# PWM DRAWER

- EMBEDDED SYSTEM PROJECT

BY: MARYAM EMAD

# PWM DRAWER

Atmega32 Interfacing with GLCD

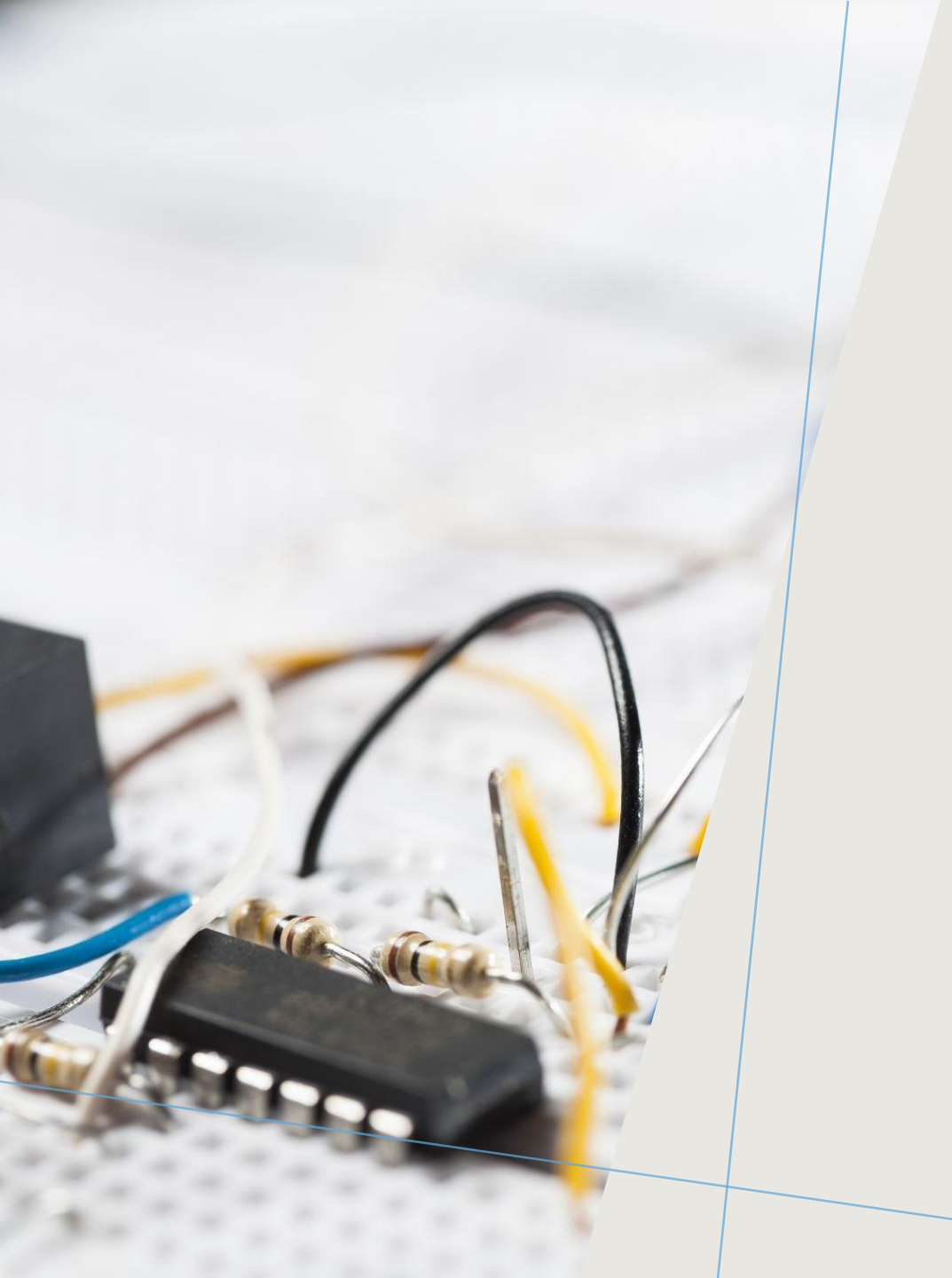
This presentation will provide a detailed overview of the *PWM Drawer* project.

This presentation will provide a detailed overview of the *PWM Drawer* project.

The goal is to display the quantity of PWM while allowing the user to draw and write on the LCD.

