt+1)&1)][color][ow][1] =veeaddressX2&0xF0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
```

```
      case -3:

dots_color[((Buffprt+1)&1)][color][ow][]=  ((veeaddressX1<<4)&0xF0 | (veeaddr
essX2>>4)&0x0F);
        dots_color[((Buffprt+1)&1)][color][ow][1] = 0; //because we have 5
leds per row
        dots_color[((Buffprt+1)&1)][color][ow][2] = 0;
        break;
      case -4:
        dots_color[((Buffprt+1)&1)][color][ow][]= veeaddressX2&0xF0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
      case -5:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
      }//end of switch
   }// end of row loop
  }// end of color loop


  if (RainbowCMD[7]>0x00){
    delayMicroseconds(RainbowCMD[7]*600);
    delayMicroseconds(delayVar[ainbowCMD[7]-1]);
  }
  delayMicroseconds(350);

  Buffprt++;
  Buffprt&=1;

  //audio switch code
  switch(audioChannels){
    case 0:  digitalWrite(11, LOW);    digitalWrite(12, LOW);
digitalWrite(13, LOW);  break;
    case 1:  digitalWrite(11, HIGH);   digitalWrite(12, LOW);
digitalWrite(13, LOW);  break;
    case 2:  digitalWrite(11, LOW);    digitalWrite(12, HIGH);
digitalWrite(13, LOW);  break;
    case 3:  digitalWrite(11, HIGH);   digitalWrite(12, HIGH);
digitalWrite(13, LOW);  break;
    case 4:  digitalWrite(11, LOW);    digitalWrite(12, LOW);
digitalWrite(13, HIGH);  break;
    case 5:  digitalWrite(11, HIGH);   digitalWrite(12, LOW);
digitalWrite(13, HIGH);  break;
    case 6:  digitalWrite(11, LOW);    digitalWrite(12, HIGH);
digitalWrite(13, HIGH);  break;
    case 7:  digitalWrite(11, HIGH);   digitalWrite(12, HIGH);
digitalWrite(13, HIGH);  break;
  }
}

//display image reading from RAM
void DispshowPicture5(void)
{
  unsigned char pi;
```

```
  signed char shifts;
  unsigned char color=0,row=0,dots=0;
  //unsigned char tempDelay;
  unsigned char fir,sec;
  unsigned char veeaddressX0, veeaddressX1, veeaddressX2;
  unsigned long time;

  shifts=((RainbowCMD[2]>>4)&0x0F);   shifts-=5;
  pi=RainbowCMD[4];
  RainbowCMD[1]=0;

  for(color=0;color<3;color++)
  {
   for (row=0;row<5;row++)
   {

      //shift doesnt work now (maybe in the future if we want to add it)
      //eeaddressX=((15*color)+(3*row)+(dots))+(pi*128);  //15 bytes per
color, 3 byte per row.  Pages of 128

      veeaddressX0=preLoaded[pi][color][ow][];  //15 bytes per color, 3 byte
per row.  Pages of 128
      veeaddressX1=preLoaded[pi][color][ow][1];  //15 bytes per color, 3 byte
per row.  Pages of 128
      veeaddressX2=preLoaded[pi][color][ow][2];  //15 bytes per color, 3 byte
per row.  Pages of 128

      switch(shifts){
       case 0:
        dots_color[((Buffprt+1)&1)][color][ow][]=  veeaddressX0;
        dots_color[((Buffprt+1)&1)][color][ow][1] = veeaddressX1;
        dots_color[((Buffprt+1)&1)][color][ow][2] = veeaddressX2;
        break;
       case 1:
        dots_color[((Buffprt+1)&1)][color][ow][]= (veeaddressX0>>4)&0x0F;

dots_color[((Buffprt+1)&1)][color][ow][1] =((veeaddressX0<<4)&0xF0 | (veeaddr
essX1>>4)&0x0F);

dots_color[((Buffprt+1)&1)][color][ow][2] =((veeaddressX1<<4)&0xF0 | (veeaddr
essX2>>4)&0x0F);
        break;
       case 2:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =veeaddressX0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =veeaddressX1;
        break;
       case 3:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =(veeaddressX0>>4)&0x0F;

dots_color[((Buffprt+1)&1)][color][ow][2] =((veeaddressX0<<4)&0xF0 | (veeaddr
essX1>>4)&0x0F);
        break;
       case 4:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
```

