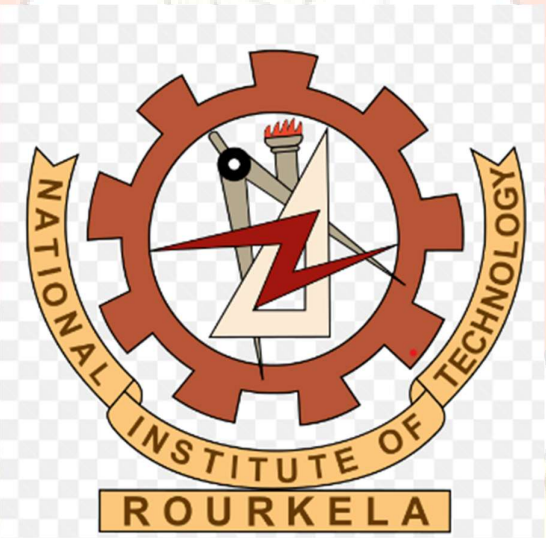
A RTOS SYSTEM TO MANAGE A USER QUEUE FOR SEQUENTIAL WASHING OPERATIONS,

WHERE COMPLETION SIGNALS ARE TRANSMITTED TO EACH USER'S ROOM VIA SERIAL COMMUNICATION, TRIGGERING THE NEXT USER IN LINE.

A Semester Project Report

From: 27th Oct, 2024 – 5th Dec,2024



for the course on

EMBEDDED SYSTEMS – EE3301

DEPARTMENT OF ELECTRICAL ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA

SUNDERGARH, ODISHA, INDIA

BY

TEAM MEMBERS

Ayush Kumar Sahu -122EE0837

Haimavatinandan Pati -122EE0607

Aniruddha Sen -122EE0612

UNDER THE ESTEEMED GUIDANCE OF

Prof. Supratim Gupta

Associate Professor

Department of Electrical Engineering, NIT Rourkela

.....INDEX.....

Contents *Page*

Chapter 1

Introduction

1.1 Introduction to the Problem Statement

In recent years, the market for washing machines has diversified, with both semi-automatic and automatic options becoming increasingly popular due to their enhanced features and user-friendly interfaces. However, in institutional settings, specifically in hostels accommodating over 500 students, the limited number of available washing machines poses significant challenges for accessibility and usage. This scarcity often results in disorder and frustration among students attempting to schedule their laundry. A critical concern in these hostels is the centralized placement of washing machines within a single room. This configuration inadvertently benefits students residing nearby, while those living farther away face unnecessary complications, such as needing to frequently check for machine availability. Many students find themselves returning only to discover that the machine they intended to use is already occupied. Additionally, the inconsistency of the timers on these machines can extend wash cycles, complicating the scheduling process for users.

1.2 Objective and Target Audience

To effectively address these challenges, our project aims to implement a robust solution grounded in a Real-Time Operating System (RTOS) specifically designed to optimize washing machine usage. This innovative system will allow up to five users to easily submit their room numbers, managing them in a First-In-First-Out (FIFO) queue. With timely notifications transmitted to each user's room via serial communication, they will be informed precisely when it is their turn to access the machine. These alerts will guide users on whether to bring their laundry to the machine or collect their cleaned items based on their queue position. In a bustling hostel environment, we anticipate that a maximum of five users will need access to the machines within any given half-hour period, but this number can be adjusted depending on the layout of the facilities and user demand. Additionally, our approach incorporates synchronization of the washing machine's operation with real-time indicators through sensors, ensuring accurate start and end times for each wash cycle. This data-driven method will eliminate reliance on potentially faulty timers, ensuring a seamless experience based on actual machine usage. By adopting this well-structured system for managing washing machine access, we aim to radically decrease competition for machine availability and significantly enhance accessibility for all students. Ultimately, our project

promises to revolutionize the efficiency of washing machine utilization in hostel environments, contributing to a smoother, more organized, and stress-free experience for students.

1.3 Components Required and Description:

8051 Microcontroller Kit: The Microcontroller kit which we have used is designed by MyTechnoCare.com. The benefits of using this kit is that it comes with the built in 8051 microcontroller (**AT89S52)** , coupled with the clock and ISP Programmer to program the IC. **The AT89S52** is a low-power, high-performance **CMOS 8-bit microcontroller** with 8K bytes of in-system programmable Flash memory. **The AT89S52 provides the following standard features: 8K bytes of Flash, 256 bytes of RAM, 32 I/O lines, Watchdog timer, two data pointers, three 16-bit timer/counters, a six-vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator, and clock circuitry**. In addition, **the AT89S52 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes**.

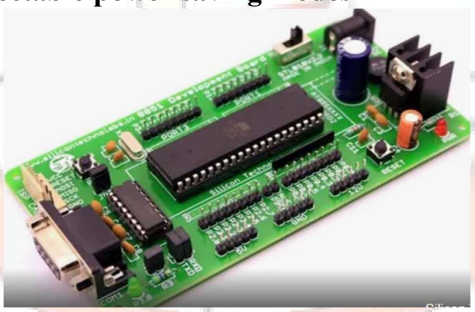


Fig 1.1: 8051 IC used in the Project.

Hall Sensor Module (MH3144): The benefit of **MH3144 module** can be operated in analog as well as digital mode allowing the developer the flexibility. The device includes a voltage regulator for operation with **supply voltages of 4.5 to 24 volts**, reverse battery protection diode, quadratic Hall-voltage generator, temperature compensation circuitry, small signal amplifier, Schmitt trigger, and **an open-collector output to sink up to 25 mA**. With suitable output pull up, **they can be used with bipolar or CMOS logic circuits**.

