MIP CEP REPORT

Ali Ahmed & Abdullah khan | 22i-2174 & 22i-1793

Description Of Project

- **Objectives**
  The project aims to design a microcontroller-based system to display Islamic prayer timings on an LCD, enable user input for customization via a keypad, and dynamically calculate and display the countdown to the next prayer. Lookup tables will be used to store weekly timings, accommodating seasonal variations.

- **Methodology**
  The system uses an AVR ATmega32 microcontroller to manage inputs, outputs, and real-time processing. Keypad inputs set prayer timings, which are stored in lookup tables. Timers and interrupts ensure real-time countdowns and dynamic updates. The design will be simulated in Proteus and programmed in Atmel Studio. The system will integrate a 16x2 LCD for output display and utilize efficient hardware-software interfacing techniques.

- **Outcomes**
  The final system will effectively display prayer timings, dynamically update countdowns, and handle user inputs seamlessly. It will demonstrate accurate realtime functionality, robust performance, and efficient resource utilization, providing a reliable and user-friendly solution for prayer scheduling.

Components Used:

1. AVR ATmega32 Microcontroller: Controls the entire system, manages inputs from the keypad, outputs to the LCD, and handles timers/interrupts.
2. LCD (16x2): Used to display current prayer times and countdowns to the next prayer.
3. Keypad (4x3): Allows the user to manually enter the prayer times.
4. Lookup Tables: Stores the prayer timings for each day of the week, accommodating seasonal shifts in timing.
5. Timers: Manages real-time calculations, such as countdowns until the next prayer.
6. Interrupts: Ensures the system can update the LCD display in real-time without delays.
7. Power Supply: Provides necessary voltage to all components.

Pin Configuration:

LCD: Port A For Data Pins and Port B for control pins

KEYPAD: PORTC IS USED

PinC(0-3) for rows and PinC(5-7) for columns

Challenges & Solution

1. **Coding Complexities**
    - **Challenge**: Managing timers and interrupts for real-time updates was complex, particularly ensuring synchronization between input handling and countdown displays.
    - **Solution**: The code was modularized into smaller functions for clarity, and extensive debugging was done using Atmel Studio. Interrupt service routines (ISRs) were optimized to minimize execution time and avoid conflicts.
2. **Hardware Interfacing Issues**
    - **Challenge**: Interfacing the microcontroller with the LCD and keypad posed difficulties in signal consistency and input/output timing.
    - **Solution**: Datasheets were carefully reviewed, pull-up resistors were added where necessary, and delays were fine-tuned to ensure proper signal transmission.
3. **Lookup Table Implementation**
    - **Challenge**: Implementing and accessing weekly prayer timings from lookup tables required efficient memory usage and accurate indexing.
    - **Solution**: The lookup table was structured using arrays, and day-to-week mapping was carefully tested to ensure correct timing retrieval.
4. **Real-Time Functionality**
    - **Challenge**: Maintaining real-time accuracy for countdowns and switching prayer times without system lag.
    - **Solution**: Hardware timers were used alongside precise delay loops. Interrupt-driven logic ensured consistent updates without overloading the CPU.
5. **Simulation Limitations**
    - **Challenge**: Simulating the entire system in Proteus was difficult due to potential mismatches between virtual and real-world behavior.
    - **Solution**: After successful simulation, functionality was validated on physical hardware to ensure seamless operation.
6. **User Input Handling**
    - **Challenge**: Ensuring error-free manual input via the keypad and managing incorrect or incomplete inputs.

- **Solution**: Input validation routines were implemented to reject invalid entries and provide prompts for correction.

7. **Seasonal Adjustment of Timings**

   - **Challenge**: Accounting for the slight daily shifts in prayer times required precise calculations as the difference in prayers was of either 1 or 2 minutes for every day.

- **Solution**: Lookup tables were pre-filled with adjusted timings based on reliable prayer time data, ensuring accurate seasonal variations.

Algorithm:

Define Atmega3

Equate:

rPrayer = R20(Stores the current prayer .)

rDay = R21(Tracks the current day of the week.)

rHour = R22(Holds the current hour.)

rMinute = R23(Holds the current minute.)

rTimeHigh = R24, rTimeLow = R25(Used for prayer time values in minutes.)

Initialize STACK (Stacks will be used as interrupts and functions are to be called)

Store high RAMEND in SPH

Store low RAMEND in SPL

Set PORTS

Set PORTA as output for LCD data (bits C4-C7).