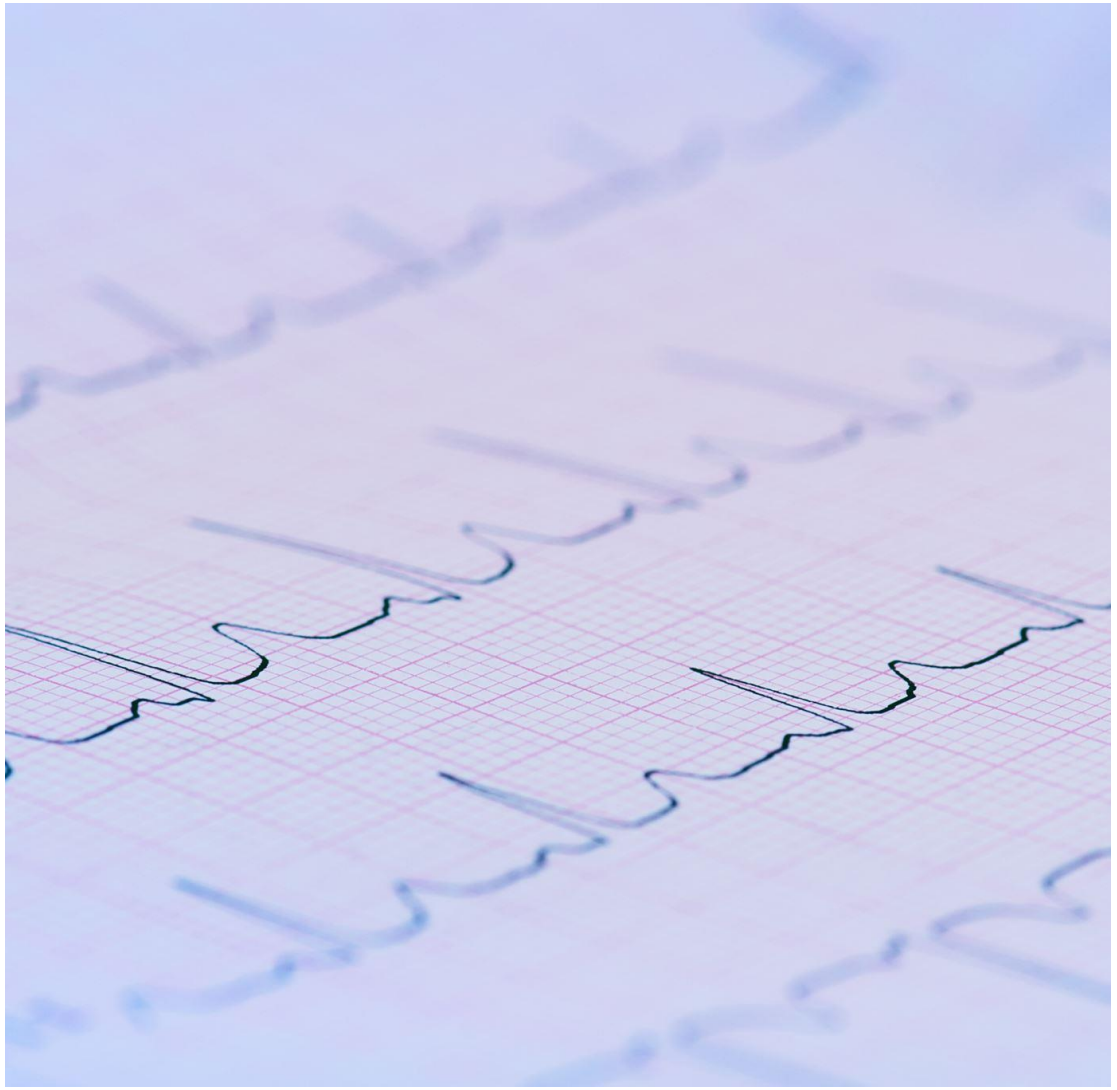
The project components and implementation details will be thoroughly discussed.

TABLE OF CONTENTS





- **Overview of the Embedded System Used**



—

- 1) Graphical
  - GLM12641BS1R
- **2)Microcontroller**
  - Atmega32
- **3)Pulse Width Modulation (PWM)**
  - PRE_64
  - Fast_PWM
  - Polling
  - Timer1
-

## GRAPHICAL LCD

An intuitive user interface

makes it easy to operate the drawer

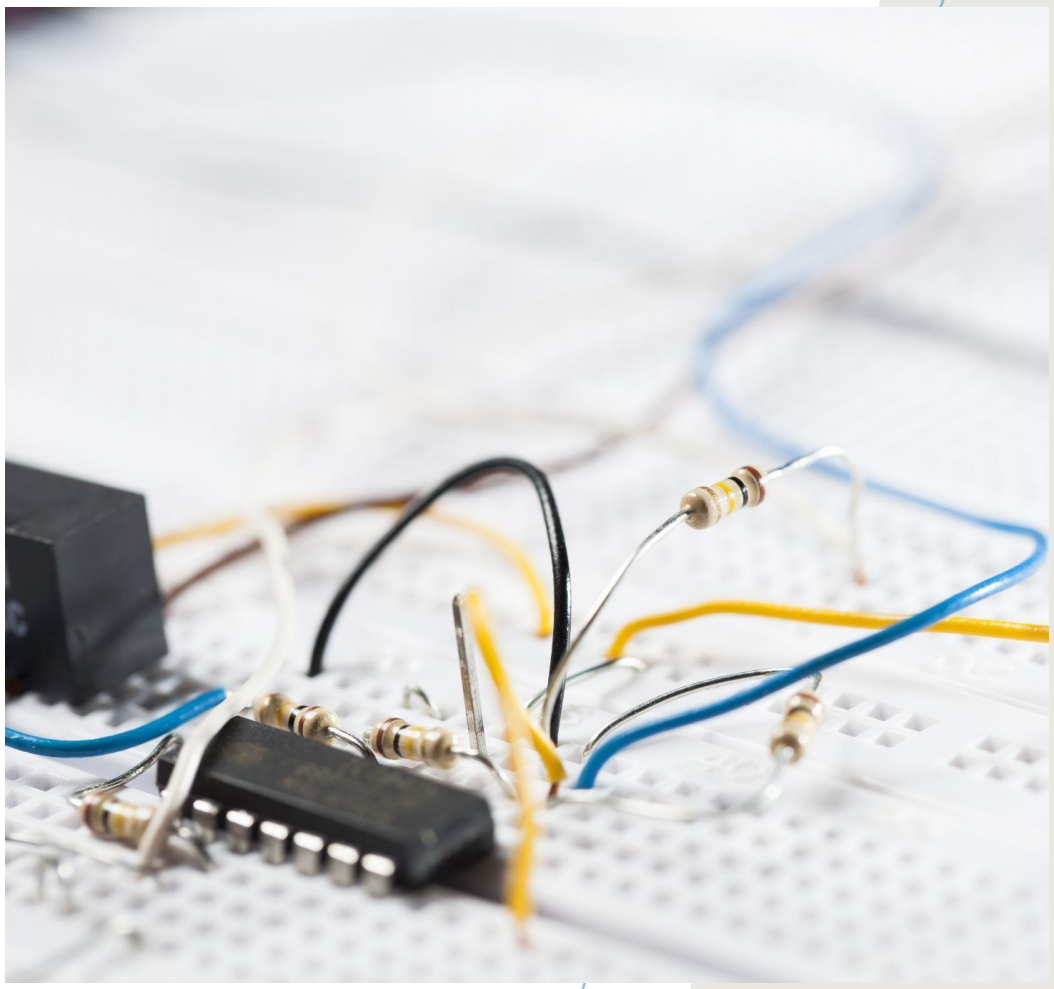
access its features seamlessly.