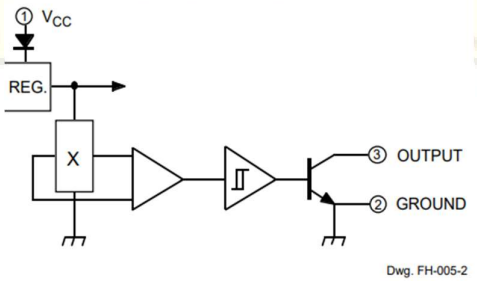


Fig 1.2: Functional Block Diagram of Hall Sensor

Components	Specifications	Model No.	Quantity	Price (₹)
8051 Microcontroller Development Board	*8051 Development Board With Onboard MAX232 & AT89S52 IC and ISP Programmer*	***AT89S52***	**1**	**851**
Power Supply Adapter	*9V/1Amps AC TO DC*	--------	**1**	**150**
LCD Display	*LCD 16×2 with Yellow/Green Backlight*	***LM016L***	**1**	**285**
Decoder	*3 X 8*	***74HC138***	**1**	**50**
Keypad	*4 x 4 Matrix*	--------	**1**	**120**
Push Buttons	------	--------	**2**	**6**
LED	------	--------	**8**	**24**
Hall Sensor	*Linear Magnetic Digital Hall Sensor* 2.3 -5.3 V 6mAmps	***MH 3144***	**1**	**40**
Nand Gates	*4- 2 input Nand gates*	***IC-LM7400***	**1**	**40**
Not Gates	*Hex Inverter*	***IC-LM7404***	**1**	**31**
And Gates	*4- 2 input And gates*	***IC-LM7408***	**1**	**40**
Resistor	*220 Ohms*	--------	**1**	**1**
Resistor & Potentiometer	*10 kOhms*	--------	**1 each**	**12**
Jumper Wires	*Male to Female & Female to Female*	--------	**1 each**	**140**
3D Printing Cost (Chassis)	--------	--------	**1**	**200**

Total Cost of all Components is ₹1990 approx.

1.6 Mechanical Design:

Figure below shows the mechanical design and dimensions of the Chassis used. The Design has been modelled in SolidWorks CAD software.