Set PORTC as input for Keypad data to store in lookup table

Set PORTB as output for LCD control signals (RS and EN).

Define a lookup table for prayer times for each day of the week

Initialize LCD Function

Set control pins (RS and EN) as output.

Send initialization commands to LCD:

4-bit mode with 2 lines.

Clear RS bit(command mode).

Send command in two 4-bits

Set RS bit(data mode).

Send data in two 4-bits

Output the higher 4-bits to PORTC(LCD PORT)

Set EN bit.

Swap 4-bits in r16 and repeat to send the lower 4-bits.

Timer1 Function

-Set Timer1 to CTC mode(Set WGM1) with a 1-second interval using compare register value(OCRA & OCRB)

Enable Timer1 compare interrupt.

Enable global interrupt(SEI)

Start Timer1 with a suitable prescaler.

Make a function keypad_scan

Scan keypad for user input.

If user inputs a new prayer time, update the lookup table and refresh the display.

Calculate lookup table address based on rDay(current day) and rPrayer(current prayer).

Load prayer time into rTimeHigh and rTimeLow.

Make Calculate Countdown(`calculate_countdown`) Function

Subtract the current time from the next prayer time to find the remaining time.

Convert the result to HH:MM format for display.

Call `lcd_init` to initialize the LCD.

Call `timer1_init` to start Timer1.

Set the initial time by setting rHour to 0 and clearing rMinute.

Set rDay and rPrayer to 0 (start of the week, Fajr prayer).

Load the first prayer time and calculate the countdown.

Continuously perform the following:

Call `update_display` to show current time and countdown on LCD.

Call `keypad_scan` to check for keypad input.

Repeat to keep updating display and checking for input.

Convert rHour:rMinute and countdown time to HH:MM format.

Send the converted time to the LCD for display continuously in a loop

ISR_Prayer:

Increment rMinute every second.

If rMinute reaches 60, reset to 0 and increment rHour.

If rHour reaches 24, reset to 0, advance to the next day by incrementing rDay.

If rDay reaches 7, reset to 0 (start of the week).

Load the next prayer time for the new day by calling `load_prayer_time`.

Call `calculate_countdown` to update the countdown until the next prayer.

SBI DDRB,3(Pin3 of portb is connected to buzzer)

SBI PORTB,3

Call Delay Function

CBI PORTB,

Make Delay Function(Delay of 2 seconds)

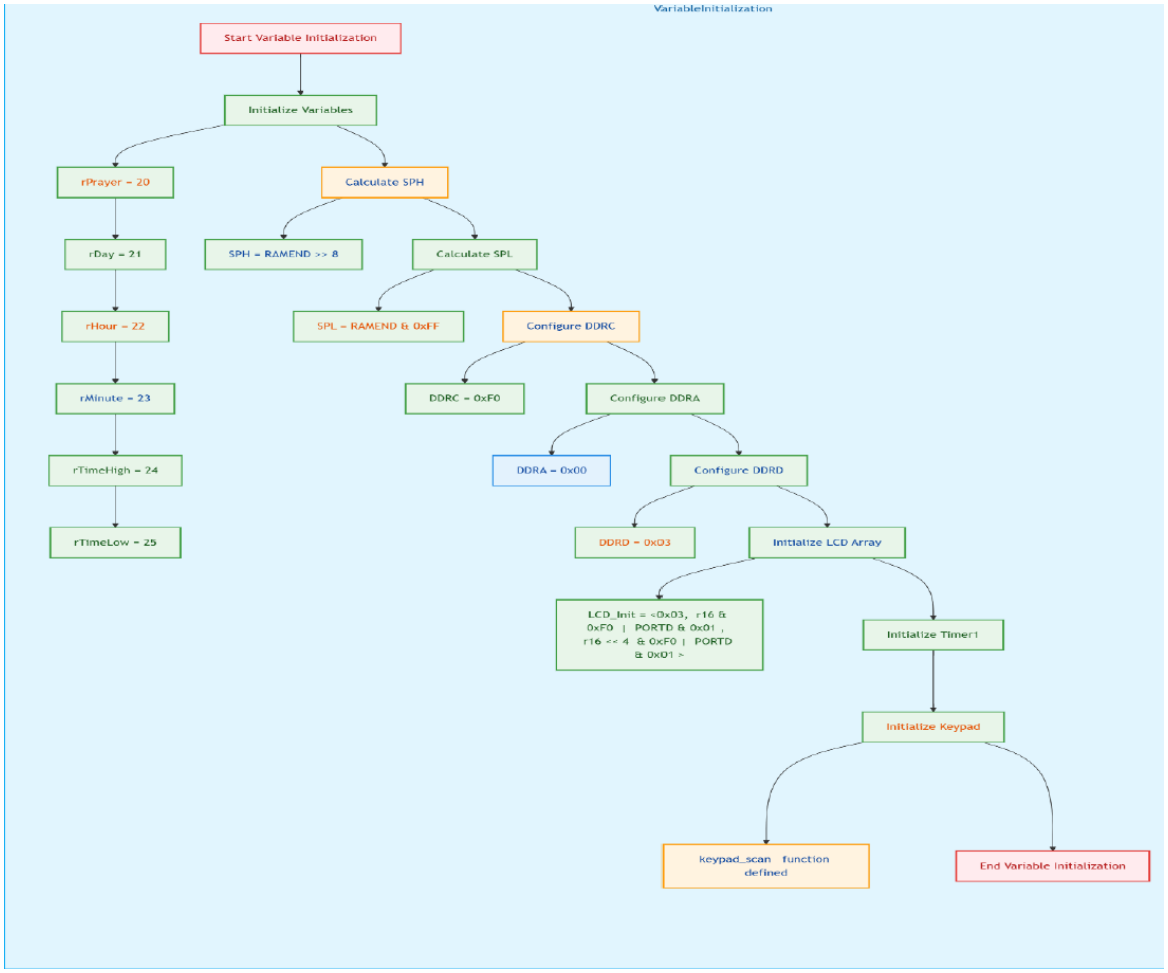
Set timer0 in normal mode with a suitable prescalar