# MICROCONTROLLER

The system is powered by a high-performance

microcontroller designed for

Precision and reliability which is ATmega32



# PULSE WIDTH MODULATION (PWM)

- The system integrates PWM technology to control the power output allowing for precise control of the drawer.

# 02

# GLCD INTERACTION AND DISPLAY

Initial Interfacing

PWM Integration

Main Functionality

# INITIAL INTERFACING

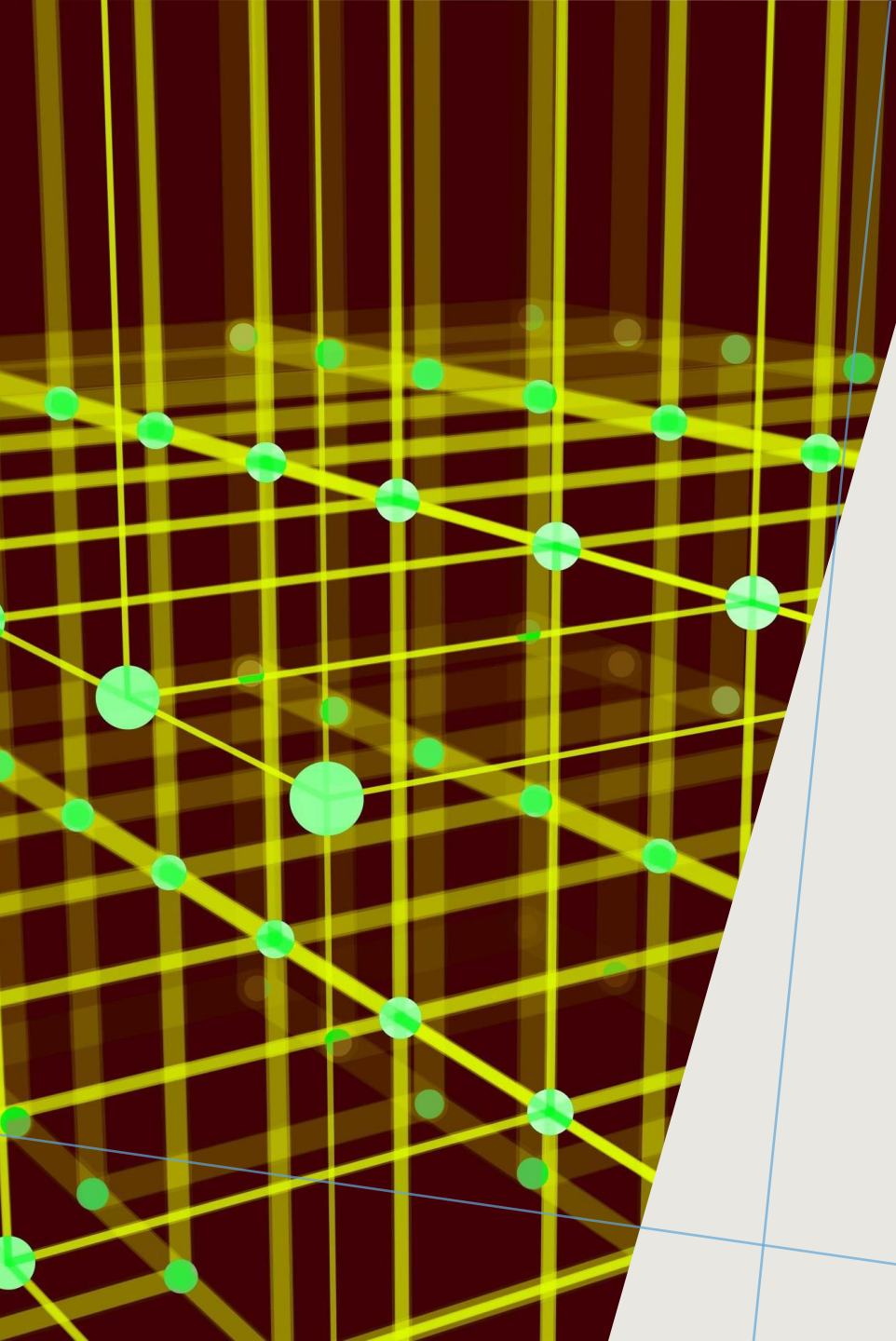
- Started by interfacing with the GLCD without PWM and achieved successful functionality for writing and drawing on it.

# PWM INTEGRATION

- Subsequently, incorporated PWM and established successful interfacing, leading to the final *main.c* integration.

## MAIN FUNCTIONALITY

- The *main.c* file depicted the comprehensive usage of PWM and GLCD for drawing PWM signals and numerical display.