```
        dots_color[((Buffprt+1)&1)][color][ow][2] =veeaddressX0;
        break;
       case 5:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       //negatives
       case -1:

dots_color[((Buffprt+1)&1)][color][ow][]= ((veeaddressX0<<4)&0xF0 | (veeaddre
ssX1>>4)&0x0F);

dots_color[((Buffprt+1)&1)][color][ow][1] =((veeaddressX1<<4)&0xF0 | (veeaddr
essX2>>4)&0x0F);
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       case -2:
        dots_color[((Buffprt+1)&1)][color][ow][]= veeaddressX1;
        dots_color[((Buffprt+1)&1)][color][ow][1] =veeaddressX2&0xF0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       case -3:

dots_color[((Buffprt+1)&1)][color][ow][]=  ((veeaddressX1<<4)&0xF0 | (veeaddr
essX2>>4)&0x0F);
        dots_color[((Buffprt+1)&1)][color][ow][1] = 0; //because we have 5
leds per row
        dots_color[((Buffprt+1)&1)][color][ow][2] = 0;
        break;
       case -4:
        dots_color[((Buffprt+1)&1)][color][ow][]= veeaddressX2&0xF0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
       case -5:
        dots_color[((Buffprt+1)&1)][color][ow][]=0;
        dots_color[((Buffprt+1)&1)][color][ow][1] =0;
        dots_color[((Buffprt+1)&1)][color][ow][2] =0;
        break;
      }//end of switch
   }// end of row loop
  }// end of color loop


  if (RainbowCMD[7]>0x00){
    delayMicroseconds(RainbowCMD[7]*600);
    delayMicroseconds(delayVar[ainbowCMD[7]-1]);
  }
  delayMicroseconds(360);

  Buffprt++;
  Buffprt&=1;

}

//displays a single pixel in the cell
```

```
void setOnePixel(void)
{
  unsigned char color=0,row=0,dots=0;
  unsigned char r, g, b, rowX, colX;
  r=RainbowCMD[2]&0x0F;
  g=(RainbowCMD[3]>>4)&0x0F;
  b=RainbowCMD[3]&0x0F;
  rowX=(RainbowCMD[5]>>4)&0x0F;
  colX=RainbowCMD[5]&0x0F;
  RainbowCMD[1]=0;

  for(color=0;color<3;color++)
  {
   for (row=0;row<5;row++)
   {
    for (dots=0;dots<3;dots++)
    {
      unsigned char data;
      switch (color){
        case 0:      data=g;  break;
        case 1:      data=r;  break;
        case 2:      data=b;  break;
        default:     break;
      }
      if (row==rowX && dots==(int(colX/2))){
        if(colX%2==0)  dots_color[((Buffprt+1)&1)][color][ow][dots]=
(data<<4)&0xF0;
        else        dots_color[((Buffprt+1)&1)][color][ow][dots]= data&0x0F;
      }
      else dots_color[((Buffprt+1)&1)][color][ow][dots]= 0x00;
    }
   }
  }
  if (RainbowCMD[7]>0x00){
    delayMicroseconds(RainbowCMD[7]*600);
    delayMicroseconds(delayVar[ainbowCMD[7]-1]+50);
  }
  delayMicroseconds(500);
  Buffprt++;
  Buffprt&=1;

}

//displays a single pixel in the cell plus audio
void setOnePixelWithAudio(void)
{
  unsigned char color=0,row=0,dots=0;
  unsigned char r, g, b, rowX, colX, audioChannels;
  r=RainbowCMD[2]&0x0F;
  g=(RainbowCMD[3]>>4)&0x0F;
  b=RainbowCMD[3]&0x0F;
  audioChannels=RainbowCMD[4];
  rowX=(RainbowCMD[5]>>4)&0x0F;
  colX=RainbowCMD[5]&0x0F;
  RainbowCMD[1]=0;