Store Value in TCNT0

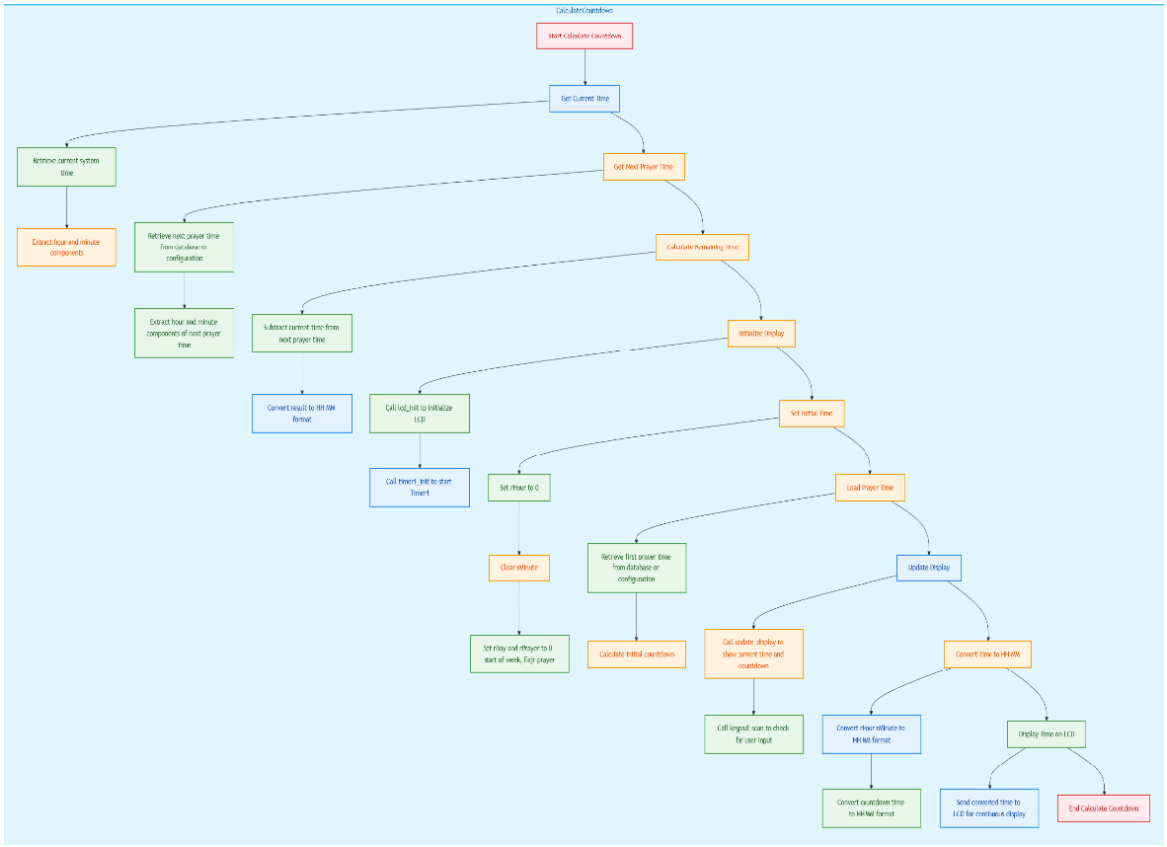
Count the delay

Flowchart:

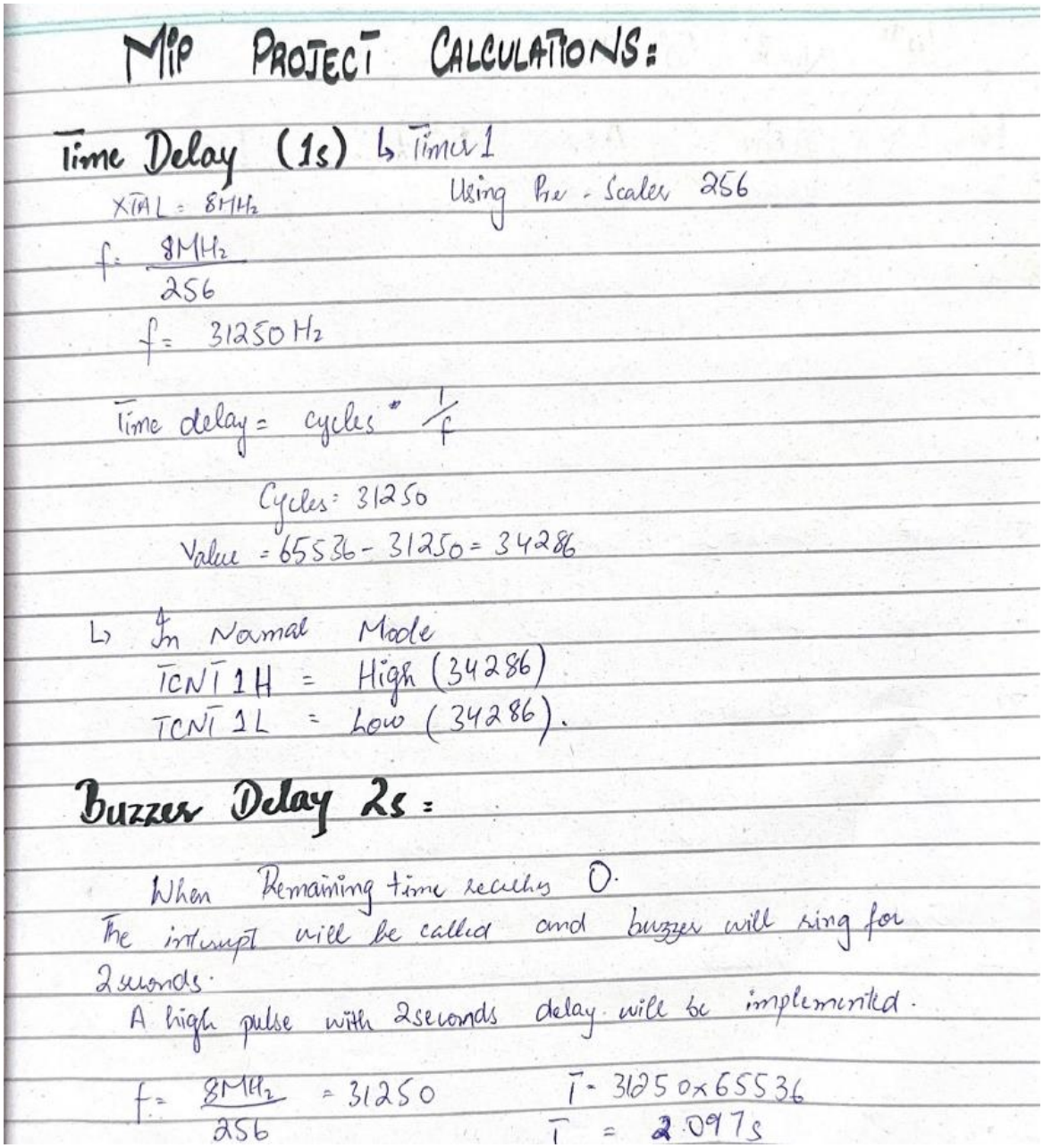
Timer Configuration & Main


VariableInitialization
Start Variable Initialization
Initialize Variables
rPrayer = 20
rDay = 21
rHour = 22
rMinute = 23
rTimeHigh = 24
rTimeLow = 25
Calculate SPH
SPH = RAMEND >> 8
Calculate SPL
SPL = RAMEND & 0xFF
Configure DDRC
DDRC = 0xF0
Configure DDRA
DDRA = 0x00
Configure DDRD
DDRD = 0x03
Initialize LCD Array
LCD_Init = <0x03, r16 & 0xF0 | PORTD & 0x01 , r16 << 4 & 0xF0 | PORTD & 0x01 >
Initialize Timer1
Initialize Keypad
Keypad_scan function defined
End Variable Initialization

Calculation countdown Function:



Calculations:



MIP PROJECT CALCULATIONS:

Time Delay (1s) ↳ Timer 1

XTAL = 8MHz Using Pre-Scaler 256

$f = \frac{8MHz}{256}$

$f = 31250\,Hz$

Time delay = cycles * 1/f

Cycles = 31250

Value = 65536 - 31250 = 34286

↳ In Normal Mode

TCNT1H = High (34286)

TCNT1L = Low (34286).

Buzzer Delay 2s =

When Remaining time reaches 0.
The interrupt will be called and buzzer will ring for 2 seconds.
A high pulse with 2seconds delay will be implemented.

$f = \frac{8MHz}{256} = 31250$ $T = 31250 \times 65536$

$T = 2.097s$

Look-up Table:

Look-Up Table:

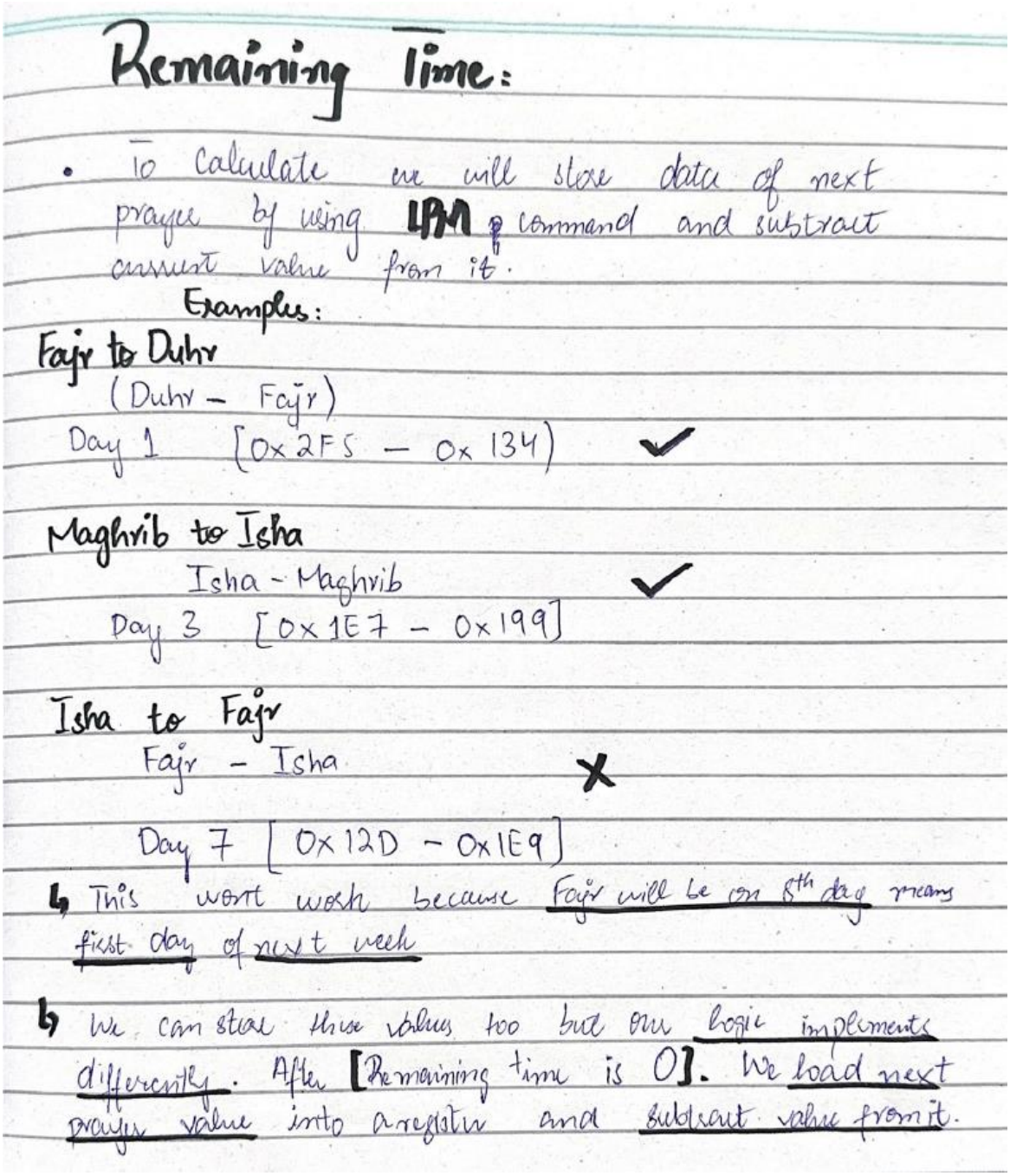
14TH Week Of. 2024

	Fajr	Zuhr	Asr	Maghrib	Isha
1)	0x134	0x2F5	0x130	0x199	0x1E6
2)	0x132	0x2F4	0x130	0x199	0x1E6
3)	0x131	0x2F4	0x130	0x199	0x1E7
4)	0x130	0x2F4	0x130	0x19A	0x1E7
5)	0x12F	0x2F3	0x131	0x19A	0x1E8
6)	0x12E	0x2F3	0x131	0x19B	0x1E8
7)	0x12D	0x2F3	0x131	0x19B	0x1E9

↳ We will use .DW (define word) directive because our values are greather than one byte.

↳ So to access to data of prayer time we will use .DW.

→ These hex values are defined in minutes.



Macros:

Look-up Table:

```
.ORG 0x2000
PrayerTimes:
    .DW 0455, 1200, 1525, 1810, 1925 ; Monday
    .DW 0454, 1200, 1525, 1811, 1926 ; Tuesday
    .DW 0453, 1200, 1526, 1812, 1927 ; Wednesday
    .DW 0452, 1200, 1527, 1813, 1928 ; Thursday
    .DW 0451, 1159, 1528, 1814, 1929 ; Friday
    .DW 0450, 1159, 1529, 1815, 1930 ; Saturday
    .DW 0451, 1158, 1530, 1715, 1931 ; Sunday
```

Ask day to show its prayer times:

```
check_day2:
    CPI R22, '2'               ; Compare the value in R22 with the ASCII code for '2' (input for Tuesday).
    BRNE check_day3            ; If not equal, branch to check_day3 to check the next day.
    LDI ZL, LOW(PrayerTimes + 10) ; Load the low byte of the address for Tuesday's prayer times into ZL.
    LDI ZH, HIGH(PrayerTimes + 10) ; Load the high byte of the address for Tuesday's prayer times into ZH.
    JMP display_prayer_times    ; Jump to the routine that displays the prayer times.

check_day3:
    CPI R22, '3'               ; Compare the value in R22 with the ASCII code for '3' (input for Wednesday).
    BRNE check_day4            ; If not equal, branch to check_day4 to check the next day.
    LDI ZL, LOW(PrayerTimes + 20) ; Load the low byte of the address for Wednesday's prayer times into ZL.
    LDI ZH, HIGH(PrayerTimes + 20) ; Load the high byte of the address for Wednesday's prayer times into ZH.
    JMP display_prayer_times    ; Jump to the routine that displays the prayer times.
```

