MOUSE PAINTER

PRO

TEAM MEMBERS

RUDRA PRATAP SUMAN, SHUBHAM GUPTA, AVINASH SHRIVASTAV

MENTOR

RISHABH MAHESHWARI

CONTENTS

INTRODUCTION

Our project consists of implementing MS paint software on graphical LCD by taking input from a PS2 mouse.Mouse is used to control the cursor on screen and choosing the desired tools on the paint window.It contains tools like:

1.Pen tool

2.Multi size eraser

3.Shapes(rectangle,circle,line)

4.Crop fumction

5.Spray paint

6.Fill function

This device also stores the last drawn image in it.When the power is switched off the drawing remains saved and is retained on switching on the power.

Also there is a reset option to clear the whole screen so as to start a new pic.

HARDWARE CONFIGURATIONS

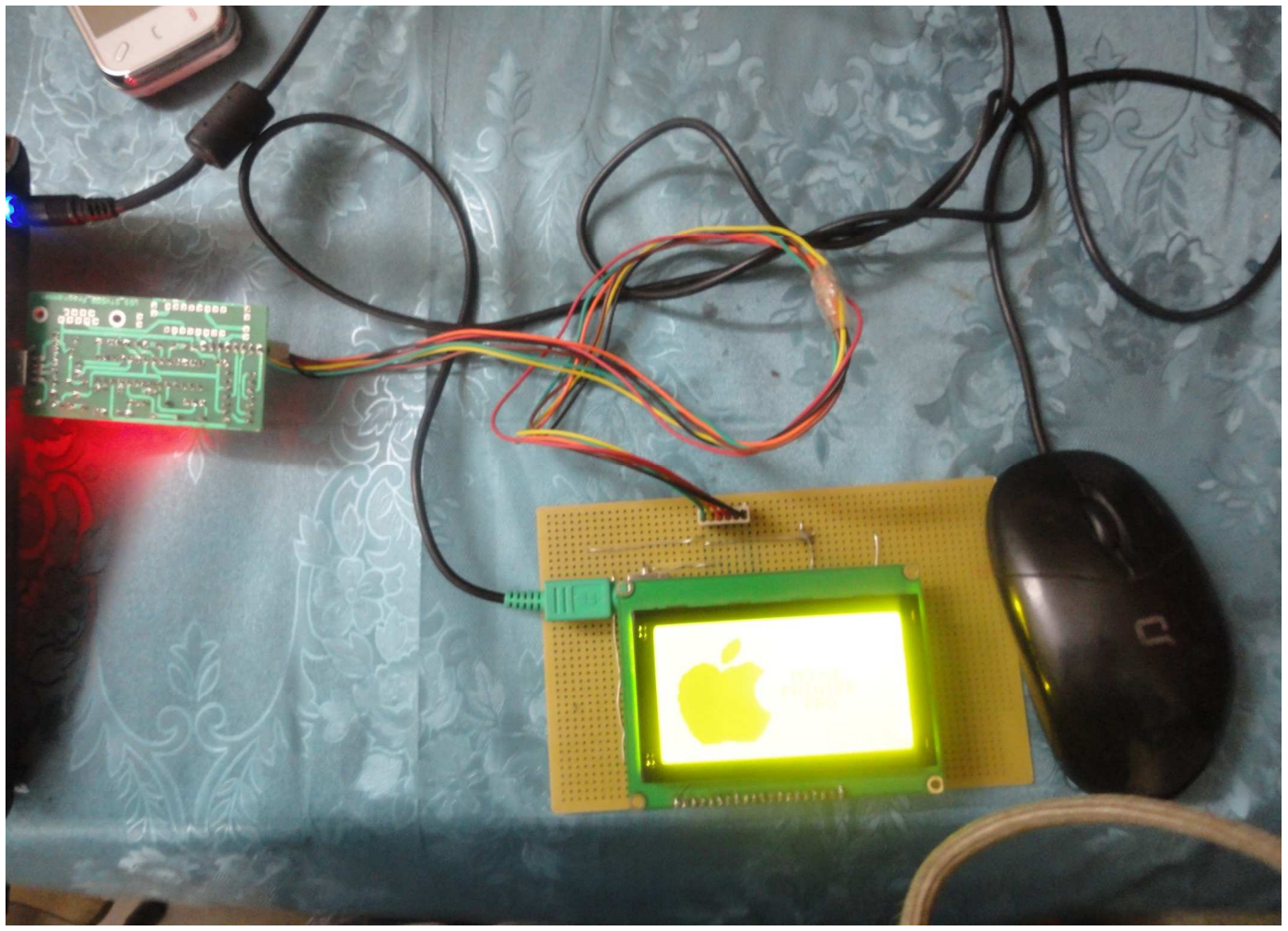
1.Atmega 32

2. 128x64 Monochromatic Graphical LCD based on ks0108 controller
3.Potentiometer

4.Push button

5.PS 2 mouse

HARDWARE



HARDWARE DETAILS:

1.**Graphical LCD**: For this project we are using a 128x64 Monochromatic Graphical LCD based on ks0108 controller.

