MCTE 4105
MECHATRONICS LAB III
SEM I || SESSION 20/21

TITLE: AUTOMATIC FISH FEEDERS

SDG CHOSEN: 14

SEC 2 || GROUP 12

NO	NAMES	MATRIC NO.
1.	Nurul Alya Sofiya Binti Shuhaimi	1719480
2.	Muhammad Izzat Bin Mohd Isa	1717987
3.	Syed Muhammad Aslam bin Syed Mohd Azizi	1719977

TABLE OF CONTENTS

1.0 INTRODUCTION

Pet is man's best friend. A variety of species have been adopted as a pet. This also includes fishes. In addition, 14.4% of Malaysian citizens have fish as their pet. As a living thing, pets also need to be fed in order for them to grow and survive. It is a basic necessity of any living thing on this planet. These pet fishes need to be fed once or twice a day, depending on the type of fish. Not to mention, most fishes can only starve for up to 2 or 3 days. They will eventually die if they are not given any food for the subsequent days.

Another essential aspect is pet fish can only live inside the aquarium or the container it is being kept in. Hence, it is totally dependent towards its owner for food. The downside of this is it can somehow restrict the movement of the owner. He or she cannot leave the fishes for more than three days. Otherwise, they will die. Furthermore, these owners can sometimes forget to feed their fishes, or even worse, and they overfed the fish, which can contaminate the water and lead to its death.

Simply being said, the conventional way of feeding the fish is very inefficient and tedious. Thus, we have decided to do an Automatic Fish Food Feeder project whose primary purpose is to assist fish pet owners in feeding their pets. Supposedly, this system can provide food for a chosen time interval, whether once, twice, or thrice a day. Moreover, this system should also be able to detect the availability of the food inside the container and alert the user whenever the food has run out. This project is inherent to do the 14th goal of the Sustainable Development Goals, which is Life Below Water.

1.1 PROBLEM STATEMENTS

1. **The fish owner cannot leave their pet for more than three days**

   Generally, fishes can only last up to 2 or 3 days without food, depending on the type of fish and their age. Hence, the owners cannot leave their fishes unattended for more than the mentioned days. In other words, the owner's movement is restricted due to his or her responsibility to feed the fish.

2. **Fish owners forget to feed the fish**

   It is inevitable that the owners can also forget to feed their fish due to their busy life. It is human to forget, but unfortunately, it can cost the fishes' life. This is because the fishes are totally dependent on their owner to provide food as they can only travel as far as the aquarium volume allows them.

3. **Fish owners overfeed the fish**

   As terrible as not feeding the fish, overfeeding the fishes can also harm them. Excessive food can contaminate the water, which is preventing the fishes from getting the necessary oxygen to breathe. Thus, the fish owners would have to repeat the tedious process of replacing the water in the tanks more often than usual. Worst case scenario, the fishes can also die from overeating

1.2 OBJECTIVES

The main objective of this project is to produce an Automatic Fish Feeder for pet fish at home whereby the owner of the pet fish (as target group) and the specific objectives can be described as the following: -

i. To design and develop automatic fish feeder for aquarium.
ii. To monitor the level of fish's food.
iii. To evaluate the performance of the developed mechanism.

2.0 WORKING PRINCIPLE

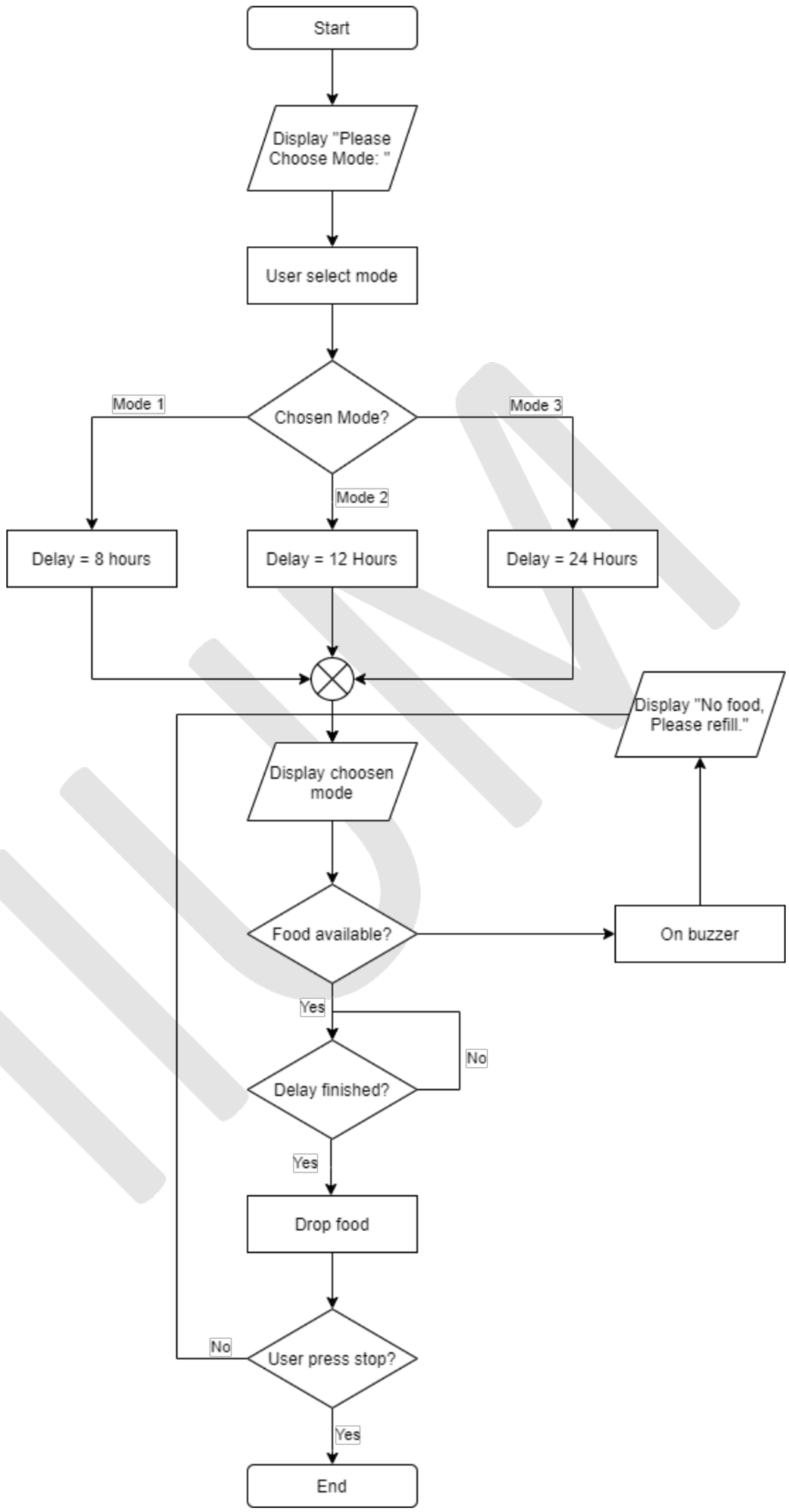


Figure 1 Flowchart of Automatic Fish Feeder

Based on Figure 1 above, the system displays "Please Choose Mode" on the Arduino LCD (I2C). Then, the user needs to select a mode to feed the fish by pushing one of the pushbuttons in the circuit. There are three types of modes which are Mode 1, Mode 2, and Mode 3.

Each of the modes has a different delay time to feed the fish. Mode 1 will delay 8 hours to feed the fish, where the system feed the fish once per day. Mode 2 will delay 12 hours to feed the fish, where the system feed the fish twice per day. Mode 3 will delay 24 hours to feed the fish, where the system feeds the fish thrice per day.

After choosing the mode, the Arduino LCD (I2C) will display the "Chosen Mode, and the 7 segment also will display the number of chosen modes. As an example, the user pushbutton for Mode 1. The Arduino LCD (I2C) will display "Mode: 1," and at the same time, the 7 segments will produce "1".

Next, if the fish food in the container is available where it is detected by the IR sensor, the motor will move and eventually drop the food into the fish aquarium. However, if there is no fish food detected by the IR sensor, then the buzzer will turn on and produce an alarm sound in order to alert the user. At the same time, the LCD will display "No food, please refill".

In order for the user to change other types of modes, he/she needs to reset the circuit by turn on the reset switch. For example, the user has wrongly pushed the mode from the system whereby he/she wanted to choose Mode 2 but mistakenly pressed Mode 1. Therefore, he/she can reset the mode by turn on the reset switch. This reset switch also acts as a safety measure when any unwanted events happened, or haywire occurred at those moments.