All days are checked using this logic

Display the relevant time:

```
; Subroutine to display a single prayer time in HH:MM format
display_time:
    ; Step 1: Convert and display the hour tens digit
    LDI R18, 0x30           ; Load ASCII offset for digit conversion
    MOV R19, R16            ; Copy the prayer time (HHMM) value into R19
    SWAP R19                ; Swap nibbles to get the high nibble (hour tens)
    ANDI R19, 0x0F          ; Mask out high nibble to isolate hour tens
    ADD R19, R18            ; Convert to ASCII by adding offset
    CALL DATAWRT            ; Write the hour tens digit to the LCD

    ; Step 2: Convert and display the hour units digit
    MOV R19, R16            ; Copy the prayer time (HHMM) value into R19 again
    ANDI R19, 0x0F          ; Mask out low nibble to isolate hour units
    ADD R19, R18            ; Convert to ASCII by adding offset
    CALL DATAWRT            ; Write the hour units digit to the LCD

    ; Step 3: Display the colon separator
    LDI R19, ':'            ; Load ASCII value for colon character
    CALL DATAWRT            ; Write the colon to the LCD

    ; Step 4: Convert and display the minute tens digit
    LSR R16                 ; Shift right to move minute tens to the lower nibble
    LSR R16                 ; (Repeat shifts to fully align the tens position)
    LSR R16
    LSR R16
    ANDI R16, 0x0F          ; Mask out high nibble to isolate minute tens
    ADD R16, R18            ; Convert to ASCII by adding offset
    CALL DATAWRT            ; Write the minute tens digit to the LCD

    ; Step 5: Convert and display the minute units digit
    MOV R19, R16            ; Copy the shifted value of R16 into R19
    ANDI R19, 0x0F          ; Mask out low nibble to isolate minute units
    ADD R19, R18            ; Convert to ASCII by adding offset
    CALL DATAWRT            ; Write the minute units digit to the LCD

    ; Step 6: Return from the subroutine
    RET                     ; End of subroutine
```

Sending Data & Command Macro:

```
CMNDWRT:
    OUT  LCD_DPRT, R16          ; Load the command (stored in R16) into the LCD data port
    CBI  LCD_CPRT, LCD_RS       ; Clear RS (Register Select) to indicate a command is being sent
    CBI  LCD_CPRT, LCD_RW       ; Clear RW (Read/Write) to set the LCD to write mode
    SBI  LCD_CPRT, LCD_EN       ; Set EN (Enable) high to latch the command into the LCD
    CALL SDELAY                 ; Short delay to allow the LCD to stabilize
    CBI  LCD_CPRT, LCD_EN       ; Clear EN (Enable) to complete the command write cycle
    CALL DELAY_100us            ; Add a delay of 100 microseconds to ensure the LCD processes the command
    RET

DATAWRT:
    OUT  LCD_DPRT, R16          ; Load the data (stored in R16) into the LCD data port
    SBI  LCD_CPRT, LCD_RS       ; Set RS (Register Select) high to indicate display data is being sent
    CBI  LCD_CPRT, LCD_RW       ; Clear RW (Read/Write) to set the LCD to write mode
    SBI  LCD_CPRT, LCD_EN       ; Set EN (Enable) high to latch the data into the LCD
    CALL SDELAY                 ; Short delay to allow the LCD to stabilize
    CBI  LCD_CPRT, LCD_EN       ; Clear EN (Enable) to complete the data write cycle
    CALL DELAY_100us            ; Add a delay of 100 microseconds to ensure the LCD processes the data
    RET
```

Next prayer time Macro:

```
next_prayer_time:
    ; Initialize counter for prayer time display duration
    ; The counter value determines how long the next prayer time is shown
    LDI R25, 3                  ; Set counter to 3 seconds (value in R25)

    ; Clear the LCD display to prepare for showing the next prayer time
    LDI R16, 0x01               ; Command to clear the LCD display
    CALL CMNDWRT                ; Send the command to the LCD
    CALL DELAY_2ms              ; Wait for 2 milliseconds to ensure the LCD processes the command
```

Conclusion:

Practical Applications

1. **Personal Use**: Individuals can use the system at home to track prayer timings and receive real-time countdowns for better time management.
2. **Mosques**: The scheduler can be installed in mosques to display accurate prayer timings and assist congregants in preparing for prayers.
3. **Educational Purposes**: The project demonstrates practical applications of microcontroller interfacing, making it an excellent tool for teaching real-time systems and embedded programming concepts.
4. **Commercial Use**: With minor modifications, the system can be adapted for commercial sale as a standalone or integrated feature in smart home systems.
5. **Community Centers**: Useful in community centers or workplaces with designated prayer areas to keep users informed of prayer times.
6. **Mobile and IoT Integration**: The project could be a foundation for developing mobile apps or IoT devices that synchronize with global prayer timing systems for added convenience.