  for(color=0;color<3;color++)
```

```
  {
   for (row=0;row<5;row++)
   {
    for (dots=0;dots<3;dots++)
    {
      unsigned char data;
      switch (color){
        case 0:     data=g;  break;
        case 1:     data=r;  break;
        case 2:     data=b;  break;
        default:    break;
      }
      if (row==rowX && dots==(int(colX/2))){
        if(colX%2==0)  dots_color[((Buffprt+1)&1)][color][ow][dots]=
(data<<4)&0xF0;
        else        dots_color[((Buffprt+1)&1)][color][ow][dots]= data&0x0F;
      }
      else dots_color[((Buffprt+1)&1)][color][ow][dots]= 0x00;
    }
   }
  }
  if (RainbowCMD[7]>0x00){
    delayMicroseconds(RainbowCMD[7]*600);
    delayMicroseconds(delayVar[ainbowCMD[7]-1]);
  }
  delayMicroseconds(420);

  Buffprt++;
  Buffprt&=1;
  //audio switch code
  switch(audioChannels){
    case 0:  digitalWrite(11, LOW);    digitalWrite(12, LOW);
digitalWrite(13, LOW);  break;
    case 1:  digitalWrite(11, HIGH);   digitalWrite(12, LOW);
digitalWrite(13, LOW);  break;
    case 2:  digitalWrite(11, LOW);    digitalWrite(12, HIGH);
digitalWrite(13, LOW);  break;
    case 3:  digitalWrite(11, HIGH);   digitalWrite(12, HIGH);
digitalWrite(13, LOW);  break;
    case 4:  digitalWrite(11, LOW);    digitalWrite(12, LOW);
digitalWrite(13, HIGH);  break;
    case 5:  digitalWrite(11, HIGH);   digitalWrite(12, LOW);
digitalWrite(13, HIGH);  break;
    case 6:  digitalWrite(11, LOW);    digitalWrite(12, HIGH);
digitalWrite(13, HIGH);  break;
    case 7:  digitalWrite(11, HIGH);   digitalWrite(12, HIGH);
digitalWrite(13, HIGH);  break;
  }
}

//stores a pixel in EEPROM memory
void setPixel(){
   unsigned char color=0;
   unsigned char r, g, b, pi, row, col;
   unsigned int eeaddressX;
   r=RainbowCMD[2]&0x0F;
   g=(RainbowCMD[3]>>4)&0x0F;
```

```
   b=RainbowCMD[3]&0x0F;
   pi=RainbowCMD[4];
   row=(RainbowCMD[5]>>4)&0x0F;
   col=RainbowCMD[5]&0x0F;

   byte aux;
   //green, blue, red
   for(color=0;color<3;color++){
    eeaddressX=((row*3)+((int)col/2))+(pi*128)+(color*15);
    aux=i2c_eeprom_read_byte(EEPROM_ADDR, eeaddressX );

    byte data;
    switch (color){
       case 0:      data=g;  break;
       case 1:      data=r;  break;
       case 2:      data=b;  break;
       default:     break;
    }
    if(col%2==0)  i2c_eeprom_write_byte(EEPROM_ADDR, eeaddressX,
(data<<4)|(aux&0x0F));
    else        i2c_eeprom_write_byte(EEPROM_ADDR, eeaddressX,
(aux&0xF0)|data);
    delay(5);
   }
}

void getPixel(){
  //do nothing for now
}

//set the audio channels to open or closed
void setAudio(){
   unsigned char audioChannels;
   audioChannels=RainbowCMD[4]&0x0F;
   //audio switch code
   if (RainbowCMD[7]>0x00){
    delayMicroseconds(RainbowCMD[7]*600);
    delayMicroseconds(delayVar[ainbowCMD[7]-1]-50);
   }
   delayMicroseconds(600);
   switch(audioChannels){
    case 0:  digitalWrite(11, LOW);    digitalWrite(12, LOW);
digitalWrite(13, LOW);  break;
    case 1:  digitalWrite(11, HIGH);   digitalWrite(12, LOW);
digitalWrite(13, LOW);  break;
    case 2:  digitalWrite(11, LOW);    digitalWrite(12, HIGH);
digitalWrite(13, LOW);  break;
    case 3:  digitalWrite(11, HIGH);   digitalWrite(12, HIGH);
digitalWrite(13, LOW);  break;
    case 4:  digitalWrite(11, LOW);    digitalWrite(12, LOW);
digitalWrite(13, HIGH);  break;
    case 5:  digitalWrite(11, HIGH);   digitalWrite(12, LOW);
digitalWrite(13, HIGH);  break;
    case 6:  digitalWrite(11, LOW);    digitalWrite(12, HIGH);
digitalWrite(13, HIGH);  break;
    case 7:  digitalWrite(11, HIGH);   digitalWrite(12, HIGH);
digitalWrite(13, HIGH);  break;
```