Each pixel of this graphical LCD is accessible through various functions available in the header files.It also provides various functions to write characters and draw different shapes.

2.**PS2 Mouse**:This is used to control the paint software on glcd.

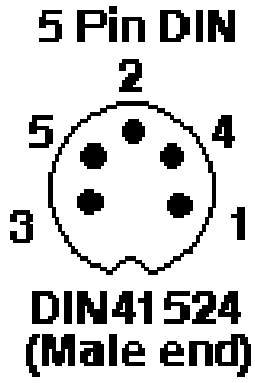
The physical PS/2 port is one of two styles of connectors: The 5-pin DIN or the 6-pin mini-DIN. Both connectors are completely (electrically) similar; the only practical difference between the two is the arrangement of pins.

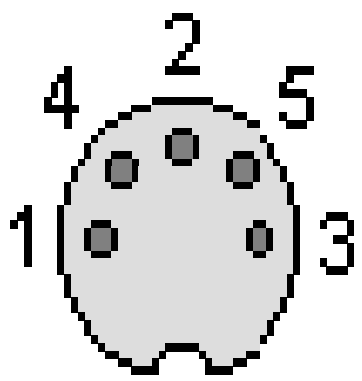
CONNECTOR:

The pinouts for each connector are shown below:

5-PIN DIN(AT/XT)

Male Female





1 - Clock

2 - Data

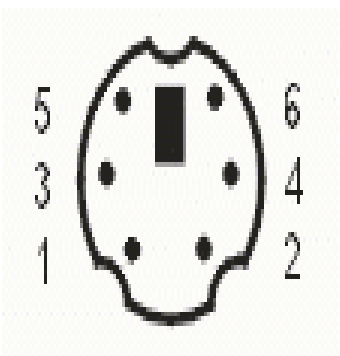
3 - Not Implemented

4 - Ground

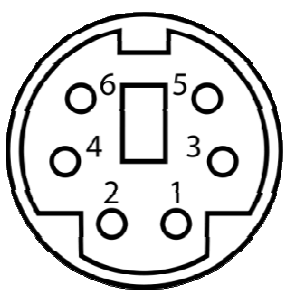
5 - Vcc (+5V)

6-pin Mini-DIN (PS/2) :

Male



Female



DETAILS:

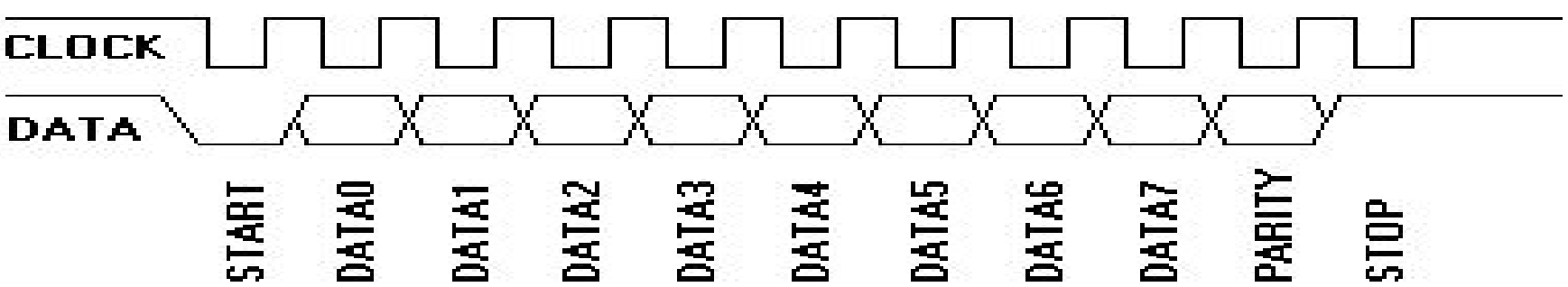
1 - Data
2 - Not Implemented
3 - Ground
4 - Vcc (+5V)
5 - Clock
6 - Not Implemented