2.1 DIGITAL LOGIC CIRCUIT

In this section, we discussed regarding digital logic circuit that is used in the Automatic Fish Feeder system. There are four inputs and three outputs. The first (Input 1), second (Input 2), and third input (Input 3) represent pushbutton 1, pushbutton 2, and pushbutton 3, respectively, to choose the mode of feeding. As for the fourth input (Input 4) indicates the IR sensor to detect whether the fish food is available or not. Then, the first output is the motor to open and close the fish feeder's lid. The second output is a buzzer to notify the user that the system has run out of fish food. The third output is seven

segments to indicate the user either the mode chosen. All of these inputs and outputs' digital logic circuit can be seen in Figure 2.

2.2 LOGIC GATE AND TRUTH TABLE

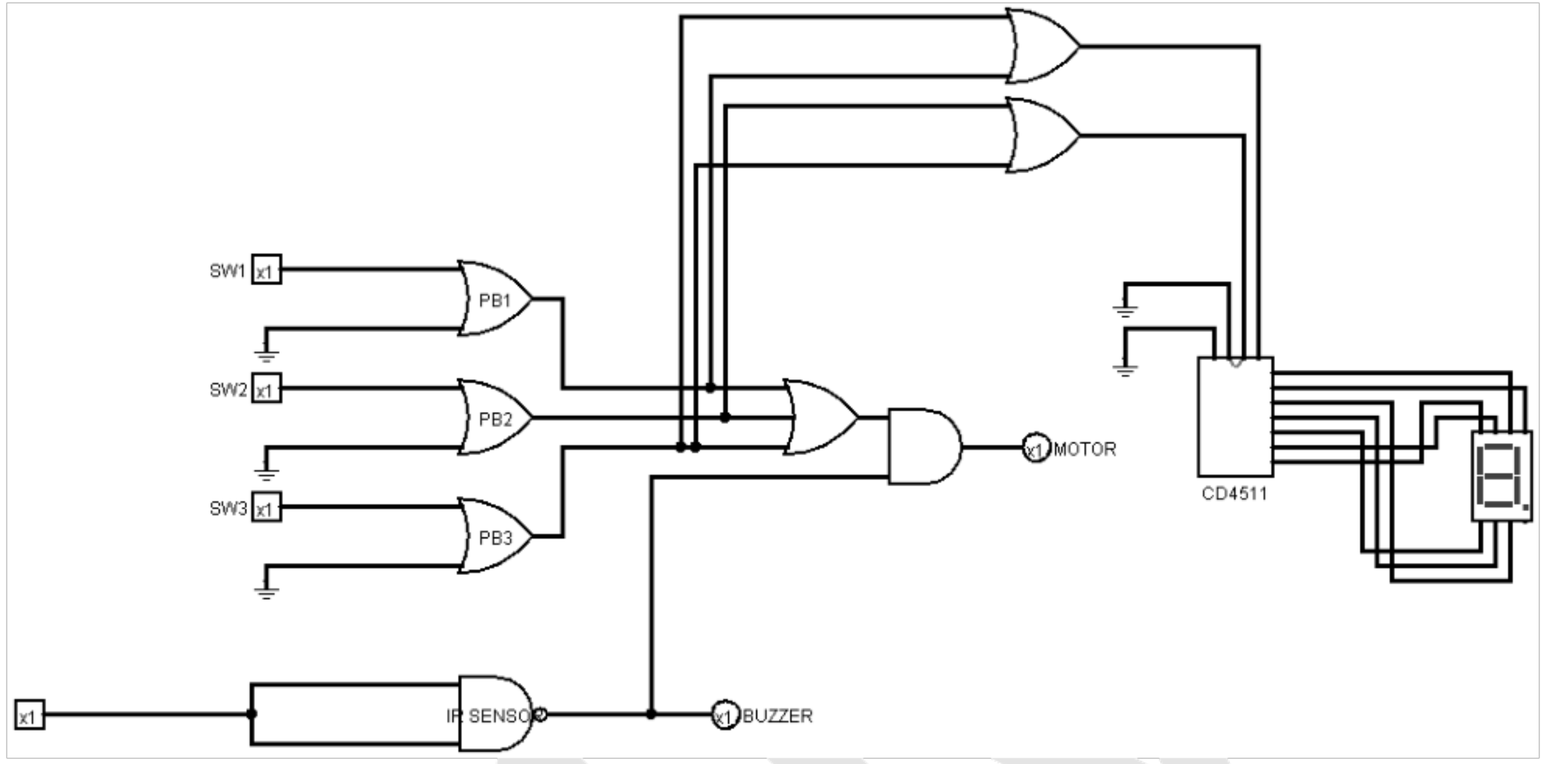


Figure 2 Digital logic circuit of automatic fish feed system

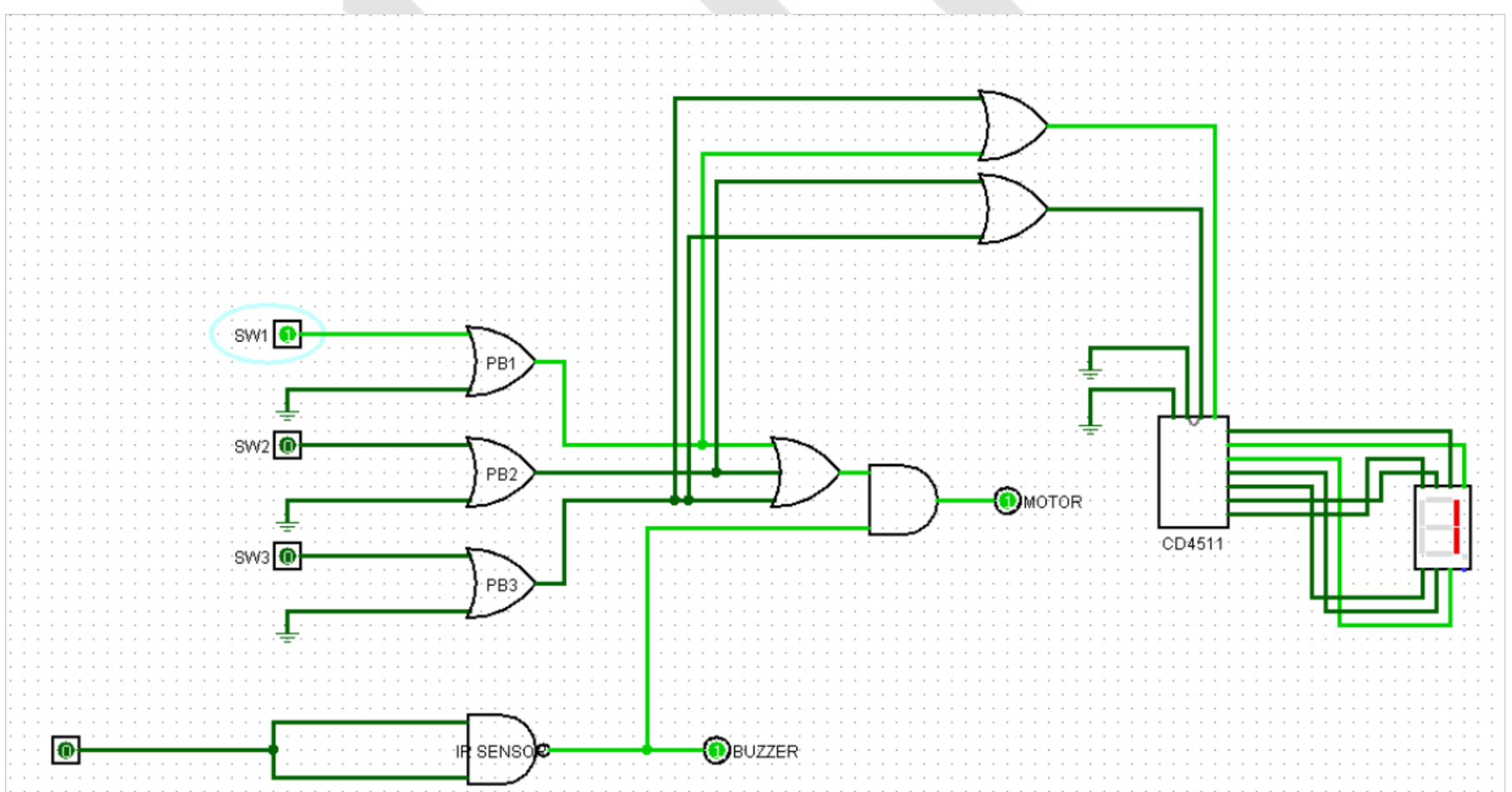


Figure 3 The 7-segment displayed Mode "1" when PB 1 is high

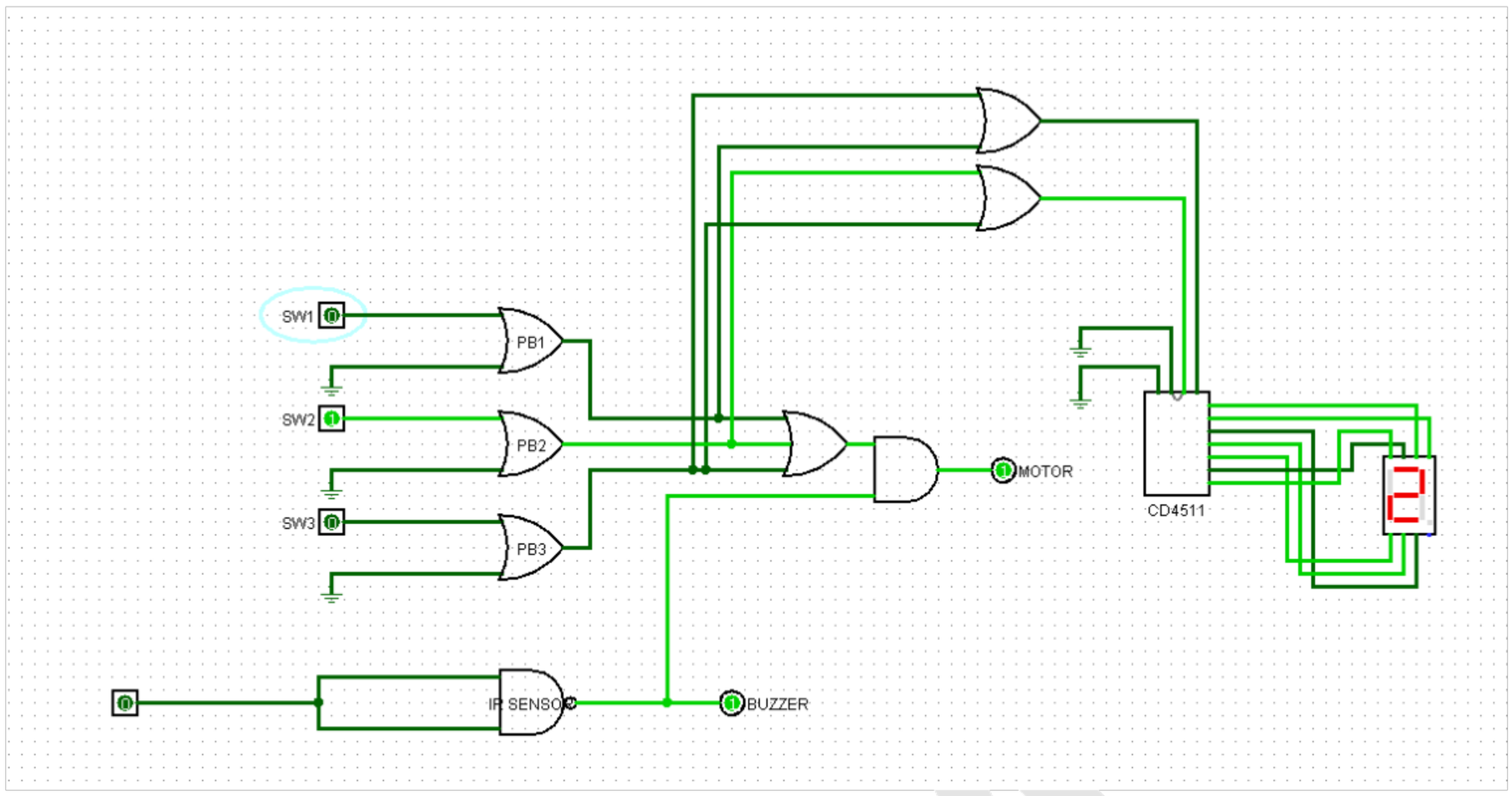


Figure 4 The 7-segment displayed Mode "2" when PB 2 is high

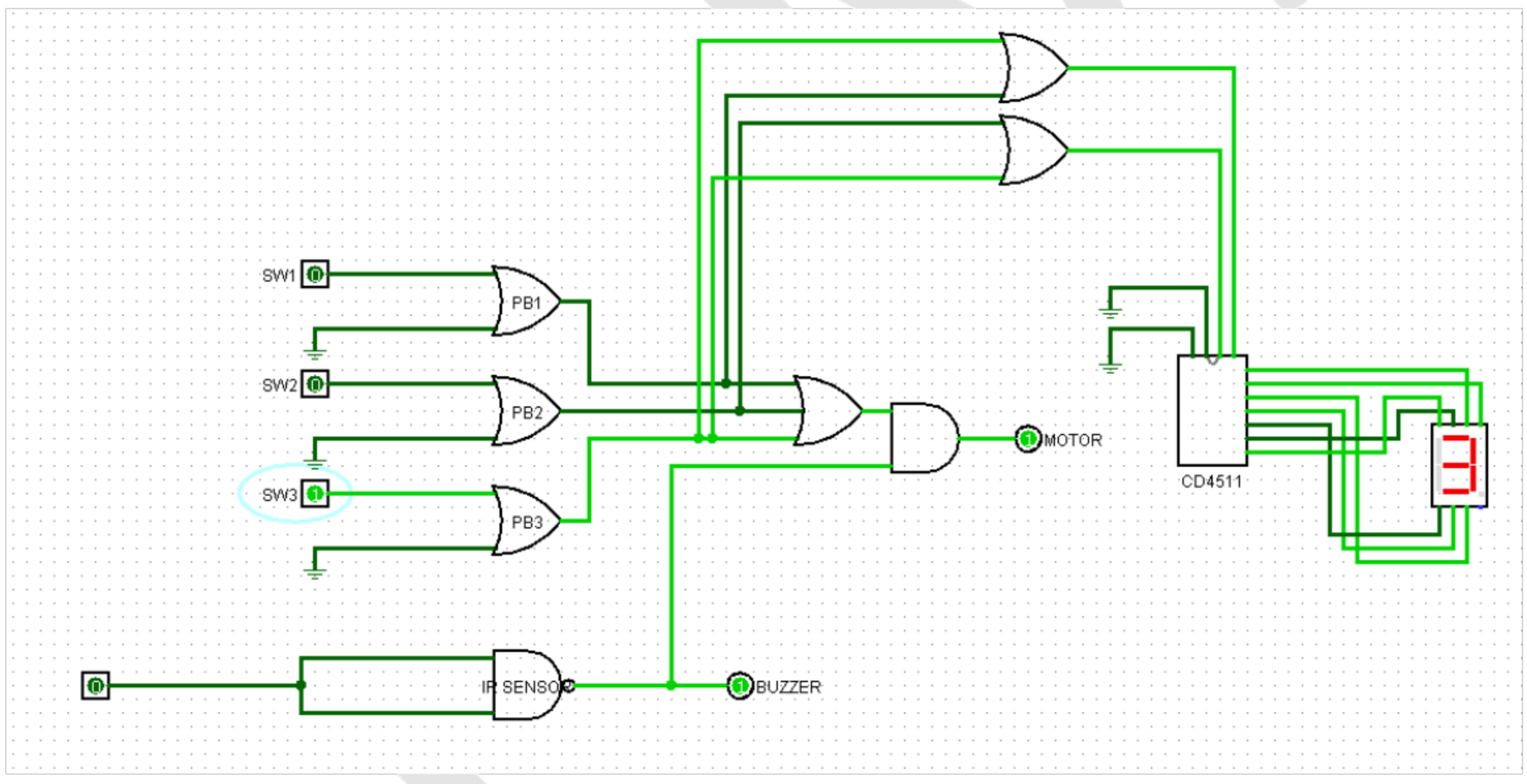


Figure 5 The 7-segment displayed Mode "3" when PB 3 is high

SW1	SW2	SW3	a	MOTOR	BUZZER
0	0	0	0	0	1
0	0	0	1	0	0
0	0	1	0	1	1
0	0	1	1	0	0
0	1	0	0	1	1
0	1	0	1	0	0
0	1	1	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	0	1	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	1	0	1	1
1	1	1	1	0	0