```
    }
}

//this version uses the same functionality thant setAudio()
void clearAudio(){
   unsigned char audioChannels;
   audioChannels=RainbowCMD[4]&0x0F;
   //clear pin number
   if (RainbowCMD[7]>0x00){
    delayMicroseconds(RainbowCMD[7]*600);
    delayMicroseconds(delayVar[ainbowCMD[7]-1]-50);
   }
   delayMicroseconds(600);
   switch(audioChannels){
      case 0:  digitalWrite(11, LOW);    digitalWrite(12, LOW);
digitalWrite(13, LOW);  break;
      case 1:  digitalWrite(11, HIGH);   digitalWrite(12, LOW);
digitalWrite(13, LOW);  break;
      case 2:  digitalWrite(11, LOW);    digitalWrite(12, HIGH);
digitalWrite(13, LOW);  break;
      case 3:  digitalWrite(11, HIGH);   digitalWrite(12, HIGH);
digitalWrite(13, LOW);  break;
      case 4:  digitalWrite(11, LOW);    digitalWrite(12, LOW);
digitalWrite(13, HIGH);  break;
      case 5:  digitalWrite(11, HIGH);   digitalWrite(12, LOW);
digitalWrite(13, HIGH);  break;
      case 6:  digitalWrite(11, LOW);    digitalWrite(12, HIGH);
digitalWrite(13, HIGH);  break;
      case 7:  digitalWrite(11, HIGH);   digitalWrite(12, HIGH);
digitalWrite(13, HIGH);  break;
   }
}

//process the command
void GetCMD(void)
{
  switch(RainbowCMD[1])
        {
          case loadPreloadedCode:
          loadPreloaded();
          break;

        case showPrefabnicate5:
          DispshowPicture5();
          break;

        case showPrefabnicate2:
          DispshowPicture2();
          break;

        case showPrefabnicate3:
          DispshowPictureWithAudio();
          break;

        case setJustOnePixel:
          setOnePixel();
          break;
```

```
        case clearDisplayCode:
          clearDisplay();
          break;

        case setJustOnePixelWithAudio:
          setOnePixelWithAudio();
          break;

        case setAudioChannel:
          setAudio();
          break;

        case clearAudioChannel:
          clearAudio();
          break;

        case setPixelInImage:
        showItsReceivingON();
          setPixel();
        clearDisplay();
          break;

        case getPixelInImage:  //this one not yet implemented
          getPixel();
          break;

        case showChar:
          DispshowChar();
          break;

          case showColor:
          DispshowColor();
          break;
        }
}

//procedure to show a pixel when the cell is receiving data
void showItsReceivingON(){
  //one pixel to know its processing
  unsigned char color=0,row=0,dots=0;
  unsigned char r, g, b, rowX, colX;
  r=0;
  g=15;
  b=0;
  rowX=2;
  colX=2;

  for(color=0;color<3;color++){
   for (row=0;row<5;row++){
    for (dots=0;dots<3;dots++){
      unsigned char data;
      switch (color){
        case 0:      data=g;  break;
        case 1:      data=r;  break;
        case 2:      data=b;  break;
        default:     break;
```

```
      }
      if (row==rowX && dots==(int(colX/2))){
        if(colX%2==0)  dots_color[((Buffprt+1)&1)][color][ow][dots]=
(data<<4)&0xF0;
        else        dots_color[((Buffprt+1)&1)][color][ow][dots]= data&0x0F;
      }
      else dots_color[((Buffprt+1)&1)][color][ow][dots]= 0x00;
    }
   }
  }
  Buffprt++;
  Buffprt&=1;
}