IMPLEMENTATION

PS2 MOUSE

Algorithm : Mouse sends a data line and a clock line to the atmega.The data is read synchronous to the clock.Mouse keeps on updating its counters for state of click and postion . As soon as it receives a data reporting command from the MCU it sends a data packet to it with all the information about its state.This is is the remote mode of operation.Data sent by the mouse is read by the atmega at each falling edge of clock aftera start bit is received.Data byte generally consists of 11 bits:

1 start bit,8 data bits,1 parity bit and a stop bit.



Atmega to mouse communication:

There are specific timing guidelines that need to be followed

while communicating with the PS-2 mouse. The process and timing for sending a byte of

data to the mouse is outlined below:

1) Bring the Clock line low for at least 100 microseconds.

2) Bring the Data line low.

3) Release the Clock line.

4) Wait for the device to bring the Clock line low.

5) Set/reset the Data line to send the first data bit

6) Wait for the device to bring Clock high.

7) Wait for the device to bring Clock low.

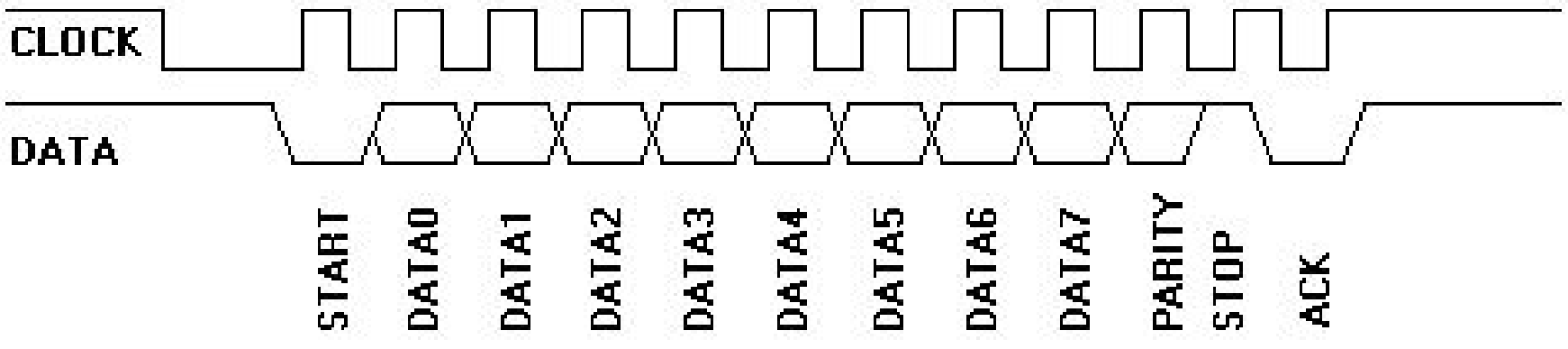
8) Repeat steps 5-7 for other seven data bits and the parity bit

9) Release the Data line.

10) Wait for the device to bring Data low.

11) Wait for the device to bring Clock low.

12) Wait for the device to release Data and Clock



After successful transmission of command from atmega to mouse an acknowledgment bit(ACK) is sent by the mouse.

DATA BYTES :

On mouse moves the mouse sends 3 bytes of information through the ps-2 port. The contents of the data packet is described below in the diagram:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X movement							
Byte 3	Y movement							

Byte 1 contains information on the direction the mouse moved wrt its previous position,status of the mouse buttons (0 for not pressed and 1 for pressed) and the x and y overflows which are set if the counters go past 255. Byte 2 and 3 contain information

regarding the x movement and the y movement. The x movement and y movement are calculated based on the counter values in the mouse. The counters get incremented by a value of 4 for a movement of 1mm on the screen.The counters are 9-bit 2's complement values with the most significant bit is the sign bit i.e. the 5^{th} and 6^{th} bit of first byte. Rest of the 8 bits constitute the 2^{nd} byte(for x movement) and 3^{rd} byte (for y movement).