Figure 6 Truth table of automatic fish feeder

Where,

SW1 = Input 1

SW2 = Input 2

SW3 = Input 3

MOTOR = Output 1

BUZER = Output 2

s	BCD	G	F	E	D	C	B	A
0	0000	0	1	1	1	1	1	1
1	0001	0	0	0	0	1	1	0
2	0010	1	0	1	1	0	1	1
3	0011	1	0	0	1	1	1	1
4	0100	1	1	0	0	1	1	0
5	0101	1	1	0	1	1	0	1
6	0110	1	1	1	1	1	0	1
7	0111	0	0	0	0	1	1	1
8	1000	1	1	1	1	1	1	1
9	1001	1	1	0	1	1	1	1

Figure 7 Truth table of 7 segment

2.3 MICROPORCESSOR

Codes with description

```
.def   dcount  =r6
.def   dcount2 =r7
.def   dcount3 =r8
.def   dcount4 =r9
;
.def   temp          = r16          ;
.def   i2cdata       = r17              ; I2C data transfer register
.def   i2cstat       = r19          ; I2C bus status register
.def   i2cdelay      = r20          ; Delay loop variable
.def   LCD_Reg       = r21          ;
.def   count  = r22                ;

.equ i2cadr = $4E                    ; set the address of the LCD
;
.equ     FirstLine_Add =0x80        ;Base address of first row of the LCD
.equ     SecondLine_Add=0xC0        ;Base address of second row of the LCD
;

.equ i2cwr = 0                             ; Write Bit
.equ i2crd = 1                             ; Read Bit (not used)
;
;Port C
.equ   I2CPORT       = PORTC       ;*Set to correspond to I/O Port used for I2C
.equ   I2CDDR        = DDRC        ;*Set to correspond to I/O Port used for I2C
.equ   I2CPIN        = PINC     ;*Set to correspond to I/O Port used for I2C
;
;
.equ   SDAP = 4
.equ   SCLP = 5
;
.equ   LCD_RS = 0                  ;low to send command, high to send data
.equ   LCD_RW = 1                  ;low = write, high = read
.equ   LCD_E = 2                    ;high (idle) high to low to send byte

;-----------------------------------------------
.cseg
.org 0
;

main:

       ldi r27, low(RAMEND)    ; set up the stack
       out SPL, r27
       ldi r27, high(RAMEND)
       out SPH, r27


       ;setting up I/O
       ldi r28, 0xFF
       ldi r29, 0x00
       out DDRD, r28        ;set PORTD of Arduino as output
       out DDRB, r29        ;set PORTB of Arduino as input

       rcall  i2c_init      ;Initialize I2C bus
       rcall  LCD_init

       call welcome         ;call subroutine to display welcome message
```

```
        start:

        sbic PINB, 3          ;check for IR Sensor condition, if low, skip next line
        call buzzer1

        sbic PINB, 0          ;check for pushbutton 1 condition, if low, skip next line
        call mode_1           ;call subroutine mode_1 if pushbutton 1 was pressed

        sbic PINB, 1          ;check for pushbutton 2 condition, if low, skip next line
        call mode_2           ;call subroutine mode_2 if pushbutton 2 was pressed

        sbic PINB, 2          ;check for pushbutton 3 condition, if low, skip next line
        call mode_3           ;call subroutine mode_3 if pushbutton 3 was pressed

        nop;

jmp start                   ;jump back to start

;subroutine buzzer1
buzzer1:

        call disp_error       ;display error message

        .cseg
        ldi r25, 0b00100011
        out tccr0a, r25 ;need out!
    ldi r25, 0b00001100
        out tccr0b, r25 ;need out!
        ldi r25, 141
        out ocr0a, r25
        ldi r25, 71
    out ocr0b, r25
        call DELAY_05
        call disp_clear         ;call subroutine to clear display
    sbi PORTD, PIND2        ;set buzzer on
        call DELAY_05
        cbi PORTD,PIND2         ;turn off buzzer
jmp main                    ;jump back to main

;subroutine buzzer2
buzzer2:
        call disp_error         ;display error message

        .cseg
        ldi r25, 0b00100011
        out tccr0a, r25 ;need out!
    ldi r25, 0b00001100
        out tccr0b, r25 ;need out!
        ldi r25, 141
        out ocr0a, r25
        ldi r25, 71
    out ocr0b, r25
    sbi PORTD, PIND2        ;set buzzer on
        call DELAY_05
        cbi PORTD,PIND2         ;turn off  buzzer
jmp mode_1                  ;jump back to mode_1

;subroutine buzzer3
buzzer3:
        call disp_error         ;call subroutine to display error

        .cseg
```

```
       ldi r25, 0b00100011
       out tccr0a, r25 ;need out!
    ldi r25, 0b00001100
       out tccr0b, r25 ;need out!
       ldi r25, 141
       out ocr0a, r25
       ldi r25, 71
    out ocr0b, r25
    sbi PORTD, PIND2       ;set buzzer on
       call DELAY_05
       cbi PORTD,PIND2        ;turn off buzzer
jmp mode_2                 ;jump back to mode_1

;subroutine buzzer4
buzzer4:
       call disp_error        ;call subroutine to display error

       .cseg
       ldi r25, 0b00100011
       out tccr0a, r25 ;need out!
    ldi r25, 0b00001100
       out tccr0b, r25 ;need out!
       ldi r25, 141
       out ocr0a, r25
       ldi r25, 71
    out ocr0b, r25
    sbi PORTD, PIND2       ;set buzzer on
       call DELAY_05
       cbi PORTD,PIND2        ;turn off buzzer
jmp mode_3                 ;jump back to mode_3

;subroutine welcome
Welcome:

    ldi       LCD_Reg,FirstLine_Add       ;cursor to 1st line, first position
       rcall  LCD_command                 ;should already be there from initialize

       ldi    ZH,HIGH(welcome1*2)  ;Point to welcome1 subroutine in LCD.INC
       ldi    ZL,LOW(welcome1*2)
       ldi    count,0x10                  ;max 16 chars per line
       rcall  WrLCD                       ;Transmit message

       ldi    LCD_Reg,SecondLine_Add      ;cursor to 2nd line, first position
       rcall  LCD_command                 ;

       ldi    ZH,HIGH(welcome2*2)  ;Point to welcome2 subroutine in LCD.INC
       ldi    ZL,LOW(welcome2*2)
       ldi    count,0x10                  ;max 16 chars per line
       rcall  WrLCD                       ;Transmit message

ret

mode_1:
       call disp_mode1              ;call subroutine to display mode 1
       call disp_1                  ;call subroutine to display 1
       call DELAY_05
       call DELAY_05
       call DELAY_05
       call DELAY_05
       call DELAY_05
       call DELAY_05
       call DELAY_05
```

```
        call DELAY_05
        call DELAY_05
        call DELAY_05
        call DELAY_05
        call DELAY_05
        sbic PINB, 3                ;check for IR sensor condition, if low, skip next line
        call buzzer2                ;call subroutine buzzer2
        sbi PORTD, PIND3            ;open feeder
        call DELAY_05
        cbi PORTD, PIND3            ;close back feeder
jmp mode_1                        ;jump back to initial of mode_1

;subroutine mode_2
mode_2:
    call disp_mode2               ;call subroutine to display mode 2
       call disp_2                  ;call subroutine to display 2
       call DELAY_05
       call DELAY_05
       call DELAY_05
       call DELAY_05
       call DELAY_05
       call DELAY_05
       sbic PINB, 3                 ;check for IR sensor condition, if low, skip next line
       call buzzer3                 ;call subroutine buzzer3
       sbi PORTD, PIND3             ;open feeder
       call DELAY_05
       cbi PORTD, PIND3             ;close back feeder
jmp mode_2                        ;jump back to initial of mode_2

;subroutine mode_3
mode_3:
       call disp_mode3              ;call subroutine to display mode 3
       call disp_3                  ;call subroutine to display 3
       call DELAY_05
       call DELAY_05
       call DELAY_05
       call DELAY_05
       sbic PINB, 3                 ;check for IR sensor condition, if low, skip next line
       call buzzer4                 ;call subroutine buzzer4
       sbi PORTD, PIND3             ;open feeder
       call DELAY_05
       cbi PORTD, PIND3             ;close back feeder
jmp mode_3                        ;jump back initial of mode_3

;subroutine to clear display
disp_clear:

       ldi    LCD_Reg,FirstLine_Add       ;cursor to 1st line, first position
       rcall  LCD_command                 ;should already be there from initialize

       ldi    ZH,HIGH(clear*2)          ;Point to clear subroutine in LCD.INC
       ldi    ZL,LOW(clear*2)
       ldi    count,0x10                  ;max 16 chars per line
       rcall  WrLCD                       ;Transmit message

       ldi    LCD_Reg,SecondLine_Add      ;cursor to 2nd line, first position
       rcall  LCD_command                 ;

       ldi    ZH,HIGH(clear*2)          ;Point to clear subroutine in LCD.INC
       ldi    ZL,LOW(clear*2)      ;
       ldi    count,0x10                      ;max 16 chars per line
       rcall  WrLCD                       ;Transmit message
```

```
ret

;subroutine to display mode 1 in LCD display
disp_mode1:

        ldi     LCD_Reg,FirstLine_Add       ;cursor to 1st line, first position
        rcall   LCD_command                 ;should already be there from initialize

        ldi     ZH,HIGH(print_mode1*2)      ;Point to print_mode1 subroutine in LCD.INC
        ldi     ZL,LOW(print_mode1*2)
        ldi     count,0x10                  ;max 16 chars per line
        rcall   WrLCD                       ;Transmit message

        ldi     LCD_Reg,SecondLine_Add      ;cursor to 2nd line, first position
        rcall   LCD_command                 ;

        ldi     ZH,HIGH(clear*2)          ;Point to print_mode1 subroutine in LCD.INC
        ldi     ZL,LOW(clear*2)       ;
        ldi     count,0x10                      ;max 16 chars per line
        rcall   WrLCD                       ;Transmit message

ret

;subroutine to display mode 2 in LCD display
disp_mode2:

        ldi     LCD_Reg,FirstLine_Add       ;cursor to 1st line, first position
        rcall   LCD_command                 ;should already be there from initialize

        ldi     ZH,HIGH(print_mode2*2)      ;Point to print_mode2 subroutine in LCD.INC
        ldi     ZL,LOW(print_mode2*2)
        ldi     count,0x10                      ;max 16 chars per line
        rcall   WrLCD                       ;Transmit message

        ldi     LCD_Reg,SecondLine_Add      ;cursor to 2nd line, first position
        rcall   LCD_command                 ;

        ldi     ZH,HIGH(clear*2)          ;Point to clear subroutine in LCD.INC
        ldi     ZL,LOW(clear*2)       ;
        ldi     count,0x10                      ;max 16 chars per line
        rcall   WrLCD                       ;Transmit message

ret

;subroutine to display mode 3 in LCD display
disp_mode3:

        ldi     LCD_Reg,FirstLine_Add       ;cursor to 1st line, first position
        rcall   LCD_command                 ;should already be there from initialize

        ldi     ZH,HIGH(print_mode3*2)      ;Point to print_mode3 subroutine in LCD.INC
        ldi     ZL,LOW(print_mode3*2)
        ldi     count,0x10                      ;max 16 chars per line
        rcall   WrLCD                       ;Transmit message

        ldi     LCD_Reg,SecondLine_Add      ;cursor to 2nd line, first position
        rcall   LCD_command                 ;

        ldi     ZH,HIGH(clear*2)        ;Point to clear subroutine in LCD.INC
        ldi     ZL,LOW(clear*2)         ;
        ldi     count,0x10                  ;max 16 chars per line
```

```
        rcall   WrLCD                           ;Transmit message

ret

;subroutine to display error in LCD display
disp_error:

        ldi     LCD_Reg,FirstLine_Add           ;cursor to 1st line, first position
        rcall   LCD_command                     ;should already be there from initialize

        ldi     ZH,HIGH(print_error1*2) ;Point to print_error1 subroutine in LCD.INC
        ldi     ZL,LOW(print_error1*2)
        ldi     count,0x10                          ;max 16 chars per line
        rcall   WrLCD                           ;Transmit message

        ldi     LCD_Reg,SecondLine_Add          ;cursor to 2nd line, first position
        rcall   LCD_command                     ;

        ldi     ZH,HIGH(print_error2*2)         ;Point to message2
        ldi     ZL,LOW(print_error2*2)          ;
        ldi     count,0x10                          ;max 16 chars per line
        rcall   WrLCD                           ;Transmit message

ret


disp_3:                 ;subrountine to display 3 in 7 segment
sbi PORTD, PIND4
sbi PORTD, PIND5
cbi PORTD, PIND6
cbi PORTD, PIND7

call DELAY_05

ret

disp_2:                 ;subrountine to display 2 in 7 segment

cbi PORTD, PIND4
sbi PORTD, PIND5
cbi PORTD, PIND6
cbi PORTD, PIND7

call DELAY_05

ret

disp_1:                 ;subrountine to display 1 in 7 segment
sbi PORTD, PIND4
cbi PORTD, PIND5
cbi PORTD, PIND6
cbi PORTD, PIND7

call DELAY_05

ret

;subroutine to delat for 0.5 second
DELAY_05:               ;subroutine to delay for 0.5 second
        ldi r30, 31             ; load r16 with 31
```

```
OUTER_LOOP:             ; outer loop label
       ldi r24, low(1021)   ; load registers r24:r25 with 1021, our new
                        ; init value
       ldi r25, high(1021)  ; the loop label

DELAY_LOOP:             ; "add immediate to word": r24:r25 are
                        ; incremented
       adiw r24, 1          ; if no overflow ("branch if not equal"), go

                        ; back to "delay_loop"
       brne DELAY_LOOP
       dec r30              ; decrement r16
       brne OUTER_LOOP      ; and loop if outer loop not finished

       ret                  ; return from subroutine

;-------------------------------------
;All custom library included
.include "LCD.inc"
.include "delays.inc"
.include "I2C.inc"
;

;Custom Library can be found with the file included in the attached file.
```