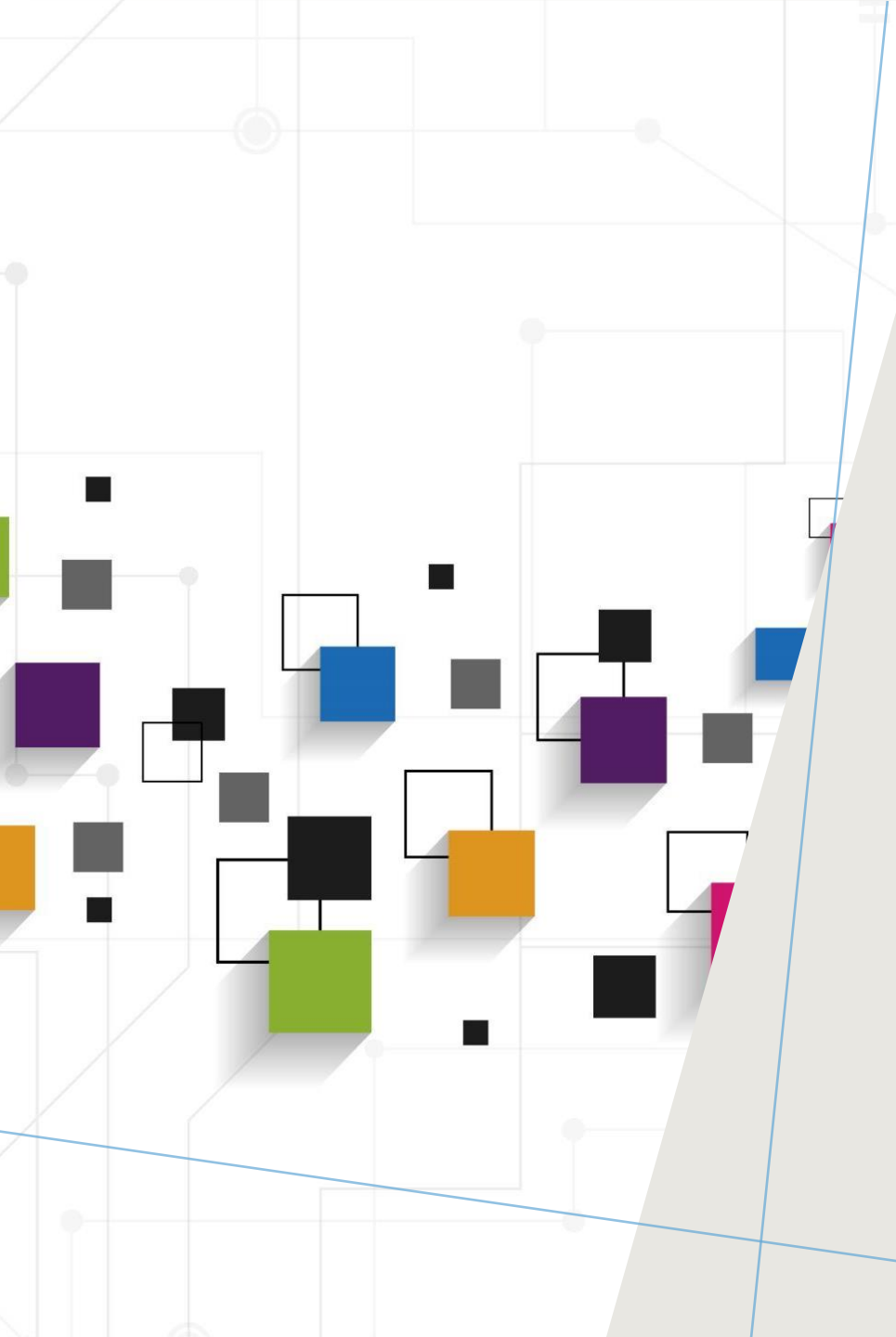
- **Implementation in Atmega32**

# ECUAL

HEADER FILES

INCLUDE FILES

MAIN
FUNCTIONALITY

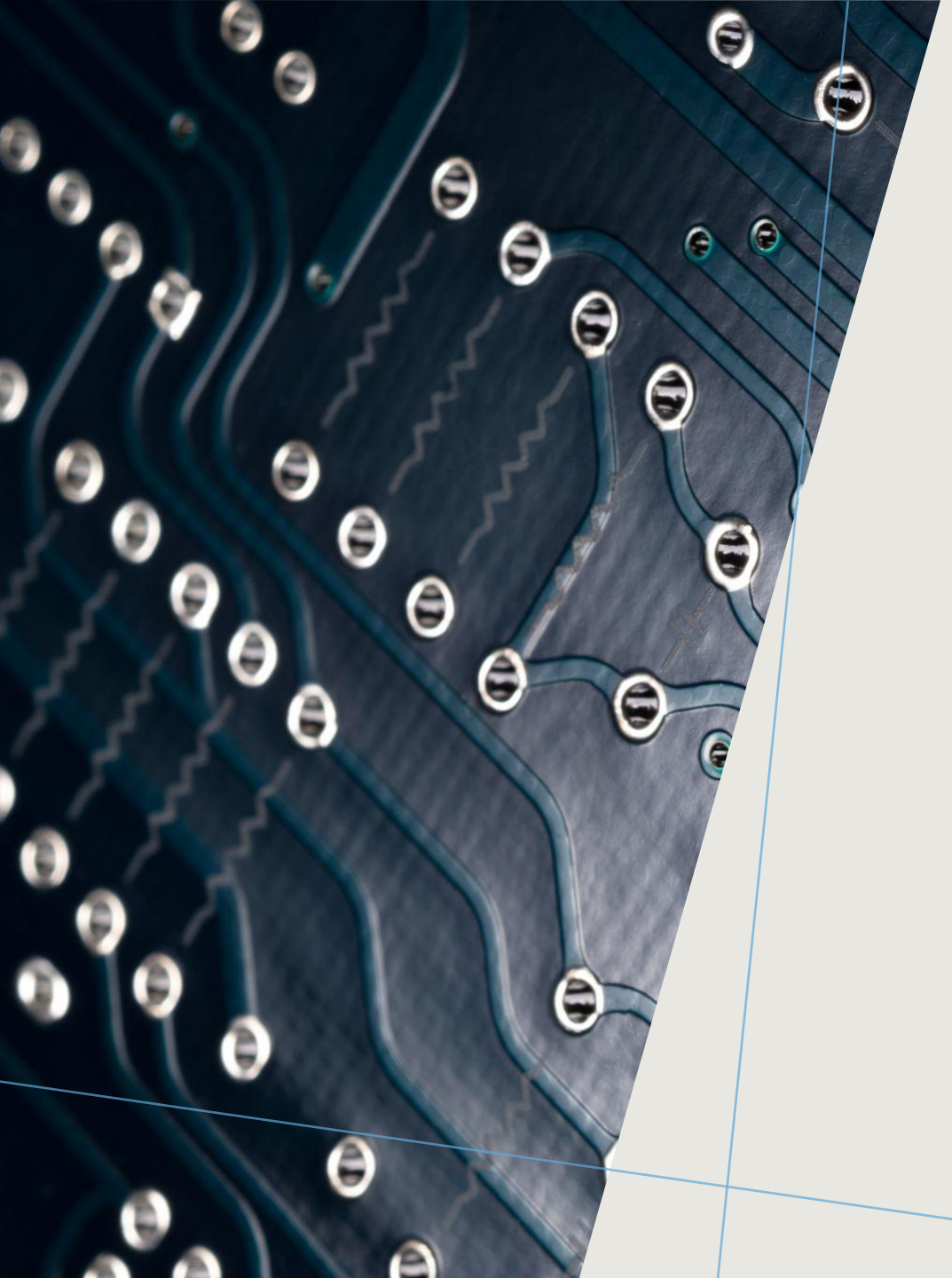


## HEADER FILES

- Introduced header files for GLCD interfacing consists of GLCD, GLCD_DATA (font and image) and PWM calculation, providing enhanced organization and modularity.