//clear the Display and shows a blank array
void clearDisplay(){
  //one pixel to know its processing
  unsigned char color=0,row=0,dots=0;

  for(color=0;color<3;color++){
   for (row=0;row<5;row++){
    for (dots=0;dots<3;dots++){
      dots_color[((Buffprt+1)&1)][color][ow][dots]= 0x00;
    }
   }
  }
  if (RainbowCMD[7]>0x00){
   delayMicroseconds(RainbowCMD[7]*600);
   delayMicroseconds(delayVar[ainbowCMD[7]-1]);
  }
  delayMicroseconds(600);
  Buffprt++;
  Buffprt&=1;
}

//loads the EEPROM symbol into RAM
void loadPreloaded(){
  unsigned char pi, piDestiny;
  unsigned char color=0,row=0,dots=0;
  //unsigned int veeaddressX0, veeaddressX1, veeaddressX2,
  int eeaddressX0;
  char value_eeaddressX_h1[20];
  char value_eeaddressX_h2[25];
  char value_eeaddressX[45];  //45+null character added in strcat-46

  pi=RainbowCMD[4];
  piDestiny=RainbowCMD[5];
  RainbowCMD[1]=0;
  eeaddressX0=(pi*128);

  //read the eproom

i2c_eeprom_read_bufferChar(EEPROM_ADDR, eeaddressX0, value_eeaddressX_h1, 20)
;
  for (int i=0; i<20;i++){  value_eeaddressX[i]=value_eeaddressX_h1[i];    }

i2c_eeprom_read_bufferChar(EEPROM_ADDR, eeaddressX0+20, value_eeaddressX_h2,
```

```
25);
  for (int i=20; i<45;i++){ value_eeaddressX[i]=value_eeaddressX_h2[i-20];  }

  for(color=0;color<3;color++)
  {
   for (row=0;row<5;row++)
   {
      /*preLoaded[piDestiny][color][ow][]=i2c_eeprom_read_byte( EEPROM_ADDR,
((15*color)+(3*row)+(0))+(pi*128));
      preLoaded[piDestiny][color][ow][1]=i2c_eeprom_read_byte( EEPROM_ADDR, (
(15*color)+(3*row)+(1))+(pi*128));
      preLoaded[piDestiny][color][ow][2]=i2c_eeprom_read_byte( EEPROM_ADDR, (
(15*color)+(3*row)+(2))+(pi*128));*/

      preLoaded[piDestiny][color][ow][]=value_eeaddressX[((15*color)+(3*row)+
(0))];  //15 bytes per color, 3 byte per row.  Pages of 128
      preLoaded[piDestiny][color][ow][1]=value_eeaddressX[((15*color)+(3*row)
+(1))];  //15 bytes per color, 3 byte per row.  Pages of 128
      preLoaded[piDestiny][color][ow][2]=value_eeaddressX[((15*color)+(3*row)
+(2))];  //15 bytes per color, 3 byte per row.  Pages of 128

   }// end of row loop
  }// end of color loop

}

//=================================================================
void shift_1_bit(unsigned char LS)  //shift 1 bit of  1 Byte color data into
Shift register by clock
{
  if(LS){   shift_data_1;  }
  else  {   shift_data_0;  }
  clk_rising;
}
//=================================================================
void flash_next_line(unsigned char line,unsigned char level) // scan one line
{
  disable_oe;
  close_all_line;
  open_line(line);
  shift_24_bit(line,level);
  enable_oe;
}

//=================================================================
void shift_24_bit(unsigned char line,unsigned char level)  // display one
line by the color level in buff
{
  unsigned char color=0,row=0;
  unsigned char data0=0,data1=0;
  le_high;
  for(color=0;color<3;color++)//GBR
  {
   for(row=0;row<4;row++)
   {
    if(row<2){
    data1=dots_color[Buffprt][color][line][ow]&0x0f;
```