Library code (LCD.inc)

```
lcd_putchar:
        push    LCD_Reg                 ;save LCD_Reg (it's destroyed in between)

        cbr     LCD_Reg, 0b00001111     ;we have to write the high nibble of our LCD_Reg first
                                        ;so mask off the low nibble
        sbr     LCD_Reg,(1<<LCD_RS) | (1<<LCD_E)     ;LCD_RS must be high to send data
;       sbr     LCD_Reg,(1<<LCD_RS)|(1<<LCD_E)|(1<<BackLight)     ;if you have a BackLight
        rcall   WriteI2C
        rcall   PulseE                  ;strobe Enable

        pop     LCD_Reg                 ;restore the LCD_Reg
        swap    LCD_Reg                 ;write the LOW nibble of the LCD_Reg to
        cbr     LCD_Reg, 0b00001111     ;HIGH port nibble

        sbr     LCD_Reg,(1<<LCD_RS) | (1<<LCD_E)     ;LCD_RS must be high to send data
;       sbr     LCD_Reg,(1<<LCD_RS)|(1<<LCD_E)|(1<<BackLight)     ;if you have a BackLight
        rcall   WriteI2C
        rcall   PulseE                  ;strobe Enable

        ldi     temp,2                  ;2 mS delay
        mov     dcount,temp
        rcall   DelaymS
        ret
;---------------------------------------------------------------
```

```
lcd_command:                    ;same as LCD_putchar, but with RS low!
        push    LCD_Reg                 ;send the upper nibble first

        cbr     LCD_Reg, 0b00001111     ;clr lower nibble
        sbr     LCD_Reg,(1<<LCD_E)      ;E high
;       sbr     LCD_Reg,(1<<LCD_E)|(1<<BackLight)     ;if you have a BackLight
        rcall   WriteI2C
        rcall   PulseE

        pop     LCD_Reg                 ;send the lower nibble
        swap    LCD_Reg

        cbr     LCD_Reg, 0b00001111     ;clr the lower nibble
        sbr     LCD_Reg,(1<<LCD_E)      ;E high
;       sbr     LCD_Reg,(1<<LCD_E)|(1<<BackLight)     ;if you have a BackLight
        rcall   WriteI2C
        rcall   PulseE
        ret
;-----------------------------------
PulseE:
        cbr     LCD_Reg,(1<<LCD_E)      ;strobe E
        rcall   WriteI2C
        ret
;-----------------------------------
LCD_init:
        ldi     temp,100                ;100 mS delay
        mov     dcount,temp
        rcall   DelaymS

        ldi     count,3                 ;send it three times

LCD_Init_Lp:
        ldi     LCD_Reg, 0x30           ;LCD is still in 8-BIT MODE

        sbr     LCD_Reg,(1<<LCD_E)      ;E high (idle)
;       sbr     LCD_Reg,(1<<LCD_E)|(1<<BackLight)     ;if you have a BackLight
        rcall   WriteI2C
        rcall   PulseE

        ldi     temp,10                 ;10 mS delay
        mov     dcount,temp
        rcall   DelaymS

        dec     count
        brne    LCD_Init_Lp             ;end of 0x30 cmd loop

        ldi     LCD_Reg, 0x20           ;change to 4-bit mode.
                                        ;LCD is still in 8-BIT MODE
        sbr     LCD_Reg,(1<<LCD_E)      ;E high (idle)
;       sbr     LCD_Reg,(1<<LCD_E)|(1<<BackLight)     ;if you have a BackLight
```

```
        rcall   WriteI2C
        rcall   PulseE

        rcall   Delay100us

        ldi     LCD_Reg, 0x28           ;NOW we are in 4-BIT MODE, 2 lines, 5*7 font
        rcall   LCD_command             ;

        rcall   Delay100us

        ldi     LCD_Reg, 0x0C           ;Display ON, cursor OFF, blink OFF
;       ldi     LCD_Reg, 0x0F           ;Display on, cursor on, blinking ON
        rcall   LCD_command

        rcall   Delay100us

        ldi     LCD_Reg, 0x01           ;clear display, cursor -> home
        rcall   LCD_command

        ldi     temp,5                  ;need > 4 mS delay
        mov     dcount,temp
        rcall   DelaymS

        ldi     LCD_Reg, 0x06           ;auto-inc cursor
        rcall   LCD_command

        rcall   Delay100us

        ldi     LCD_Reg, 0x02           ;return cursor to Home Display
        rcall   LCD_command

        rcall   Delay100us
        ret
;---------Writes String to LCD display from Z pointer ------------
WrLCD:
        lpm     LCD_Reg,Z+
        rcall   lcd_putchar
        dec     count           ;
        brne    WrLCD           ;
        ret
;---------------------------------

clear:
.db     "               "

welcome1:
.db     "WELCOME TO GROUP"

welcome2:
.db     "   12 PROJECT   "
```

```
print_mode1:
.db     "MODE:   1      "

print_mode2:
.db     "MODE:   2      "

print_mode3:
.db     "MODE:   3      "

print_error1:
.db     "  ERROR! FOOD   "

print_error2:
.db     "  TRAY EMPTY!   "
;---------------------------------
```