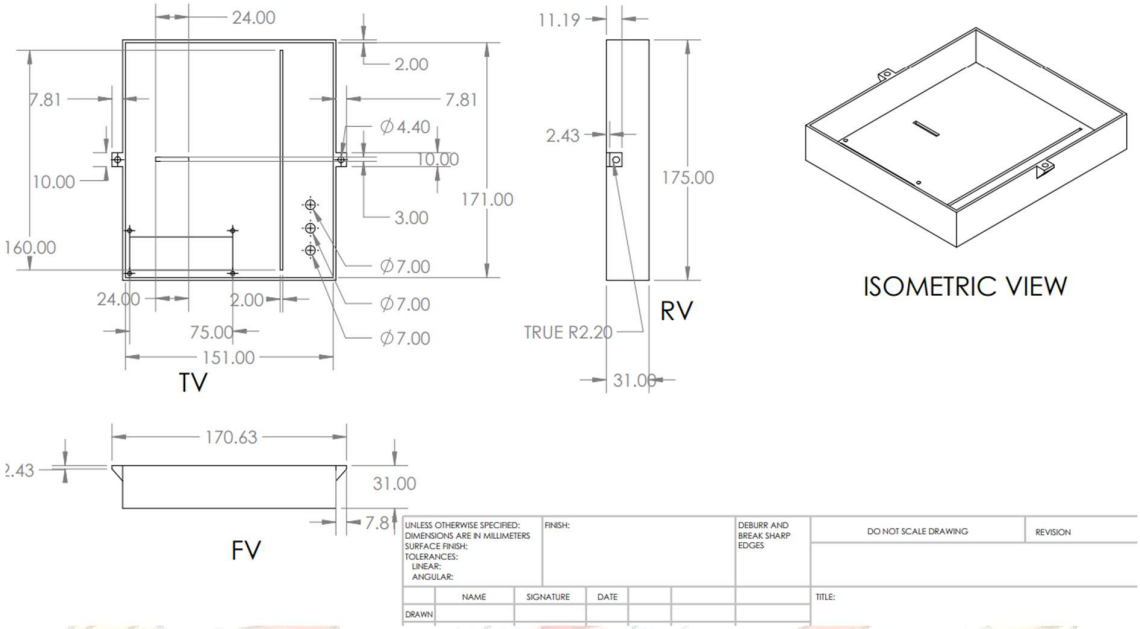


Fig 1.3: Design parameters of the upper base

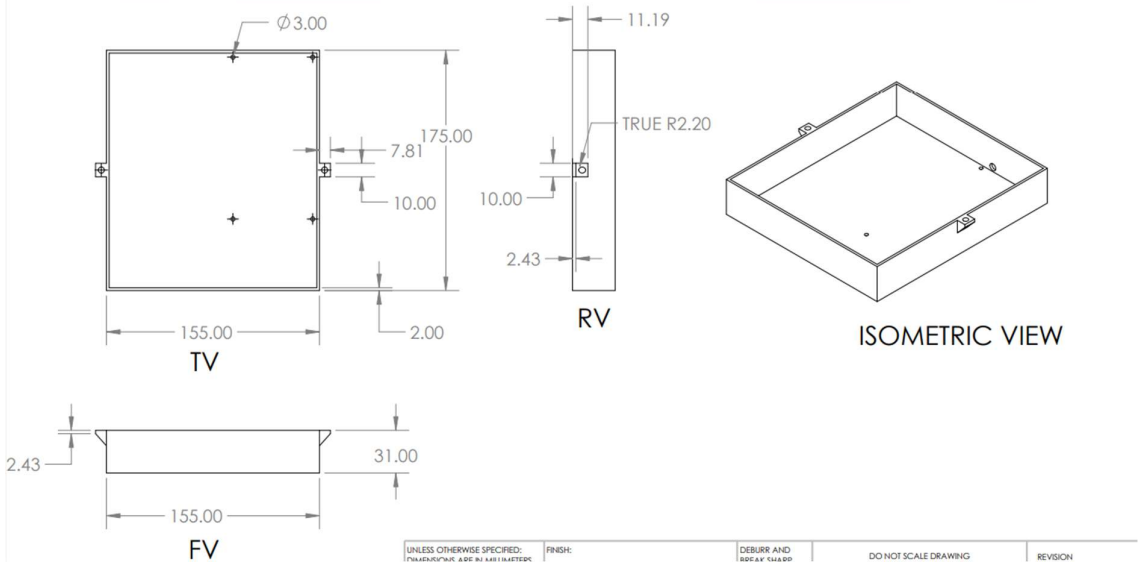


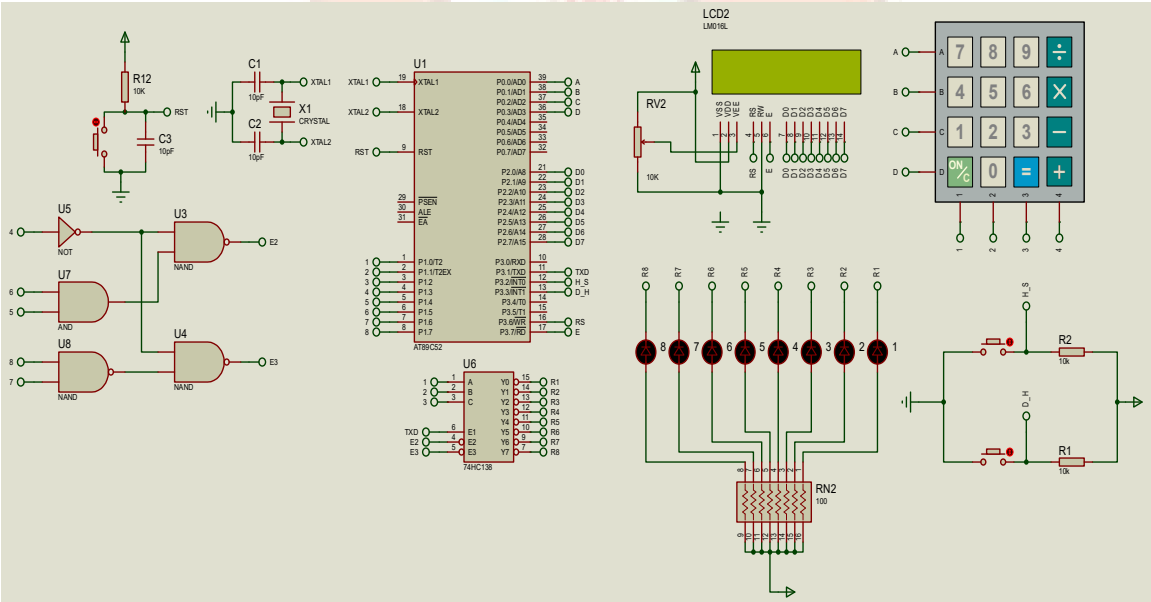
Fig 1.4: Design parameters of the lower base

Chapter 2

Circuit Diagram and Implementation

.....

2.1 Connections and Circuit Diagram



2.2 Procedure

LCD connections

In this configuration, the **D0-D7** pins of the LCD module are connected to the **P2.0-P2.7** ports of the 8051 microcontroller, allowing for data transmission required for display. The **RS** (Register Select) and **E** (Enable) pins of the LCD are wired to **P3.6** and **P3.7** on the microcontroller, respectively. These connections are essential for selecting the operating

mode of the LCD, determining if the input is a command or display data. The modes are controlled by the state of the **RS** pin:

- **Command Mode:** When **RS = 0**, the LCD interprets inputs as commands, enabling functions such as display clearing, cursor positioning, and display adjustment.
- **Data Mode:** When **RS = 1**, the LCD processes the inputs as data, displaying **ASCII** characters that correspond to the supplied binary values.

The Enable (**E**) pin is crucial for transferring data or commands to the LCD. A high-to-low pulse on this pin signals the LCD to read the input and carry out the specified command or display the data. By properly managing these signals, the microcontroller can effectively control both the functionality and the display output of the LCD.

- Lastly **VSS** is connected to ground terminal and the **VDD** is connected to the **5V** power supply. **VEE** terminal which is used to control the brightness of the LCD is connected to the variable terminal of a **10kΩ** potentiometer whose terminals are connected to power supply **5V** and ground.

Decoder connections

In this configuration the **A**, **B**, **C** terminals of the decoder **74HC138** are connected to the **P1.0**, **P1.1**, **P1.2** terminals of the microcontroller.

- We have selected port 1 as during serial communication the port 1 is free from any other work. The remaining pins of the port 1 of the microcontroller are connected to the active low pins of the decoder through NAND and NOR gates to produce output logic 1 when the output of pins **P1.3** to **P1.7** have output as **'01100'** respectively.
- This is to symbolise that when user enters address as 1 it is stored in ASCII as **31H** so the port **P1** will give output as **'10001100'** (LSB to MSB). Thus the combination of the remaining port pins are given to the decoder enable lines. At last the input to the active high enable pin is given from the **TXD** (**P3.1**) pin of 8051. The output from the decoder being active low is given to the anode of LEDs which are then connected to 5V through resistors.
- Thus, when the decoder is active and **'000'** is given to the **ABC** terminals then **Y0** will be **0** and only that LED will glow.