Mouse which are not plug and play and require initialisation at the time of boot.The initialisation process is:

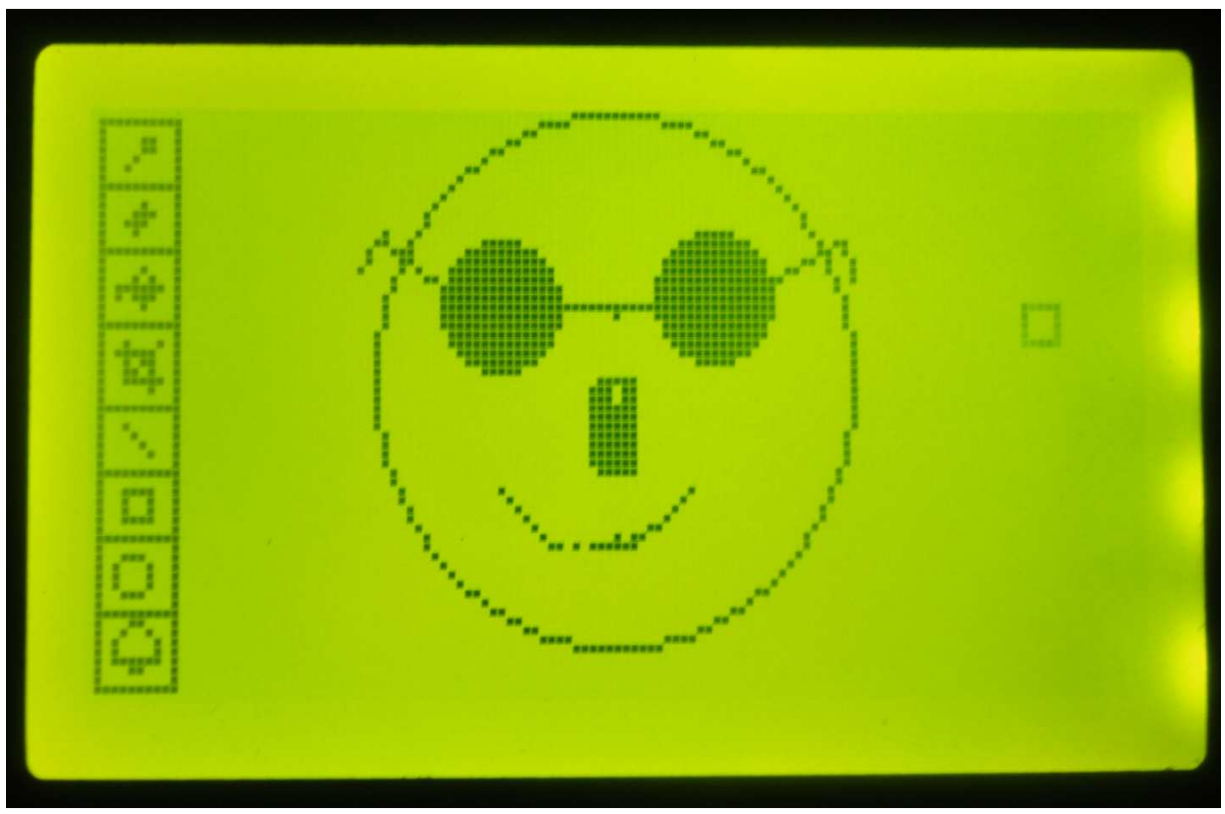
Power-on Reset:
Mouse: AA Self-test passed
Mouse: 00 Mouse ID
Host: FF Reset command
Mouse: FA Acknowledge
Mouse: AA Self-test passed
Mouse: 00 Mouse ID
Host: FF Reset command
Mouse: FA Acknowledge
Mouse: AA Self-test passed
Mouse: 00 Mouse ID