Library Code (DELAYS.inc)

```
;*************************************************
;**** Constant declarations Data Rate for delay ****
;.equ    Crystal = 7372          ;
;.equ    Crystal = 3686          ;
.equ    Crystal = 16000         ;
;*************************************************************************
;*
;* FUNCTION
;*       delay
;*
;* DESCRIPTION
;* Make delay 1mS (x dcount).
;* uses dcount, dcount2, and dcount3
;* come here with a value in dcount (1 = 1mS)
;*************************************************************************
DelaymS:
        ldi     temp,40
        mov     dcount3,temp
dl2:
        ldi     temp,(Crystal/120)
        mov     dcount2,temp
dl1:
        dec     dcount2
        brne    dl1

        dec     dcount3
        brne    dl2

        dec     dcount
        brne    DelaymS
        ret
;*************************************************************************
; 40us at 16 MHz
```

```
;
Delay40us:
        ldi     temp,211
        mov     dcount,temp

D40_uS:
        dec     dcount
        brne    D40_uS
        ret
;*************************************************************************
;
; 100us at 16 MHz
;
Delay100us:
        ldi     temp,3
        mov     dcount,temp

        ldi     temp,16
        mov     dcount2,temp

D100_uS:
        dec     dcount2
        brne    D100_uS
        dec     dcount
        brne    D100_uS
        ret
;*************************************************************************
```

Library Code (I2C.inc)

```
;*************************************************************************
;*
;* FUNCTION
;*      i2c_init
;*
;* DESCRIPTION
;*      Initialization of the I2C bus interface.
;*
;* USAGE
;*      Call this function once to initialize the I2C bus. No parameters
;*      are required.
;*
;* RETURN
;*      None
;*
;* NOTE
;*      PORTD and I2CDDR pins not used by the I2C bus interface will be
;*      set to Hi-Z (!).
;*
;* COMMENT
;*      This function can be combined with other PORTD initializations.
```

```
;*
;*****************************************************************************
;
i2c_init:
        clr     temp            ;
                                ;
        out     PORTB,temp      ; set I2C pins to open collector
        out     DDRB,temp
        ret
;
;*****************************************************************************
;
;*
;* FUNCTION
;*      i2c_hp_delay
;*      i2c_qp_delay
;*
;* DESCRIPTION
;*      hp - half i2c clock period delay (normal: 5.0us / fast: 1.3us)
;*      qp - quarter i2c clock period delay (normal: 2.5us / fast: 0.6us)
;*
;*      Adjusted for arduino board with 16MHz clock
;*
;* USAGE
;*      no parameters
;*
;* RETURN
;*      none
;*
;*****************************************************************************

i2c_hp_delay:
        ldi     i2cdelay,24             ;5uS delay @ 16MHz

i2c_hp_delay_loop:
        dec     i2cdelay
        brne    i2c_hp_delay_loop
        ret

i2c_qp_delay:
        ldi     i2cdelay,11             ;2.5uS delay @ 16MHz

i2c_qp_delay_loop:
        dec     i2cdelay
        brne    i2c_qp_delay_loop
        ret
;
;*****************************************************************************
;
WriteI2C:
        rcall   i2c_start               ;send start bit and address & get ack bit
;       brcs    errorloop               ;not enabled in this version
```

```
        mov     i2cdata,LCD_Reg     ;
        rcall   i2c_write           ;send data byte, get ack bit and send stop bit
;       brcs    errorloop           ;not enabled in this version

        rcall   i2c_stop            ;
        ret

;errorloop                          ;resend X amount of times if you want
;       rjmp    errorloop
;*****************************************************************************
;*
;* FUNCTION
;*      i2c_start
;*
;* DESCRIPTION
;*      Generates start condition and sends slave address.
;*
;* USAGE
;*      i2cadr - Contains the slave address and transfer direction.
;*
;* RETURN
;*      Carry flag - Cleared if a slave responds to the address.
;*
;* NOTE
;*      IMPORTANT! : This function must be directly followed by i2c_write.
;*
;*****************************************************************************
;
i2c_start:
        ldi     i2cdata,i2cadr ; copy address to transmitt register
        sbi     I2CDDR,SDAP                 ; data line low (start bit)
        rcall   i2c_qp_delay   ; quarter period delay
;
;continue to i2c_write
;
;*****************************************************************************
;*
;* FUNCTION
;*      i2c_write
;*
;* DESCRIPTION
;*      Writes data (one byte) to the I2C bus. Also used for sending
;*      the address.
;*
;* USAGE
;*      i2cdata - Contains data to be transmitted.
;*
;* RETURN
;*      Carry flag - Set if the slave respond transfer.
;*
```

```
;* NOTE
;*      IMPORTANT! : This function must be directly followed by i2c_get_ack.
;*
;*********************************************************************************

i2c_write:
        sec                             ; set carry flag
        rol     i2cdata         ; shift in carry and out bit one
        rjmp    i2c_write_first

i2c_write_bit:
        lsl     i2cdata         ; if transmit register empty

i2c_write_first:
        breq    i2c_get_ack             ; goto get acknowledge
        sbi     I2CDDR,SCLP                     ; force SCL low

        brcc    i2c_write_low ; if bit high
        nop                             ;(equalize number of cycles)
        cbi     I2CDDR,SDAP                     ;release SDA
        rjmp    i2c_write_high

i2c_write_low:                          ; else
        sbi     I2CDDR,SDAP                     ; force SDA low
        rjmp    i2c_write_high          ; (equalize number of cycles)

i2c_write_high:
        rcall   i2c_hp_delay  ; half period delay
        cbi     I2CDDR,SCLP                     ; release SCL
        rcall   i2c_hp_delay  ; half period delay

        rjmp    i2c_write_bit
;*********************************************************************************
;*
;* FUNCTION
;*      i2c_get_ack
;*
;* DESCRIPTION
;*      Get slave acknowledge response.
;*
;* USAGE
;*      (used only by i2c_write in this version)
;*
;* RETURN
;*      Carry flag - Cleared if a slave responds to a request.
;*
;*********************************************************************************

i2c_get_ack:
        sbi     I2CDDR,SCLP                     ; force SCL low
```

```
        cbi     I2CDDR,SDAP                 ; release SDA
        rcall   i2c_hp_delay        ; half period delay
        cbi     I2CDDR,SCLP                 ; release SCL

i2c_get_ack_wait:
        sbis    I2CPIN,SCLP         ; wait SCL high
                                    ;(In case wait states are inserted)
        rjmp    i2c_get_ack_wait

        clc                         ; clear carry flag
        sbic    I2CPIN,SDAP         ; if SDA is high
        sec                         ; set carry flag
        rcall   i2c_hp_delay  ; half period delay
        ret
;
;*****************************************************************************
;*
;* FUNCTION
;*      i2c_stop
;*
;* DESCRIPTION
;*      Assert stop condition.
;*
;* USAGE
;*      No parameters.
;*
;* RETURN
;*      None.
;*
;*****************************************************************************
;
i2c_stop:
        sbi     I2CDDR,SCLP                 ; force SCL low
        sbi     I2CDDR,SDAP                 ; force SDA low
        rcall   i2c_hp_delay  ; half period delay
        cbi     I2CDDR,SCLP                 ; release SCL
        rcall   i2c_qp_delay  ; quarter period delay
        cbi     I2CDDR,SDAP                 ; release SDA
        rcall   i2c_hp_delay  ; half period delay
        ret
;*****************************************************************************
```

3.0 PROCEDURE

The list of components, circuit configuration, and experiment setup will be explained in this section.