# SAME HEADER FILES MADE WILL HAVE ITS TWIN INCLUDE FILE FOR IMPLEMENTING THE FUNCTIONS



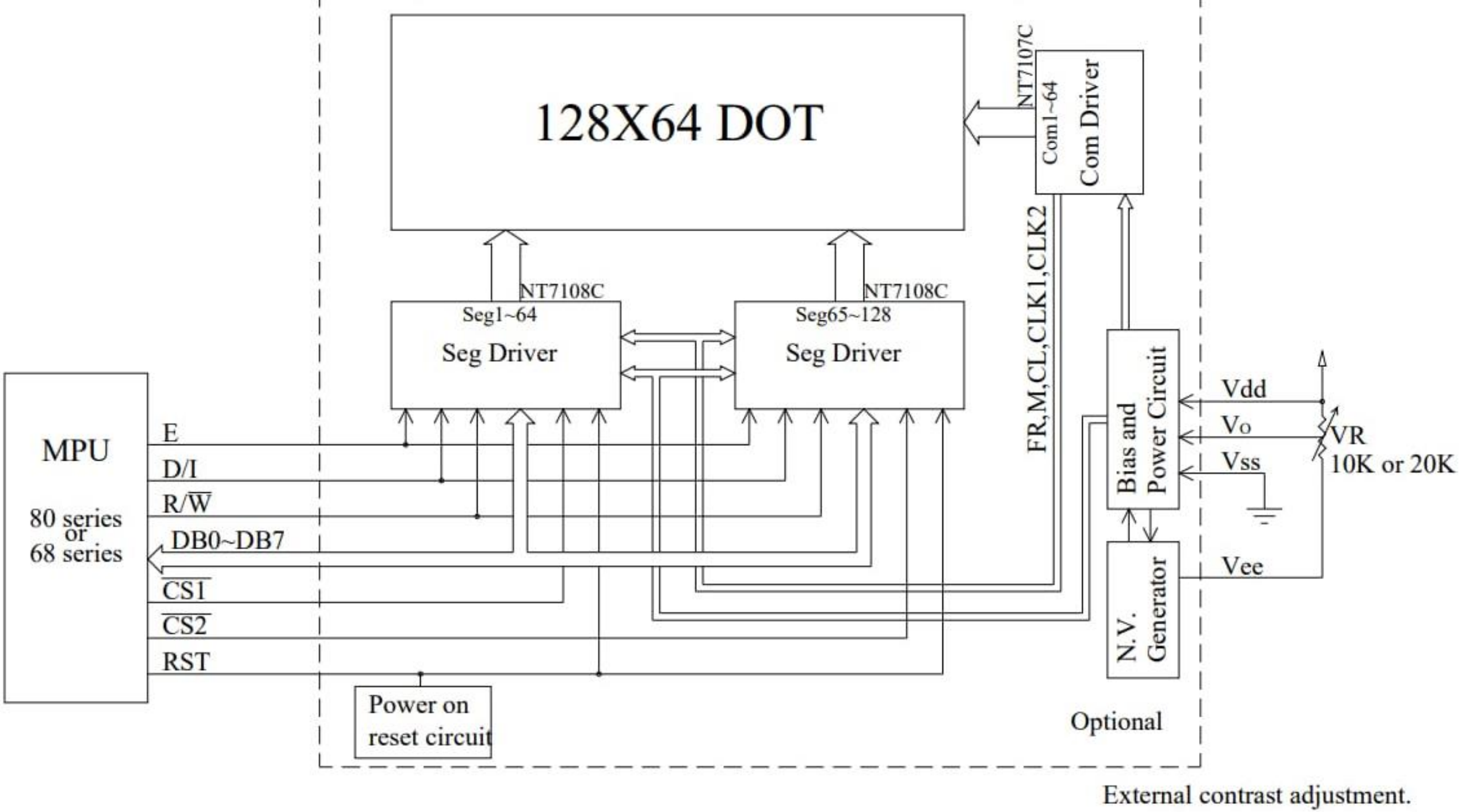
## MAIN FUNCTIONALITY

- Highlighted the *main.c* file responsible for integrating all components into a coherent and functional system.