Host: FF Reset command
Mouse: FA Acknowledge
Mouse: AA Self-test passed
Mouse: 00 Mouse ID
Host: F2 Read Device Type
Mouse: FA Acknowledge
Mouse: 00 Mouse ID
Host: E8 Set resolution
Mouse: FA Acknowledge
Host: 03 8 Counts/mm
Mouse: FA Acknowledge
Host: E6 Set Scaling 1:1
Mouse: FA Acknowledge
Host: F3 Set Sample Rate
Mouse: FA Acknowledge
Host: 28 decimal 40
Mouse: FA Acknowledge
Host: F4 Enable
Mouse: FA Acknowledge
Initialization complete...
But most of the mouses are plug and play and do not require initialisation.

The data bytes sent by the mouse are read and decoded appropriately according to our need.

GLCD Interfacing

We needed to interface an GLCD to our microcontroller so that we can display mouse painter pro window on.we used a 128x64GLCD.so we had 128X64 pixels availbale and we can each one of them by simple functions.



Circuit Connection

There are 20 pins in an GLCD; See reverse side of the GLCD for the PIN configuration.
The connections have to be made as given below:

GLCD connections for PORT B AND C

Setting up in Microcontroller

When we connect an GLCD to Atmega32, two full PORT is dedicated to it. To enable GLCD interfacing in the microcontroller, just click on the GLCD tab in the Avr Wizard and select the PORT at which you want to connect the GLCD. We selected PORTA and PORTC. Avr Wizard now shows you the complete list of connections which you will have to make in order to interface the GLCD.

GLCD settings on AVR wizard window.
As you can see, there are some special connections other than those to uC, Vcc and gnd. These are general GLCD settings. Pin 17 and PIN 3 is for the GLCD contrast, ground it through a <10kΩ resistance/ potentiometer for optimum contrast. Pin 19 & Pin 20 are for GLCD backlight, give them permanent +5V and GND respectively as we need to glow it continuously.

Printing Functions

Now once the connections have been made, we are ready to display something on our screen. Displaying our name would be great to start with. Some of the general GLCD functions which you must know are:

g lcd_clear()

Clears the glcd. Remember! Call this function before the while(1) loop, otherwise you won't be able to see anything!

glcdSetAddress(x,y)

Place the cursor at coordinates (x,y) and start writing from there. The first coordinate is (0,0). Hence, x ranges from 0 to 127 and y from 0 to 7 in our GLCD. Suppose you want to display something starting from the 5th character in second line, then the function would be
glcdSetAddress(5,1);

glcdWriteChar(char c)

To display a single character. E.g.,
glcdWriteChar ('c');

glcdPutStr (string)

To display a constant string. Eg,
glcdPutStr("IIT Kanpur");

itoa(int val, char arr[])

It stores the value of integer val in the character array arr. E.g., we have already defined int i and char c[20], then
itoa(i,c);
glcdPutStr (c);

HOW TO USE INTERNAL EEPROM?

What is the EEPROM memory and why would I use it?

Most of the AVRs in Atmel's product line contain at least *some* internal EEPROM memory. EEPROM, short for *Electronically Erasable Read-Only memory*, is a form of non-volatile memory with a reasonably long lifespan. Because it is non-volatile, it will retain its information during periods of no AVR power and thus is a great place for storing sparingly changing data such as device parameters.

The AVR internal EEPROM memory has a limited lifespan of 100,000 writes - reads are unlimited.

How is is accessed?

The AVR's internal EEPROM is accessed via special registers inside the AVR, which control the address to be written to (EEPROM uses byte addressing), the data to be written (or the data which has been read) as well as the flags to instruct the EEPROM controller to perform a write or a read.

The C language does not have any standards mandating how memory other than a single flat model (SRAM in AVRs) is accessed or addressed. Because of this, just like storing data into program memory via your program, every compiler has a unique implementation based on what the author believed was the most logical system.