3.1 LIST OF COMPONENTS

The list of components that used in this system can be seen in Table 1 below: -

Table 1 List of components

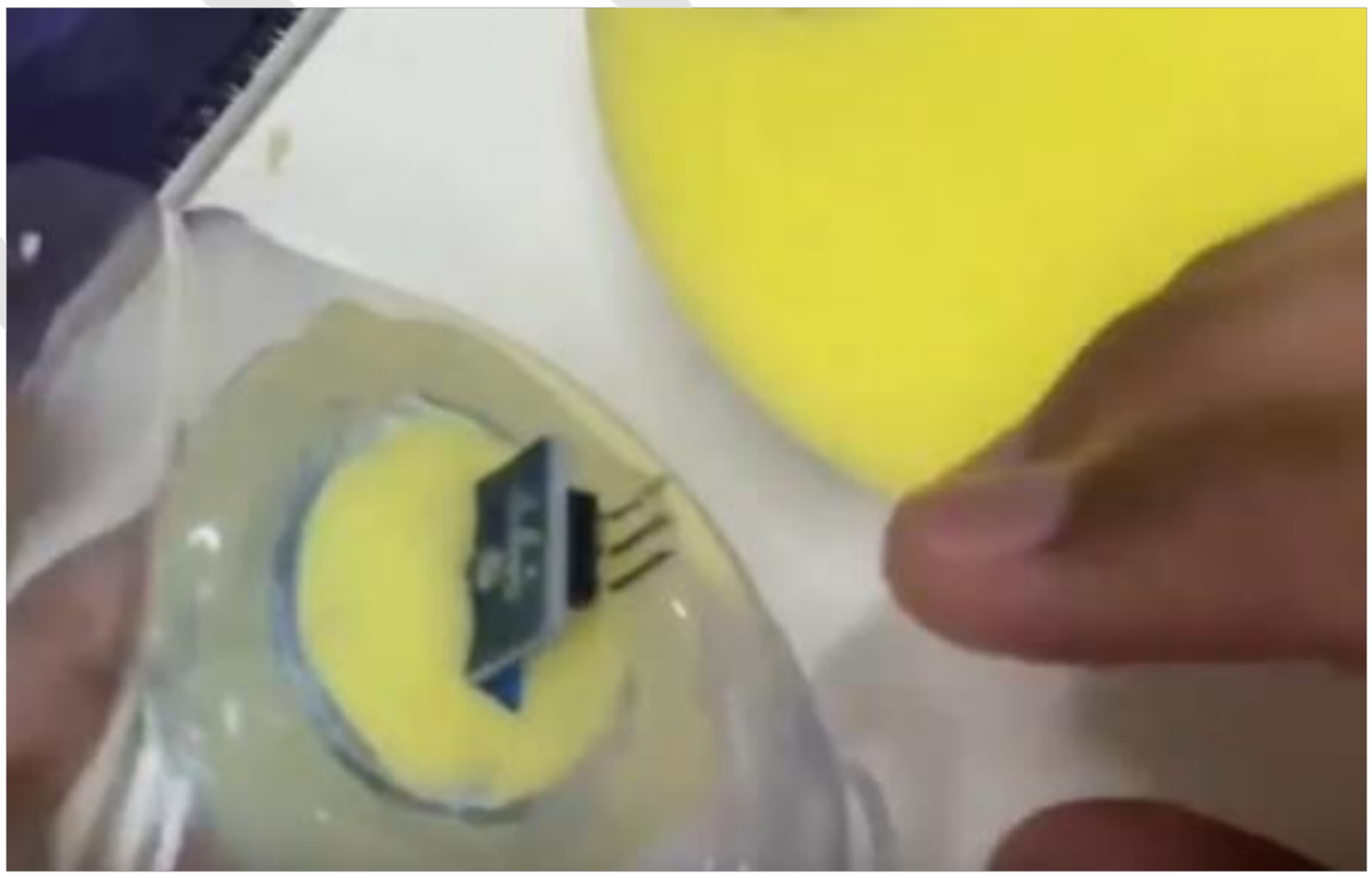
No	Names	Quantity
1	Arduino UNO	1
2	IR sensor module	1
3	6V Motor	1
4	Mini piezo buzzer	1
5	Push button	3
6	3 pin mini slide switch	1
7	7 segments display	1
8	7 segment decoder (IC CD4511)	1
9	16 x 2 LCD Display	1
10	Resistor	4
11	1.5V battery	4
12	4x 1.5V battery case	1
13	20000 mAh power bank	1
14	Breadboard	2
15	Aluminium profile 2020 (30mm)	2
16	Aluminium profile 2020 (20mm)	2
17	Metal plate (8mm x 3mm)	2
18	Food container	1
19	FD04A Motor Driver	1

3.2 EXPERIMENT SETUP

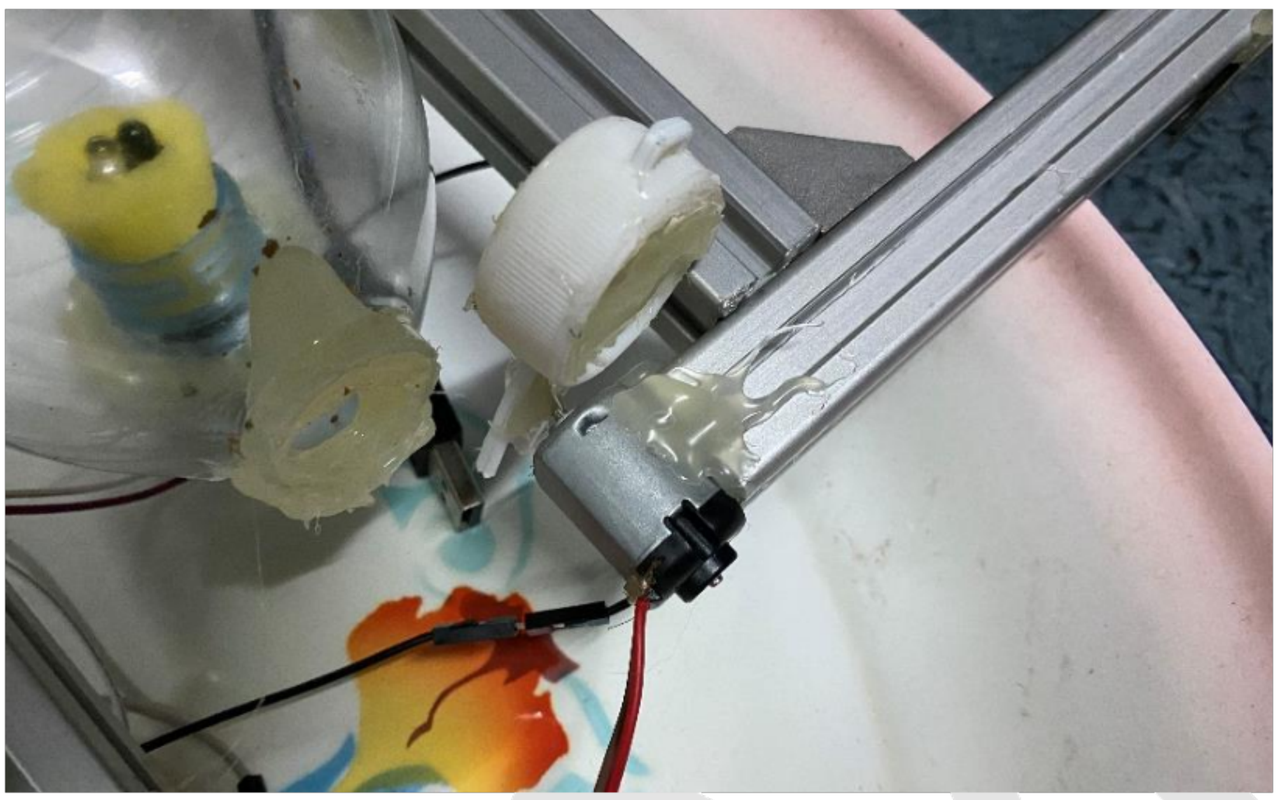
1. First of all, the food container was being prepared by cutting the orifice of the bottle which it will be the bottom part of the food container as the hole to allow dispensing the fish food.

