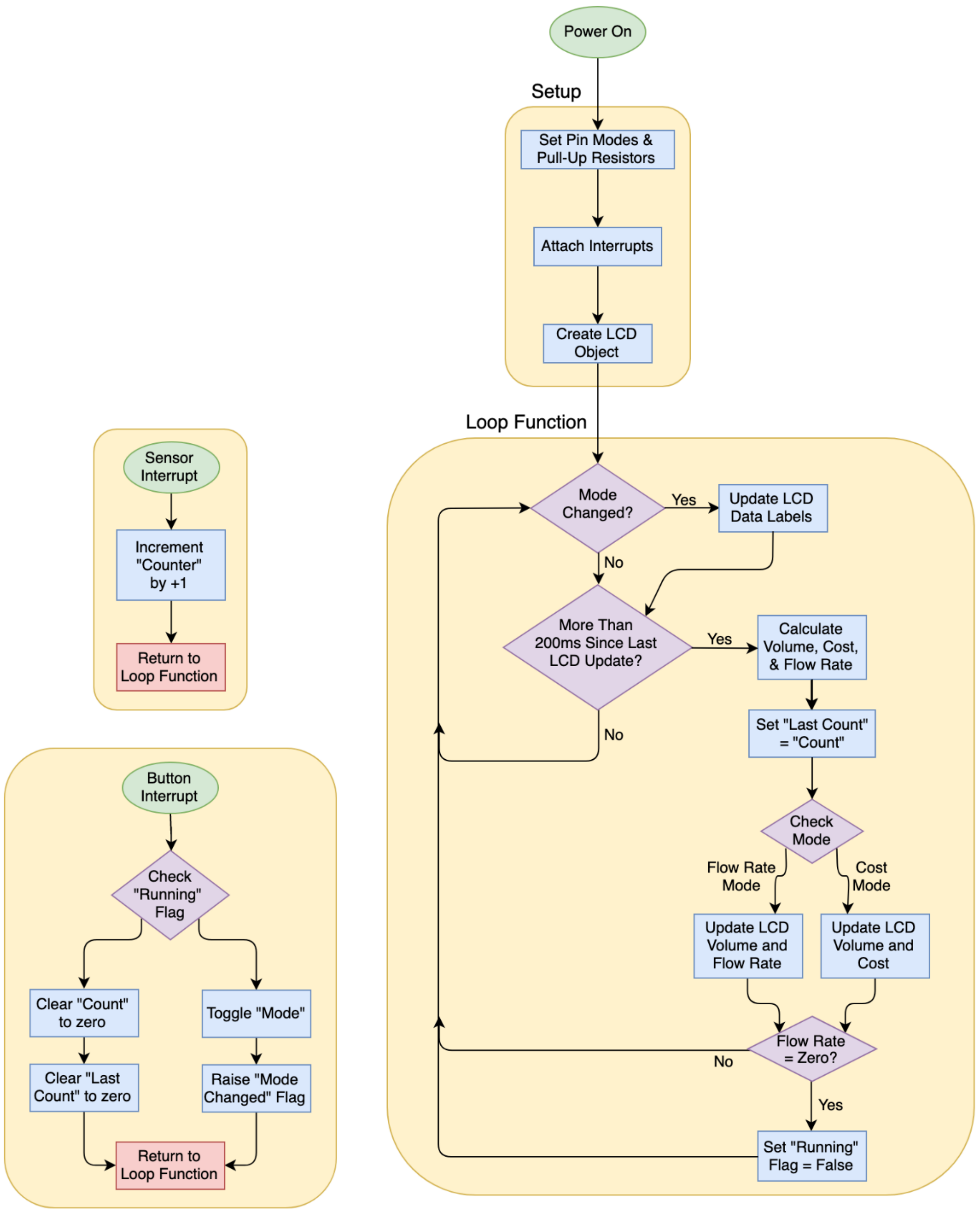
Software Explanation

1. Mind Map


Power On
Setup
Set Pin Modes & Pull-Up Resistors
Attach Interrupts
Create LCD Object
Loop Function
Mode Changed?
Yes
Update LCD Data Labels
No
More Than 200ms Since Last LCD Update?
Yes
Calculate Volume, Cost, & Flow Rate
No
Set "Last Count" = "Count"
Check Mode
Flow Rate Mode
Cost Mode
Update LCD Volume and Flow Rate
Update LCD Volume and Cost
Flow Rate = Zero?
No
Yes
Set "Running" Flag = False
Sensor Interrupt
Increment "Counter" by +1
Return to Loop Function
Button Interrupt
Check "Running" Flag
Clear "Count" to zero
Clear "Last Count" to zero
Toggle "Mode"
Raise "Mode Changed" Flag
Return to Loop Function

2. Drivers

2.1 LCD

All of LCD pins are on the same port which is **PORTA** of the microcontroller as this port does not have any predefined registers so we can easily set our driver on and it will work properly.