Sensor connections:

In this system two sensors are primarily used :

- **Hall Sensor:** To detect whether there is any flow of current or not in the power cable of the washing machine. The Hall sensor we have used is ***MH706,*** **which can act as an analog sensor or a digital sensor.** In our system we have used it in digital mode hence the connections are such that the ***D0*** pin of the Hall Sensor is connected to the **INT0 interrupt pin**, while **Vcc and Gnd pins are connected as usual.**
- **Contact Sensor:** To detect whether the current user has taken its laundry from the machine and the next user has put its laundry in the machine. This is done by attaching the contact sensor to the lead of the washing machine. In our system we have used a push button as an alternative to the contact sensor for economical purposes. The connections of the push button is done such that the one end of the push button is connected to **INT1 interrupt pin** , while **other end of the push button is grounded.**

Keypad connections

For keypad connections we have connected the row terminals **A**, **B**, **C**, **D** to the pins **P0.0**, **P0.1**, **P0.2**, **P0.3** respectively. Also we have connected the column terminals of the keypad **1**, **2**, **3**, **4** to pins **P0.0**, **P0.1**, **P0.2**, **P0.3** respectively.

2.3 Code Explanation

1. ***Definitions and Port Initialization***- This section defines the ports and assigned pins for specific components. The **RS** and **E** controls are responsible for executing commands on the LCD. The lines designated as **HALL_SENSOR_INT** and **PUSH_BUTTON_INT** are utilized for external interrupts linked to the hall sensor and push button, respectively. The port assignments are as follows: ***Port 1 (P1)*** is dedicated to managing the keypad columns, select lines, and enable pins for the decoder. ***Port 2 (P2)*** interfaces directly with the **LCD** display. ***Port 0 (P0)*** is responsible for controlling LCD command lines and processing keypad input rows. Lastly, ***Port 3 (P3)*** handles serial communication as well as external interrupts.

2. ***Variable Initialization-*** In this phase, registers are allocated for managing delays, storing pointers, and controlling the limits of the queue.

3. ***Reset and Interrupt Vectors-*** The main program commences at address ***0000h*** and employs interrupt vectors located at addresses ***0003h*** and ***0013h*** for handling ***INT0*** and ***INT1*** interrupts, respectively.

4. ***Main Initialization Routine-*** The initialization routine sets queue limits, stack pointers, and global interrupt settings. It also configures the LCD with specific settings pertaining to its display mode. Additionally, interrupts for ***INT0 (hall sensor)*** and ***INT1 (push button)*** are configured during this phase.

5. ***Main Loop (Loop Routine)-*** The main loop actively checks and updates the status of the queue, displaying pertinent information on the LCD. It includes a keypad input check whereby pressing the **'*'** key initiates an operation, subsequently decrementing the user count. In instances where the queue reaches capacity, the program transitions to a designated **QUEUE_FULL** subroutine.

6. ***Queue Full Subroutine-*** In this subroutine, a **"QUEUE FULL"** message is displayed on the LCD, and the system awaits user interaction. When a slot becomes available, the system removes the next user's address from the queue, increments the queue count, and then pauses until the washing machine cycle is completed.

- **INFLOOP routine:** This subroutine is one of the most crucial subroutine in the entire system, since it handles both the interrupts and decides which tasks is to be performed.
- It broadly performs two types of Tasks:
- **Task -1:** It checks **whether the user which is addressed is the last user in the queue or not**. If this is true, then no need to extract details for the next user since the queue is empty and **the program control must move to the main loop for fresh input. If the condition is false then the program should check the Task-2 of the *INFLOOP*.** This it achieves by using the ***0AH register* ,** to check for whether the user is the **last user** or not.
- **Task -2**: It checks whether two tasks of ***PUSH_BUTTON_ISR*** has been performed or not for every user address (except the last user of the queue). If two tasks are performed then the program control must move to the main loop for fresh input , **or else stay in the infinite loop**.

```
INFLOOP:                          ; WAIT FOR ISR INTERRUPT INT0 : Washing machine to complete.
;-----------------------------------;ISR0/ISR1 is called from here.
; Two tasks are performed :
;----------------------------------- ;Task 1 to check if the last user's laundry has been successfully washed or not
          CLR C
          MOV A ,#01H
          SUBB A,0AH              ; If value stored in 0AH is one then go back to the main loop (Last user has been addressed) else move on!!
          JZ LAST
;----------------------------------- ;Task2 : If the two tasks done by INT1 interrupt has been performed or not
          CLR C                   ;Clear Carry to avoid any wrong results from the following instruction
          MOV A, 08H
          CJNE A, #02H, INFLOOP   ; Check if two tasks has been performed or not
          JMP interJMP2

LAST:
          MOV 0AH, #00H
          JMP interJMP2           ;If queue is empty go to LOOP for user input
```