- **Atmega32 and GLCD Interface**

- GLCD Interface
- Drawing
- Writing


128X64 DOT
NT7107C
Com1~64
Com Driver
NT7108C
Seg1~64
Seg Driver
NT7108C
Seg65~128
Seg Driver
FR,M,CL,CLK1,CLK2
Bias and Power Circuit
N.V. Generator
Vdd
Vo
Vss
Vee
VR
10K or 20K
MPU
80 series
or
68 series
E
D/I
R/W
DB0~DB7
CS1
CS2
RST
Power on reset circuit
Optional
External contrast adjustment.

| Instruction | RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | Function |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Display on/off | L | L | L | L | H | H | H | H | H | L/H | Controls the display on or off. Internal status and display RAM data is not affected. L:OFF, H:ON |
| Set address (Y address) | L | L | L | H | Y address (0-63) | | | | | | Sets the Y address in the Y address counter. |
| Set page (X address) | L | L | H | L | H | H | H | Page (0-7) | | | Sets the X address at the X address register. |
| Display Start line (Z address) | L | L | H | H | Display start line (0-63) | | | | | | Indicates the display data RAM displayed at the top of the screen. |
| Status read | L | H | Busy | L | On/ Off | Reset | L | L | L | L | Read status.<br>BUSY L: Ready<br>H: In operation<br>ON/OFF L: Display ON<br>H: Display OFF<br>RESET L: Normal<br>H: Reset |
| Write display data | H | L | Write data | | | | | | | | Writes data (DB0: 7) into display data RAM. After writing instruction, Y address is increased by 1 automatically. |
| Read display data | H | H | Read data | | | | | | | | Reads data (DB0: 7) from display data RAM to the data bus. |

# GLCD INTERFACE

**Define and connections**

TotalPage 8

Data_Port= PORTA  Command_Port=PORTC  Data_Port_Dir=DDRA Command_Port_Dir=DDRC

RS =PC0 RW=PC1  EN=PC2  CS1 =PC4

CS2=PC5  RST=PC3

**Functions**

```
void GLCD_Command(char Command);
```

```
void GLCD_Data(char Data);
```

```
void GLCD_Init();
```

```
void GLCD_ClearAll();
```

```
void GLCD_String(char page_no, char *str);
```

```
void GLCD_Str(const char*  image);
```

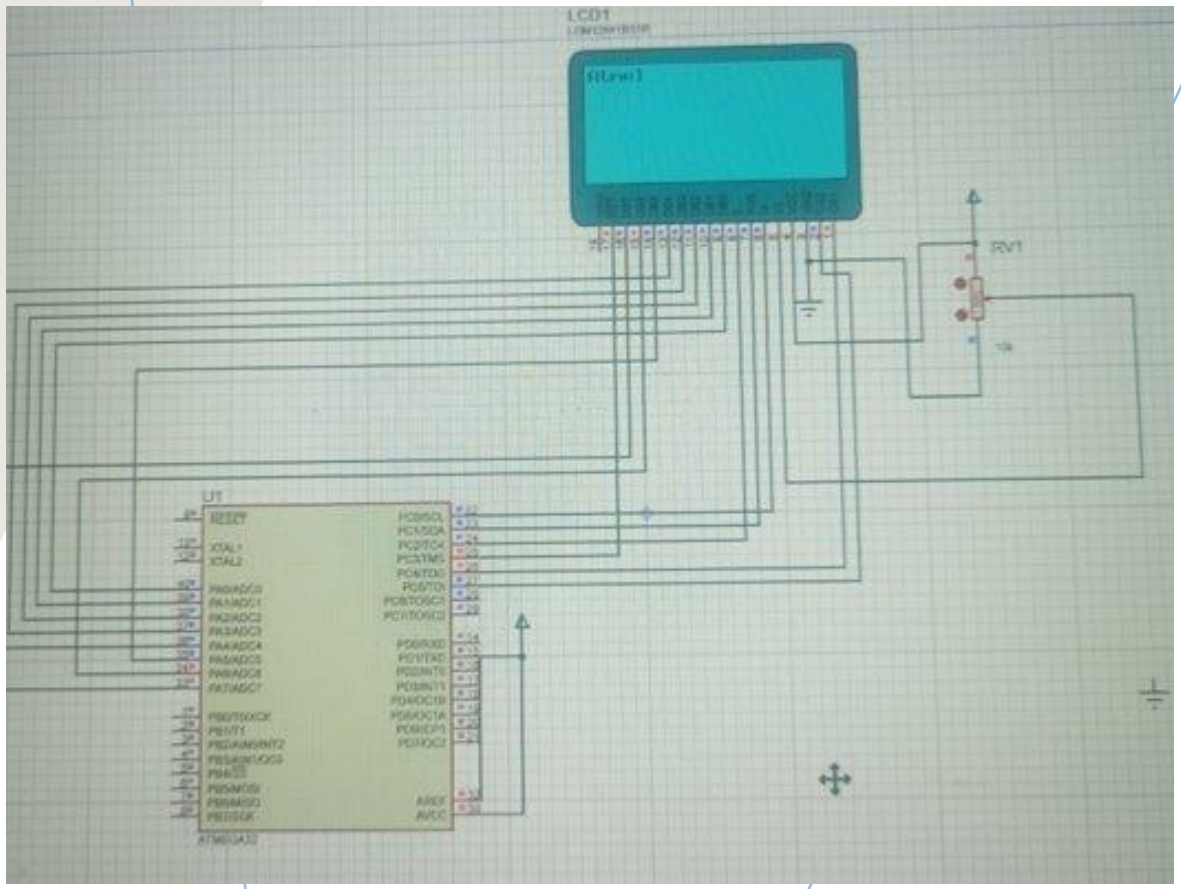
>>

- int main(void)
- {
- GLCD_Init(); /* Initialize GLCD */
- GLCD_ClearAll(); /* Clear all GLCD display */
- GLCD_String(img); /* Print Image Array */
- while(1);}