We have chosen to work with 4-bit data mode as it will cost us less pins and this will help us generate a good-looking PCB design as well as we do not need the full speed of the LCD. Therefore, we decided to use the first four pins of the LCD **from PIN (0) to PIN (3)**.

The other hardware pins are connected and programmed as follows:

- Enable Pin is connected to PIN (5) of the Microcontroller
- Read/Write is connected to PIN (6) of the Microcontroller
- Register Select is connected to PIN (6) of the Microcontroller

Then, we have implemented various functions to be generic for every project in the future and to be portable to work with any IDE.

The function names and description are implemented in that way:

- **LCD_init:** Initialize the LCD by setting up the pins direction and choosing the data mode to work with.
- **LCD_sendCommand:** Send the required command to the screen.
- **LCD_displayCharacter:** Display the required character on the screen
- **LCD_displayString:** Display the required string on the screen
- **LCD_moveCursor:** Move the cursor to a specified row and column index on the screen
- **LCD_displayStringRowColumn:** Display the required string in a specified row and column index on the screen
- **LCD_intgerToString:** Display the required decimal value on the screen
- **LCD_print_float:** Display the required floating value on the screen
- **LCD_clearScreen:** Send the clear screen command

2.2 Buzzer

This driver is just a simple driver as it only required to make the buzzer beep when the program code requires.

This driver have been made only to make our project on the type of the layered ones, therefore we can change the microcontroller easily.

We decided to connect the buzzer to **PIN (6)** of **PORT D** as it was simple for us to draw our PCB track without unwanted overlaps.

This driver consisted of (3) main functions which names and functionality shown in the upcoming lines:

- **Buzzer_Init:** Initialize Buzzer by adjusting the direction of its Pin as Output
- **Buzzer_On:** Turn on the Buzzer to start its beeping for specific action
- **Buzzer_Off:** Turn off the Buzzer to stop its beeping when needed

3. Explanation

This project is designed with concept of the layered architecture to make it portable with any kind of IDE and could be uploaded to other types of microcontrollers with simple edits.

3.1 H File

We started with the specification of the sensor and the button pins to **PORT D** with **PINS (2) and (3)** respectively. Since we are using it to generate interrupts to avoid using the blocking polling methods which could affect the functionality of the project and its calculations.

After that, we have produced some predefined named definitions to make our project easy to read and edit for other programmers:

- **RUNNING (1) / STOPPED (0):** values to track the water status
- **Rate_Mode (1) / Cost_Mode (0):** values to track the current active mode
- **REFRESH_RATE:** constant value to update the screen every 200 milli second to keep track of the updated values
- **COST_PER_LITER:** constant value to calculate the cost of water per liter in piasters according to the first price bracket of the Egyptian law

Then we started to declare global variables for the main parameters of our project, so it can be accessed in the main and interrupts' sections and generated as follows:

- **g_pulses:** store the number of pulses generated from the flow sensor
- **g_last_pulses:** store the last number of pulses generated from the flow sensor
- **g_mode:** store the mode of the screen
- **g_mode_changed:** store if the mode of the screen has changed
- **g_water_status:** store the status of the water

3.2 C File

We started with **INT0** and setup it to receive signal from the sensor to avoid the polling technique as it could cost us lack of efficiency which we do not want, the details of the interrupt was made as follows:

- **Setup:**
    - Configure the sensor pin as Input Pin
    - Trigger INT0 with the Raising Edge
    - Enable external interrupt pin INT0
- **Functionality:**
    - Increment the counter of the pulses received from the sensor

Then we moved to **INT1** and setup it to receive the button click to be able to generate a specific action according to the interrupt predefined action, the details of the interrupt was made as follows:

- **Setup:**
    - Configure the button pin as Input Pin
    - Trigger INT1 with the Raising Edge
    - Enable external interrupt pin INT1
- **Functionality:**
    - Water running phase: toggle the mode and change the flag indicting that the mode has changed
    - Water stopped phase: clears the counters of the number of pulses and the last number of pulses