Fig 2.2: Snippet of the INFLOOP performing the above mentioned operations

7. ***Push and Pop Operations (Queue operations)-*** This section implements ***push (enqueue)*** and ***pop (dequeue)*** functions to effectively manage addresses stored in **RAM**, following a first-in, first-out **(FIFO)** principle. These operations manipulate the stack pointer to facilitate the storing and retrieval of user data.
    - In our system we have **maintained Circular Queue** by using the two pointers approach. ***R6 is used for bottom pointer and R2 is used for top pointer.***
    - As there are many routines which are called, this might affect the queue structure by causing potential data loss.
    - Thus, we came with the idea to divide the RAM into two major parts:
    - From the address ***2AH*** the RAM is used for general purposes like subroutine calling etc.; from ***20H to 2AH* ,** RAM is designated for maintaining the queue. Thus, when any sub routine is called the ***SP*** moves to the address above ***2AH*** and when any queue operations are performed the **SP** comes to the address range ***20H to 2AH.***

```
; Push : First increase the value of stack pointer and then Push the value
PushVal:
        CJNE R6,#2AH,ContPush
        MOV R6,#1FH          ; Queue Address is Exhausted . Start Over with Initial Value (Circular Queue)
ContPush:
        MOV R7,SP            ; Store the current value of the stack pointer in R7
        MOV SP , R6          ; Update the value of stack pointer to the top of the stack (first val: 1F is moved to stack pointer)
        Push Acc             ; Push the value of Accumulator into the stack (first value is stored in 20h)
        CLR A
        INC R6               ;Increment R6 to store the next top of the stack address.
        MOV SP, R7            ; update the stack pointer to the address before the Subroutine operations
        MOV R7,#00H
        RET

;Pop : First pop the value then decrement the stack pointer.
PopVal:  CJNE R2,#2BH,ContPop   ;Queue Address is exhausted . Start popping from 20h RAM Address
        MOV R2,#20H
ContPop:
        MOV R7,SP            ; Store the current value of the stack pointer in r7
        MOV SP , R2          ; Update the value of SP to the bottom of the stack:(First val is present at 20h)
        Pop Acc
        MOV R5, A            ; Store the result in R5
        CLR A
        INC R2               ;Increment R2 to store the next bottom of the stack address
        MOV SP,R7            ; update the stack pointer to the address before the subroutine operations.
        MOV R7,#00H
        RET
```

Fig 2.2: Snippet of the circular queue implementation code using 2 pointer approach.

8. ***Serial Communication Routine-*** . This sub routine performs the serial communication to the designated user. This is achieved by selecting a particular address to which we want to communicate using the ***3*8 decoder.*** Since the model we have used is pretty basic and does not encompasses long distance communications, therefore we have used only **the TXD pin** as a signal generator pin, which will make **5 blinks for the user** to take his/her laundry and **10 blinks for the next user** to come with his/her laundry. The number of blinks is denoted by the **17H** register.
    - However, we **do have included the code for serial communication using UART protocol** in **mode 1 operation mode**. This code could be used if we want to give some message to the user **that could be displayed in the LCD screen at the user's room.**
    - Making a wireless communication using Bluetooth module is difficult to achieve, since we need to send different messages to different address locations. This could make the project less economical.

```
SerialComm:
            MOV P1,R5                   ;Sending the address as select lines to the decoder
SerialMsg:
            CPL P3.1                    ; Toggle LED on P3.1 (ON/OFF)
            MOV R4,#0AH                 ;Have a delay of
            ACALL DelayProg             ; Short delay between blinks
            DJNZ 17H,SerialMsg
            CLR A                       ;Clear the Accumulator for further use
            CLR P3.1                    ;To turn of the P3.1 after toggling.
            RET
```

Fig 2.3: Snippet of the Serial communication

9. ***LCD Control Commands-*** The program encompasses subroutines designed for the transmission of ***commands (send_command)*** and ***data (send_data)*** to the LCD. A Delay subroutine is utilized to ensure proper timing is maintained during LCD operations.

10. ***Interrupt Service Routines (ISRs- The Hall Sensor Interrupt Service Routine (HALL_SENSOR_ISR)*** signals the completion of the washing machine cycle and updates the task counters accordingly. It awaits user confirmation to ensure that clothes have been collected.

- **INF_LOOP2 routine:** This routine which is present inside the ***HALL_SENSOR_ISR*** basically waits for the user to press the push button and trigger the **INT1** interrupt.
- When the interrupt has been triggered, it sets the ***IE1 bit***, which signifies that there is a pending interrupt. Upon detecting this change, the control of the program comes out of the ***HALL_SENSOR_ISR*** and move to the ***INFLOOP*** routine, from where the pending interrupt is addressed.

```
INF_LOOP2:
          JB IE1,ReturnInt1                ;If the second interrupt has been called then finish ISR0 and start ISR1.
          JMP INF_LOOP2

ReturnInt1:
           RETI                            ; Return from interrupt
```

Fig 2.3: Snippet of the INF_LOOP2 subroutine

Conversely, the Push Button Interrupt Service Routine **(PUSH_BUTTON_ISR)** determines the next user in line by evaluating stored data and notifying them when to commence their cycle.

- The **PUSH_BUTTON_ISR** performs two types of tasks each time when its called consecutively:
- **Task -1:** To extract the next user data from the queue and inform the user about his/her turn through serial communication. The extraction procedure is not performed for the last user in the queue, since after him/her there are no user data is present in the queue. Thus, the control directly moves to the main loop for fresh input. This is achieved by using a register ***0AH,*** which checks **if the current user is the last user in the queue or not**.
- **Task -2:** To wait for the confirmation that the next user has put its laundry in the washing machine, following which the control will move to the main loop to take new input values.
- The entire operation is observed by **an assigned flag register *08H,*** which **counts the number of tasks that are performed by the INT1 interrupt**

```
;TASK-1:-------------------------------------;1.)Extracting the next user address ; 2.) Perform Serial comm to that address.
;TASK-2:-------------------------------------;2.) Wait for the next user to put its clothes and move to main loop
PUSH_BUTTON_ISR:

          MOV A,08H                ;Move the task counter to Accumulator
          CLR C                    ;Clear Carry to avoid any wrong results
          SUBB A,#00H              ;If task counter is #00h :execute Task-1 and return.
          JZ TASK_1
          CLR C
          SUBB A,#01H              ;If task counter is #01h :execute Task-2 and return.
          JZ TASK_2
```

Fig 2.3: Snippet of how 2 tasks are performed when same ISR is called at different times.