# WRITING

- **glcd-font.h**
- char font[][5];
- **glcd-font.c**
- char font[][5] = {
- {0x00, 0x00, 0x00, 0x00, 0x00} // 20 (Space)
- ,{0x00, 0x00, 0x5f, 0x00, 0x00} // 21 !
- ,{0x00, 0x07, 0x00, 0x07, 0x00} // 22 "
- ,{0x14, 0x7f, 0x14, 0x7f, 0x14} // 23 #

```
INT MAIN(VOID)
{
GLCD_INIT(); /* INITIALIZE GLCD
*/
GLCD_CLEARALL(); /* CLEAR
ALL GLCD DISPLAY */
GLCD_STRING(0,"ATMEL"); /*
PRINT STRING ON 0TH PAGE OF
DISPLAY */
WHILE(1);
}
```



# 05

- **PWM Calculation & Control**

# 1) CONTROL

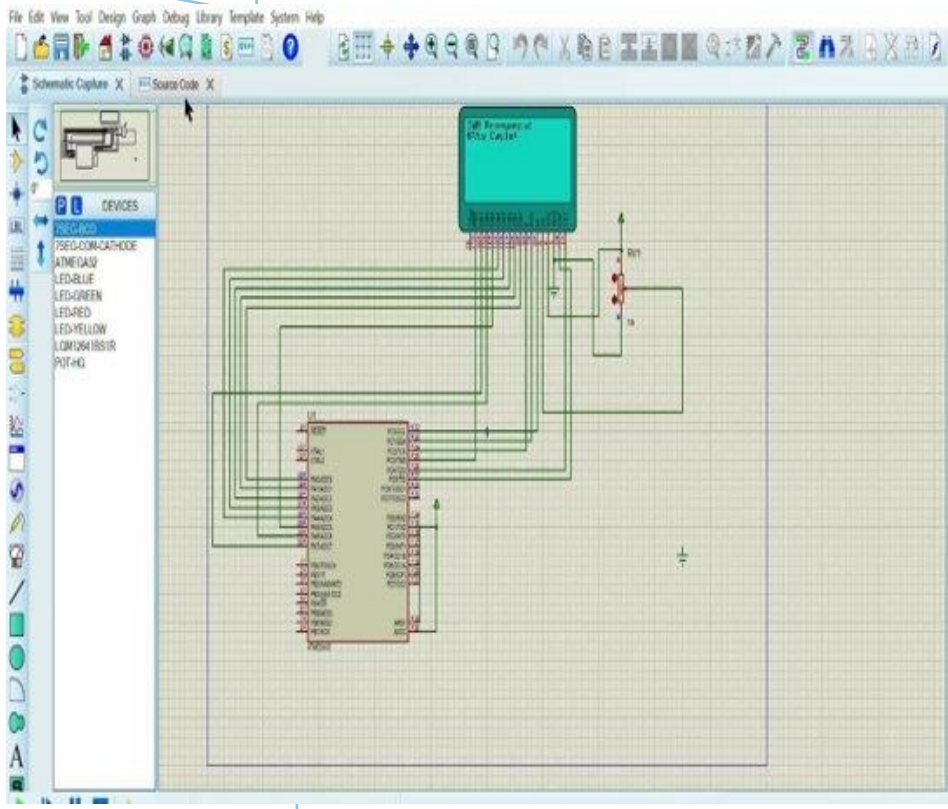
*PWM.h*

```
void PWM_init();
uint_16 PWM_getFrequency();
uint_8 PWM_getDutyCycle();
Void PWM_init() {
TIMER_cfgstruct pwmConfig;
pwmConfig.pre = PRE_64;
pwmConfig.mode = Fast_PWM;
pwmConfig.Op_mode = Polling;
pwmConfig.Timer = Timer1;
Timer_Setup(pwmConfig);
Timer1_start();
}
```

# 2) CALCULATIONS

- **uint_16 PWM_getFrequency();**
- The PWM frequency is calculated using the formula:
- Frequency = F_CPU / (2 * prescalerValue * (topValue + 1)) * (1 + compareValue / (topValue + 1))
  - `F_CPU`: Clock frequency of the microcontroller.
  - `prescalerValue`: The value used to divide the clock frequency.
  - `topValue`: The top value for the PWM waveform.
  - `compareValue`: The compare value for the PWM waveform.
- **uint_8 PWM_getDutyCycle();**
- Calculating Duty Cycle:
  - Based on the `com1aMode`:
    - For toggle mode or normal mode, duty cycle is assumed to be 0%.
    - For certain other modes, duty cycle is assumed to be 0%.
    - For clear on compare match or set on compare match mode, duty cycle is calculated as a percentage:
      - `dutyCycle = (compareValue * 100) / (topValue + 1)` or
      - `(topValue - compareValue) *100 / (topValue+1)` respectively.