```
    data0=dots_color[Buffprt][color][line][ow]>>4;
    }
    else if(row==2){
    data1=0;
    data0=dots_color[Buffprt][color][line][ow]>>4;
    }
    else{
    data1=0;data0=0;
    }

    if(data0>level)  //gray scale,0x0f aways light
    {
      shift_1_bit(1);
    }
    else
    {
      shift_1_bit(0);
    }

    if(data1>level)
    {
      shift_1_bit(1);
    }
    else
    {
      shift_1_bit(0);
    }
   }
  }
  le_low;
}

//=================================================================
void open_line(unsigned char line)    // open the scaning line
{
  switch(line)
  {
  case 0:{    open_line0;    break;    }
  case 1:{    open_line1;    break;    }
  case 2:{    open_line2;    break;    }
  case 3:{    open_line3;    break;    }
  case 4:{    open_line4;    break;    }
  case 5:{    open_line5;    break;    }
  case 6:{    open_line6;    break;    }
  case 7:{    open_line7;    break;    }
  }
}


//   External EEPROM - Wathc out this one does work for Microchip EEPROMs
//

void i2c_eeprom_write_byte( int deviceaddress, unsigned int eeaddress, byte
data ) {
  int rdata = data;
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));   // Address High Byte
```

```
  Wire.send((int)(eeaddress & 0xFF));  // Address Low Byte
  Wire.send(rdata);
  Wire.endTransmission();
}

// Address is a page address, 6-bit (63).  More and end will wrap around
// But data can be maximum of 28 bytes, because the Wire library has a buffer
 of 32 bytes
void i2c_eeprom_write_page( int deviceaddress, unsigned int eeaddresspage,
byte* data, byte length ) {
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddresspage >> 8)); // Address High Byte
  Wire.send((int)(eeaddresspage & 0xFF)); // Address Low Byte
  byte c;
  for ( c = 0; c < length; c++)
   Wire.send(data[c]);
  Wire.endTransmission();
  delay(10);                    // need some delay
}

byte i2c_eeprom_read_byte( int deviceaddress, unsigned int eeaddress ) {
  byte rdata = 0xFF;
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));   // Address High Byte
  Wire.send((int)(eeaddress & 0xFF));  // Address Low Byte
  Wire.endTransmission();
  Wire.requestFrom(deviceaddress,1);
  if (Wire.available()) rdata = Wire.receive();
  return rdata;
}

// should not read more than 28 bytes at a time!
void i2c_eeprom_read_buffer( int deviceaddress, unsigned int eeaddress, byte
*buffer, int length ) {
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));   // Address High Byte
  Wire.send((int)(eeaddress & 0xFF));  // Address Low Byte
  Wire.endTransmission();
  Wire.requestFrom(deviceaddress,length);
  //int c = 0;
  for ( int c = 0; c < length; c++ )
   if (Wire.available()) buffer[c] = Wire.receive();
}

 // should not read more than 28 bytes at a time!
void i2c_eeprom_read_bufferChar( int deviceaddress, unsigned int eeaddress,
char *buffer, int length ) {
  Wire.beginTransmission(deviceaddress);
  Wire.send((int)(eeaddress >> 8));   // Address High Byte
  Wire.send((int)(eeaddress & 0xFF));  // Address Low Byte
  Wire.endTransmission();
  Wire.requestFrom(deviceaddress,length);
  //int c = 0;
  for ( int c = 0; c < length; c++ )
   if (Wire.available()) buffer[c] = Wire.receive();
}
```

```
/*
Notes:

  // TESTS FOR EACH FUNCTION BEGIN HERE
  Serial.println("Writing Test:");
  for (int i=0; i<32; i++){        // loop for first 20 slots
   i2c_eeprom_write_byte(EEPROM_ADDR,i,77-i);
// write address + 65 A or 97 a  77-i
   Serial.print(".  ");
   delay(5);                  // NEED THIS DELAY! (tests suggest it can be as
                           // small as delay(3) -- BBR
  }

  Serial.println("");
  //delay(500);
  Serial.println("Reading Test:");
  for (int i=0; i<45; i++){        // loop for first 20 slots
   Serial.print(i2c_eeprom_read_byte(EEPROM_ADDR, i),BYTE);
   Serial.print(" ");
  }
  */
```