11. ***Keypad Input Handling-*** This aspect of the program detects keypad input, identifies the respective row and column, and retrieves the **ASCII** value corresponding to the pressed key using lookup tables. The keypad has been code using standard procedures that is used to interface the keypad with the system.
    - If the **'E'** key, designated as the **Enter key**, is pressed, control returns to the main loop; otherwise, the program reverts to the keypad input routine.
    - If the **'C'** key, designated as the **Clear key ,** is pressed, then it allows the user to clear any data which the user might have mistakenly typed in the system.

```
clear:   CJNE A,#43H,Copy
         MOV P2, #01H                          ;Clear display screen
         ACALL send_command
         LJMP  Loop
```

Fig 2.4 Clear subroutine to clear any input that was typed mistakenly

12. ***Lookup Table for Keypad ASCII Values-*** A dedicated lookup table comprises **ASCII** mappings for each keypad row, facilitating the conversion of key presses into **ASCII** characters for further processing.

```
ORG 300H

KCODE0:DB  '1','2','3','A'    ;ROW 0
KCODE1:DB  '4','5','6','B'    ;ROW 1
KCODE2:DB  '7','8','9','C'    ;ROW 2
KCODE3:DB  '*','0','E','D'    ;ROW 3
```

Fig 2.5: Lookup table used for keypad

2.4 Flow of the Program:

In this section flow of the program is explained so that any developer can make necessary changes in the program to suit their purpose.

- The program flow starts from the microcontroller, which is seeking the address of the user. The microcontroller displays the **"ADDRESS"** in the LCD for user input.
- The user filled its address **effectively from 0 to 7 for this prototype**. The address entered by the user is converted to the ASCII value and pushed into a queue in RAM for subsequent operations.
- In this prototype the system can handle **a maximum of 5 users**. After 5 users have filled in their address, the microcontroller will display **"QUEUE FULL"** indicating that the queue has been filled to its maximum capacity and it can no longer take any input. This causes the microcontroller to disable the keypad to avoid any entering of input.
- ***Note:*** At the **time of starting the queue must be full** for the microcontroller to perform subsequent operations. This condition is necessary only during the start of the system and **not mandatory while running it**.
- Once a queue full has occurred, the program will enter into an infinite loop waiting for the washing machine to give an interrupt that it has finished washing the laundry.
- This **interrupt is detected by the digital Hall Sensor**, which detects the current flowing in the Power cable of the washing machine.
- After every cycle the current in the washing machine becomes minimum causing the Hall sensor to go low.
- The **Hall Sensor interrupt** is edge triggered causing the interrupt to trigger at falling edges only.

- Once the interrupt is triggered the program enters into the ISR, whereby it sends a beeping message to the current user to take its laundry via serial communication. In the prototype, this is done with **the LED blinking 5 times indicating the washing machine cycle has been completed** and the current user can come and take the laundry.
- After this microcontroller enters into a second infinite loop inside the ISR waiting for the next interrupt i.e waiting for the current user to come and take back its laundry and make the machine free to use.
- When the current user comes to take the laundry back, the user must open the washing machine lead and close it . This causes a contact sensor at the window of the washing machine to trigger and send an interrupt signal to the microcontroller. This also cause the program counter to come out of the first ISR so as to execute the next ISR.
- This action makes the **IE1** of **TMOD** register in the microcontroller as 1, indicating that there is an interrupt to entertain. When the program counter on returning to the first infinite loop sees a pending interrupt, it quickly jumps to the **PUSH_BUTTON_ISR**.
- In the **PUSH_BUTTON_ISR**, the program performs **two types of tasks** each at different times when it is called.
- **The Task-1** is performed when the current user has triggered the contact sensor (by opening and closing the lead of the washing machine -edge triggering) indicating that **the current user has taken its laundry**. This initiates the microcontroller to **"extract**" the next user address from the queue and through serial communication inform him that its his turn to wash his laundry.
- After this action the microcontroller is in wait state for the next user to bring its laundry.
- **The Task -2** is initiated when **the next user bring its laundry** it will again trigger the contact sensor (by opening and closing the lead of the washing machine – edge triggering) indicating that **the next user has put his laundry in the machine.**
- After this action, **the microcontroller returns to the main program and perform its normal routine**. And this cycle repeats.

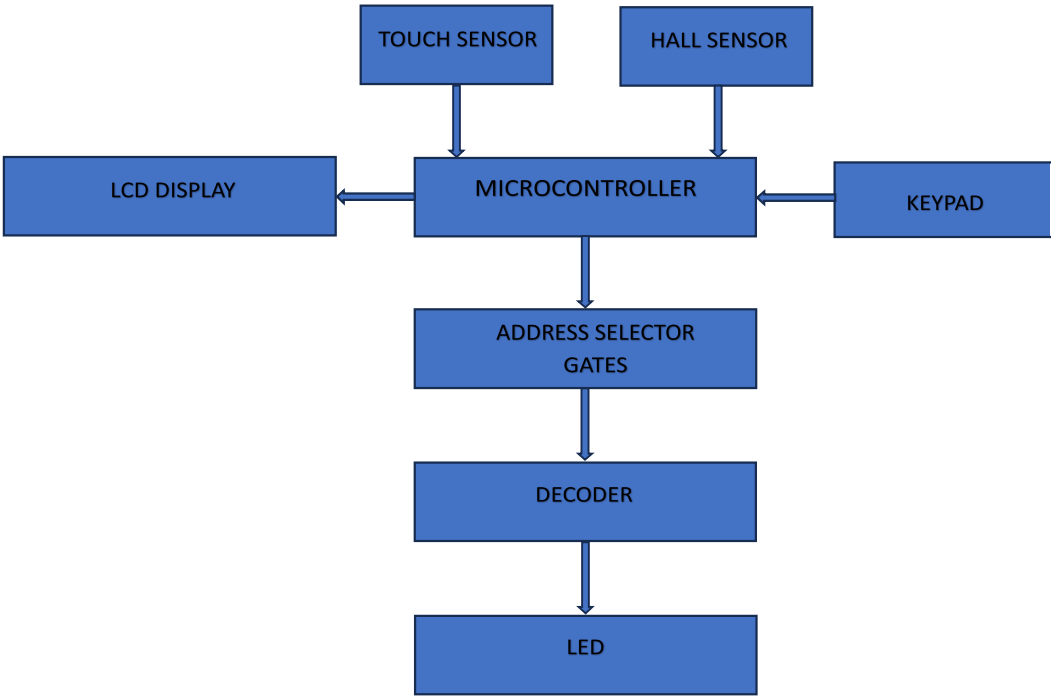
2.5 User Manual:

In this section the user interaction with the system has been explained, so that the user using this system has no issues in interacting with the system.

- Upon turning on the system, the system will ask for the user address by displaying **"ADDRESS"** in the LCD. The address the user needs to give should be a single digit from **0 to 7 only for this system**.
- Once the address has been typed by the user, the user should press the **'#' – enter** in the keypad indicating that the address has been **rightly saved in the system**.
- **Note**: In case the user has typed a wrong address before pressing enter, then he/she can **press 'C' – clear** in the keypad **to clear the input** and the user can enter a fresh input again.

- After the address has been saved in the system, the user will be notified to **bring his/her laundry** through serial communication **by flickering a LED 1 times** ,and **U_2_C (User 2 come)** will be displayed in the LCD.
- When the user reaches to the washing machine with his/her laundry, the user needs to open the lead of the washing machine – put his/her laundry – then close the lead of the washing machine. Then the user can give its input and can go back to his room.
- When the washing machine has completed its cycle, it will inform the user to come and **take back his/her laundry**. This will be notified via serial communication by **flickering the LED 5 times** and **U_1_F (User 1 Fetch)** is displayed on the screen.
- The user must go and take the laundry back and **make sure that the lead of the washing machine should be closed.**

2.6 Flowchart:



Chapter 3

Results

3.1 Simulation:

Below is the image of the simulation done in Proteus at a particular instant, when user1 has finished with the laundry.

Note : The address of user1 here is given as '2'.

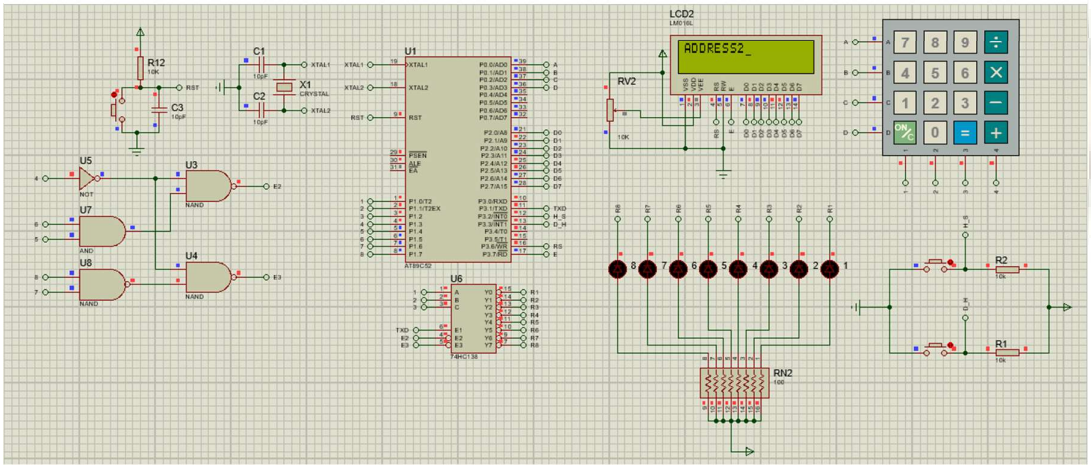


Fig 3.1

When user 1 has completed his laundry, the corresponding led blinks:

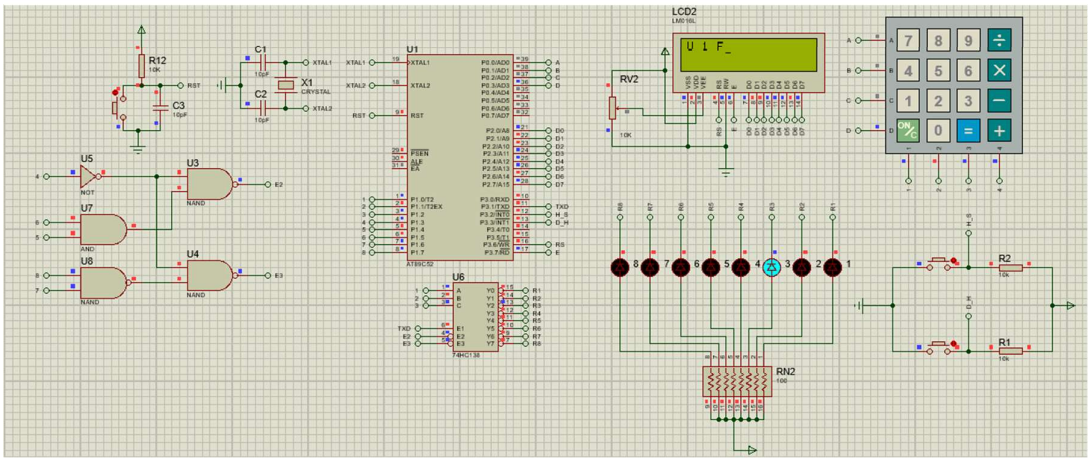


Fig 3.2

3.2 Hardware Demo:

Below is the one instant of hardware demonstration after when 5 users have given their inputs and the Queue has been exhausted. (User sequence – 0, 4,6,7,2)
After several tests we have concluded that the system is 95.7 % accurate with 3 errors occurring for every 70 iterations. The errors primarily occur at the beginning but is resolved by resetting the system.