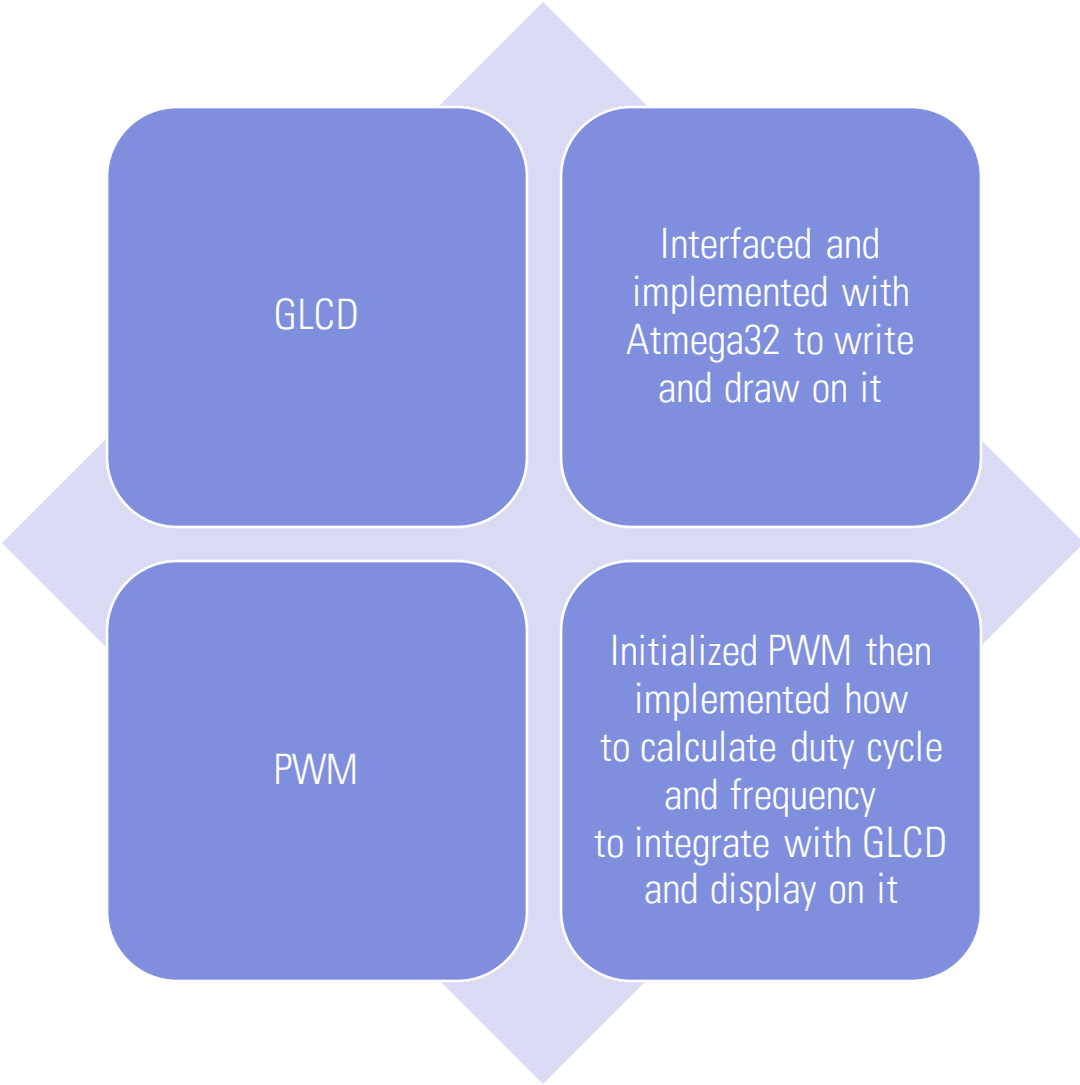
```
INT MAIN(VOID) {
PWM_INIT(); // INITIALIZE PWM
GLCD_INIT(); // INITIALIZE GLCD

WHILE(1) {
// READ PWM FREQUENCY AND DUTY CYCLE
UINT16_T FREQUENCY = PWM_GETFREQUENCY();
UINT8_T DUTYCYCLE = PWM_GETDUTYCYCLE();

// DRAW PWM SIGNAL ON GLCD
GLCD_CLEARALL(); // CLEAR THE GLCD DISPLAY
GLCD_STRING(0, "PWM FREQUENCY: "); // DISPLAY PWM FREQUENCY
GLCD_STRING(1, "DUTY CYCLE: "); // DISPLAY PWM DUTY CYCLE
CHAR FREQUENCY_STR[5];
CHAR DUTYCYCLE_STR[4];
SPRINTF(FREQUENCY_STR, "%D", FREQUENCY); // CONVERT FREQUENCY TO STRING
SPRINTF(DUTYCYCLE_STR, "%D%%", DUTYCYCLE); // CONVERT DUTY CYCLE TO STRING
GLCD_STRING(0, FREQUENCY_STR); // DISPLAY PWM FREQUENCY VALUE
GLCD_STRING(1, DUTYCYCLE_STR); // DISPLAY PWM DUTY CYCLE VALUE
}

RETURN 0;
}
```



# 06

## CONCLUSION

# 1) SUMMARIZE


GLCD
Interfaced and implemented with Atmega32 to write and draw on it
PWM
Initialized PWM then implemented how to calculate duty cycle and frequency to integrate with GLCD and display on it

# 2) APPLICATIONS

- **Harnessing PWM for Versatile Applications**
  - **Dynamic Lighting Control**: Implementing PWM allows for precise control over LED brightness, enabling dynamic lighting effects in various applications such as ambient lighting, signage, and automotive lighting.
  - **Motor Speed Regulation**: PWM can be employed to regulate motor speeds efficiently, enabling precise control over mechanisms in robotics, automation, and industrial systems.
  - **Analog Signal Generation**: By modulating PWM signals, analog-like waveforms can be synthesized, facilitating applications like audio synthesis, waveform generation, and sensor signal simulation.

- **Artistic Expression**: Enable creative individuals to express their artistic talents digitally through drawing and writing.
- **Educational Tools**: Offer educational institutions an innovative tool to facilitate interactive learning and creative expression.
- **Professional Design**: Provide professionals with a versatile tool for creating professional designs with precision and efficiency.

# *THANK YOU!*