Using the AVRLibC EEPROM library routines:

The AVRLibC, included with WinAVR, contains prebuilt library routines for EEPROM access and manipulation. Before we can make use of those routines, we need to include the eeprom library header:

Code:
```
#include <avr/eeprom.h>
```

At the moment, we now have access to the eeprom memory, via the routines now provided by eeprom.h. There are three main types of EEPROM access: byte, word and block. Each type has both a write and a read variant, for obvious reasons. The names of the routines exposed by our new headers are:

Quote:
```
uint8_t eeprom_read_byte (const uint8_t *addr)
void eeprom_write_byte (uint8_t *addr, uint8_t value)
uint16_t eeprom_read_word (const uint16_t *addr)
void eeprom_write_word (uint16_t *addr, uint16_t value)
void eeprom_read_block (void *pointer_ram, const void *pointer_eeprom, size_t n)
void eeprom_write_block (void *pointer_eeprom, const void *pointer_ram, size_t n)
```

In AVR-GCC, a word is two bytes while a block is an arbitrary number of bytes which you supply (think string buffers).

To start, lets try a simple example and try to read a single byte of EEPROM memory, let's say at location 46. Our code might look like:

Code:
```
#include <avr/eeprom.h>

void main(void)
{
   uint8_t ByteOfData;
   ByteOfData = eeprom_read_byte((uint8_t*)46);
}
```

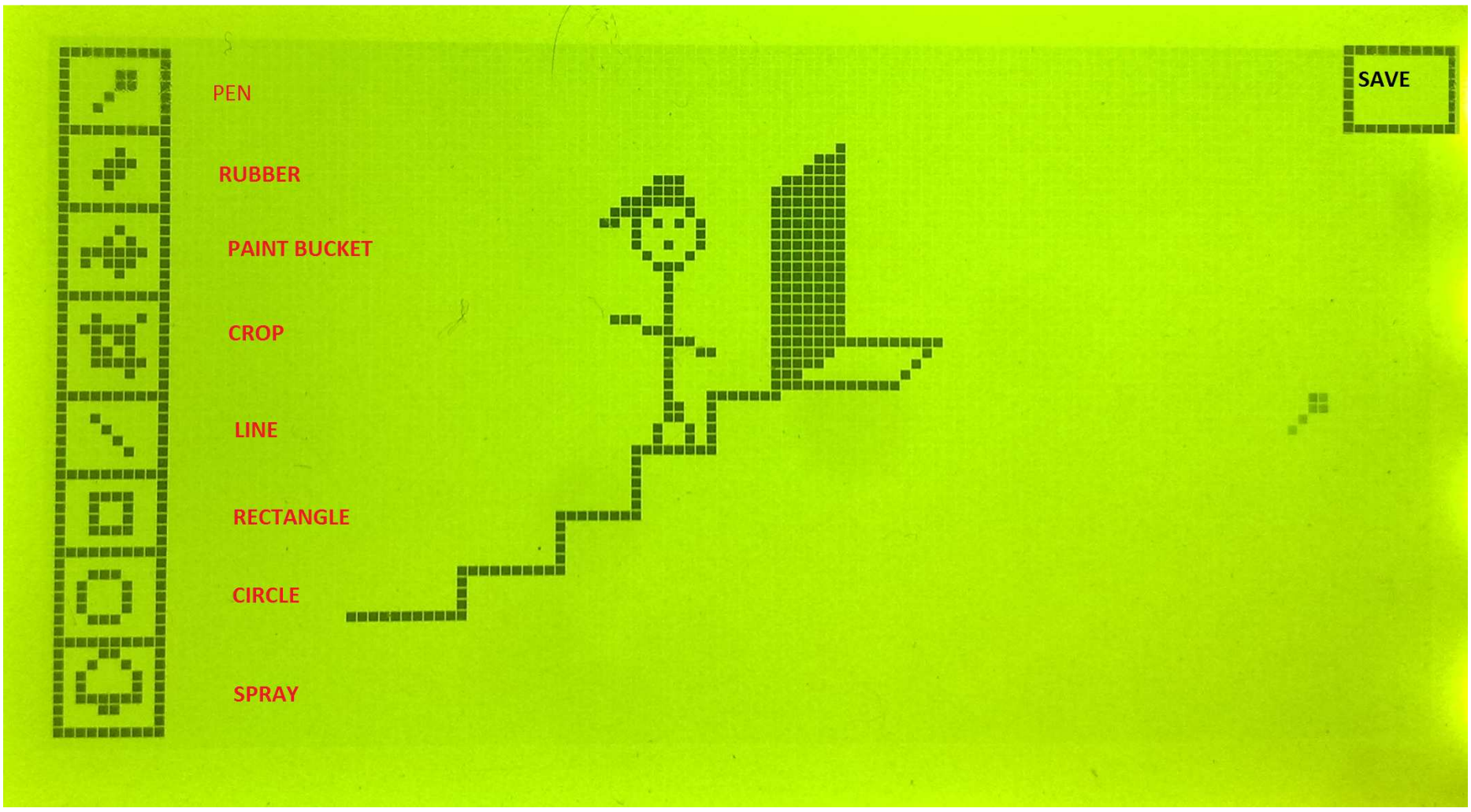
This will read out location 46 of the EEPROM and put it into our new variable named "ByteOfData". How does it work? Firstly, we declare our byte variable, which I'm sure you're familiar with:

Code:
```
uint8_t ByteOfData;
```

Now, we then call our eeprom_read_byte routine, which expects a pointer to a byte in the EEPROM space. We're working with a constant and known address value of 46, so we add in the typecast to transform that number 46 into a pointer for the eeprom_read_byte function.

Normal pointers specify the size of the data type in their declaration (or typecast), for example "uint8_t*" is a pointer to an unsigned byte and "int16_t*" is a pointer to a signed int.

Painter Window



Memory issue

Basically the biggest problem i faced was of memory.i wanted to store the information about each pixel whether it is black or white.there are 128x64 pixel and i tried to store each one just by using a unsigned integer only it fill require 8kb of memory space.so i d=find a interesting solution

‘1’ represent black

‘0’ represent white

So i was in need now to store only 0 ot one.since a unsigned integer is of 8 bit i can store 8 pixel in a single integer.so achived a 8 times compression and by chance the internal eeprom is also 1kb so now i can store it in it also.

- PEN : By using this we can darken any pixel.

- ERASER : Available in two sizes, by clicking we can erases any pixel.by right cliking we can change the size

- PAINT BUCKET :This function uses the flood fill algorithm to fill any boundary.

  Flood fill, also called ***seed fill,*** is an algorithm that determines the area connected to a given node in a multi-dimensional array. It is used in the "bucket" fill tool of paint programs to determine which parts of a bitmap to fill with color, and in games such as Go and Minesweeper for determining which pieces are cleared. When applied on an image to fill a particular bounded area with color, it is also known as ***boundary fill***.

The algorithm

The flood fill algorithm takes three parameters: a start node. The algorithm looks for all nodes in the array which are connected to the start node by a path of the target color, and changes them to the replacement color. There are many ways in which the flood-fill algorithm can be structured, but they all make use of a queue or stack data structure, explicitly or implicitly. One implicitly stack-based (recursive) flood-fill implementation (for a two-dimensional array) goes as follows:

```
Flood-fill (node, target-color, replacement-color):
 1. If the color of node is not equal to target-color, return.
 2. Set the color of node to replacement-color.
 3. Perform Flood-fill (one step to the west of node, target-color, replacement-
color).
    Perform Flood-fill (one step to the east of node, target-color, replacement-
color).
    Perform Flood-fill (one step to the north of node, target-color, replacement-
color).
    Perform Flood-fill (one step to the south of node, target-color, replacement-
color).
 4. Return.

This is what I have used.
```

- **CROP**: This function clears every thing apart from the portion we choose.

- SHAPES: shapes like line ,rectangle and circle are available in our mouse painter pro.ractangle and circle were available in glcd library but the line one was emply so I filled it u.

- SPRAY:A small version of ms-paint spray function. For randomly generate the number of pixels we want to darken we used the rand() function of math libraray.

- SAVE: for this I saved the whole 16x64 matrix in intternl eeprom.and I have alredy told how to acess internal eeprom.

Bibliography

This have been possible with the help of such helpful coordinators RAJAT ARORA , RISHAB MAHESHWARI AND GANESH PITCHIAH.

- http://en.wikipedia.org/wiki/Flood_fill
- http://www.computer-engineering.org/ps2protocol/
- http://www.computer-engineering.org/ps2mouse/
- http://courses.cit.cornell.edu/ee476/FinalProjects/s2004/jcc72/index.html