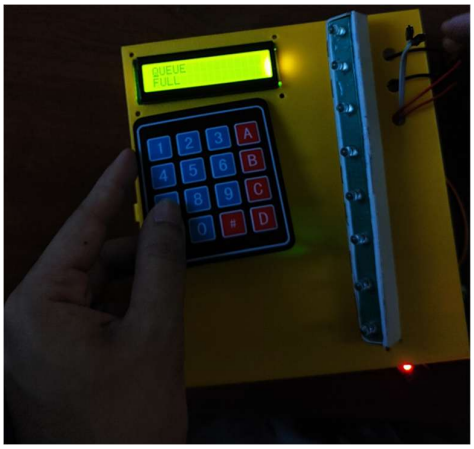


Fig 3.3: Condition of Queue Full (after 5 users have been taken)

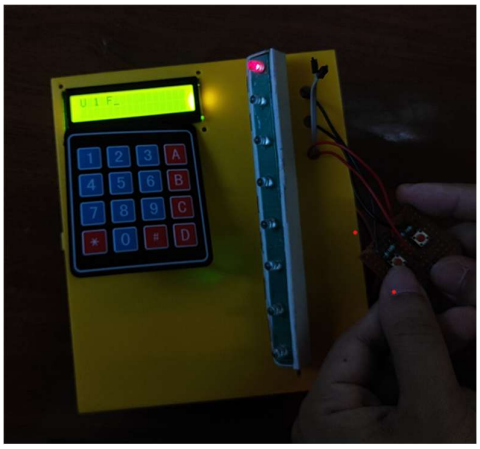


Fig 3.4: Telling user 1 to take its laundry (5 blinks), after Hall sensor interrupt ---U_1_F is displayed.

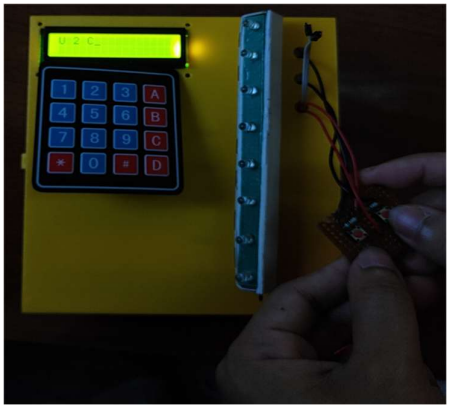


Fig 3.5: Asking User 2 to come with its laundry -1 blink , after contact sensor interrupt ----U 2 C is displayed

Fig 3.6: Next iteration: User 4 has completed its laundry and is informed (5 blinks) ---- hall sensor interrupt ---- U_1_F is displayed

Chapter 4

Conclusion and Future Work

This project successfully demonstrates the implementation of a Real-Time Operating System (**RTOS**) to manage a user queue for sequential washing operations. While this solution provides a basic approach to the problem, it lays a strong foundation for further automation and functionality enhancements, which could significantly increase the system's versatility and establish it as a valuable feature in future washing machines.

Our current system supports a ***maximum of five users per cycle***; however, with minor program modifications and the integration of ***external RAM***, we can ***expand the capacity for user data storage***. Additionally, the current design suffers from data loss upon system shutdown. To address this issue, user data could ***be backed up to a cloud-based storage solution***, ensuring data integrity and resilience against power interruptions. Enabling this feature would require ***Wi-Fi connectivity***, which can be achieved using a cost-effective microcontroller like the ***NodeMCU***. With Wi-Fi access, data communication could be enhanced further through internet-based serial communication or ***GSM technology***, broadening the scope for ***remote management and control***.

Furthermore, we could implement ***a password system*** that sends a passkey to the user, allowing the washing machine to start only when the correct passkey is entered. This feature would help prevent users from interrupting the queue and cause any redundancy, which is currently a flaw in our system.

We can also utilize ***a smart cable*** to optimize and enhance interrupts, enabling us to measure the current, voltage, and total energy consumption at any given moment.

Additionally, integrating ***a separate timer*** into the washing machine would allow us to monitor whether the washing machine's timer is running on time. This would ensure that the functionality being implemented can be communicated to the user effectively.

Chapter 5

References

Below are the references we have referred to complete this project.

- https://josephscollege.ac.in/lms/Uploads/pdf/material/SERIAL-COMMUNICATION.pdfhttps://josephscollege.ac.in/lms/Uploads/pdf/material/SERIAL-COMMUNICATION.pdf
- https://ebooks.inflibnet.ac.in/csp13/chapter/serial-port-programming-in-assembly/#:~:text=Serial%20communications%20of%20the%208051%20is%20established%20with%20PC%20through,with%20the%20help%20of%20Timerhttps://ebooks.inflibnet.ac.in/csp13/chapter/serial-port-programming-in-assembly/#:~:text=Serial%20communications%20of%20the%208051%20is%20established%20with%20PC%20through,with%20the%20help%20of%20Timer
- https://www.youtube.com/watch?v=YCYzMhEjoeY
- https://www.brainkart.com/article/Interfacing-and-Program-for-Keyboard-to-8051-Microcontroller_7838/
- https://www.youtube.com/watch?v=YlLMLLqPV0U
- https://skill-lync.com/student-projects/project-1-implement-a-fully-functional-queue-in-c-language-using-linked-lists-10