After that, we have started to implement our main function and gone through some specific steps which are:

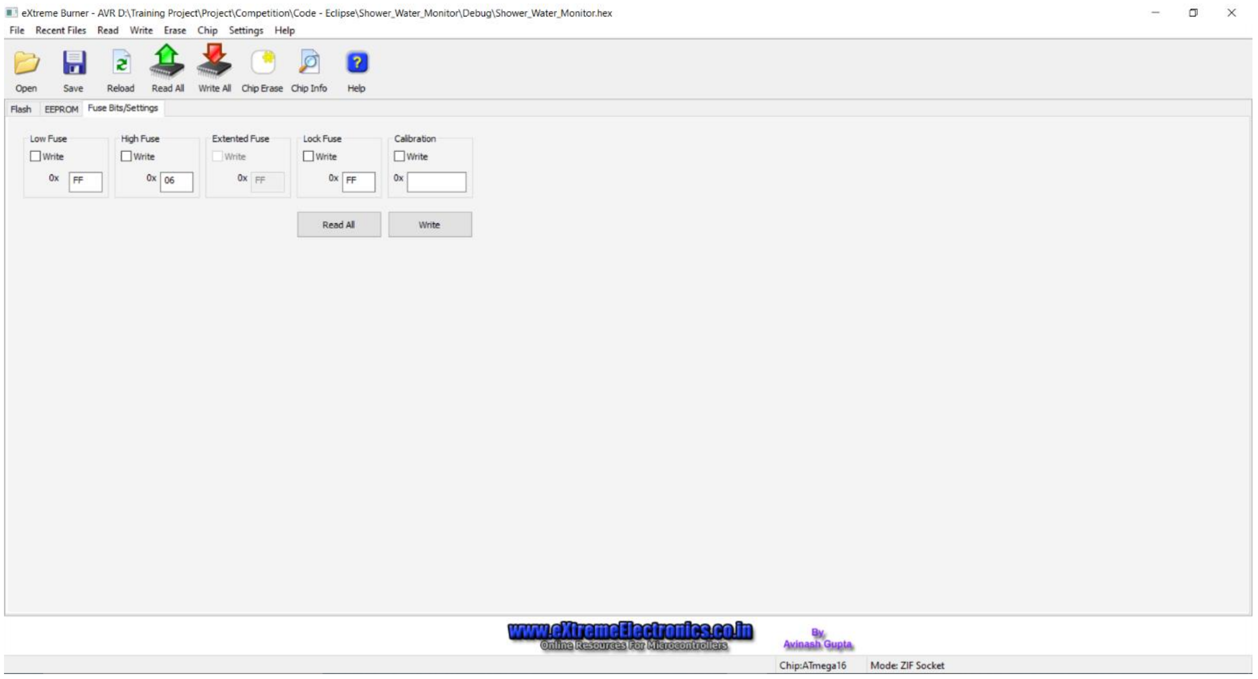
- **Initialization:**
    - Variables to store the parameters for further calculations
        - volume, cost, rate
    - Initialize drivers to be ready to call functions from
        - LCD, Buzzer
    - Enable global interrupts to be able to generate the pre-talked interrupts

- **Application Code:**
  - Present the text labels for the values to be introduced
    - Total, Cost, Rate
  - Calculate the values through mathematical equation
    - volume = g_pulses / 450.00
    - cost = volume * COST_PER_LITER
    - rate = 1000.00 * (g_pulses - g_last_pulses) / 450.00 / REFRESH_RATE
  - Present the values that was calculated
    - Total, Cost, Rate
  - Start the buzzer when reaches 55 litres
  - Assign the values of the counters with its new values
    - g_last_pulses, g_pulses, g_water_status
  - Delay with the refresh rate

4. PROGRAMMING PHASE

We have used in-circuit programming plugin as designed from the first in PCB design with 8-Mhz frequency oscillator, so we used USBasp from Atmel.

And we uploaded the project with the help of (eXtreme Burner – AVR) with following fuse bits to make our controller re-programmable:



eXtreme Burner - AVR D:\Training Project\Project\Competition\Code - Eclipse\Shower_Water_Monitor\Debug\Shower_Water_Monitor.hex

Low Fuse	High Fuse	Extented Fuse	Lock Fuse	Calibration
☐ Write	☐ Write	☐ Write	☐ Write	☐ Write
0x FF	0x 06	0x FF	0x FF	0x

Chip:ATmega16 Mode: ZIF Socket