2. Then, bottom part of the bottle was cut to make the opening to refill the food

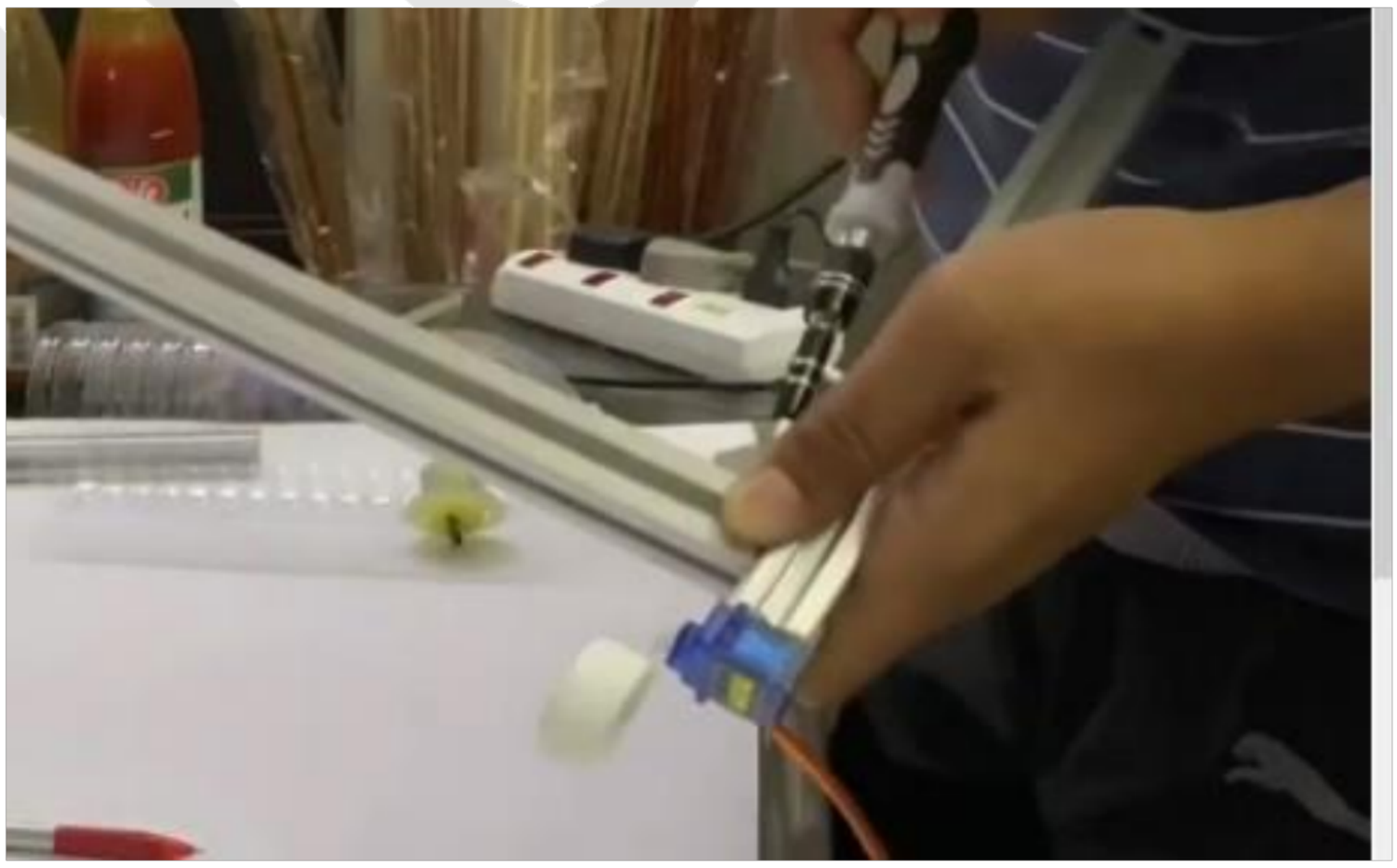
3. Then, a hole was made at a few cm above the bottom part of food container to place the infrared sensor. The orifice was being glued to hole using hot glue gun. first before inserting the infrared sensor and the sponge. This sponge is to prevent fish food from spilling out through the hole.



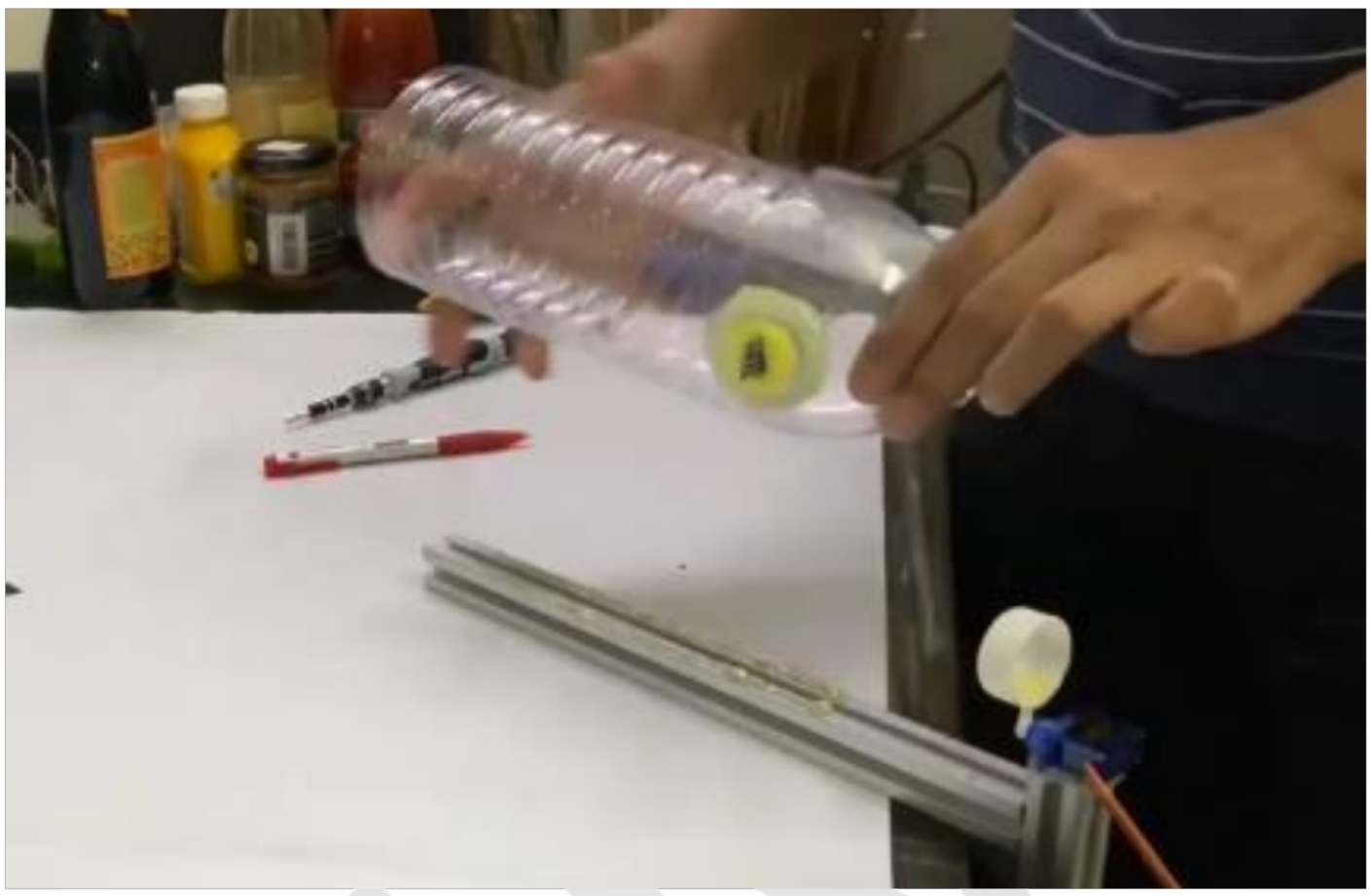
4. Next, another hole was made at the bottle cap and the motor shaft was inserted into the hole and glued together using hot glue gun. The motor was also glued to the aluminium profile.



5. The next step is to screw together all the aluminium profile that have been cut based on desired measurements to form the stand for the food container.



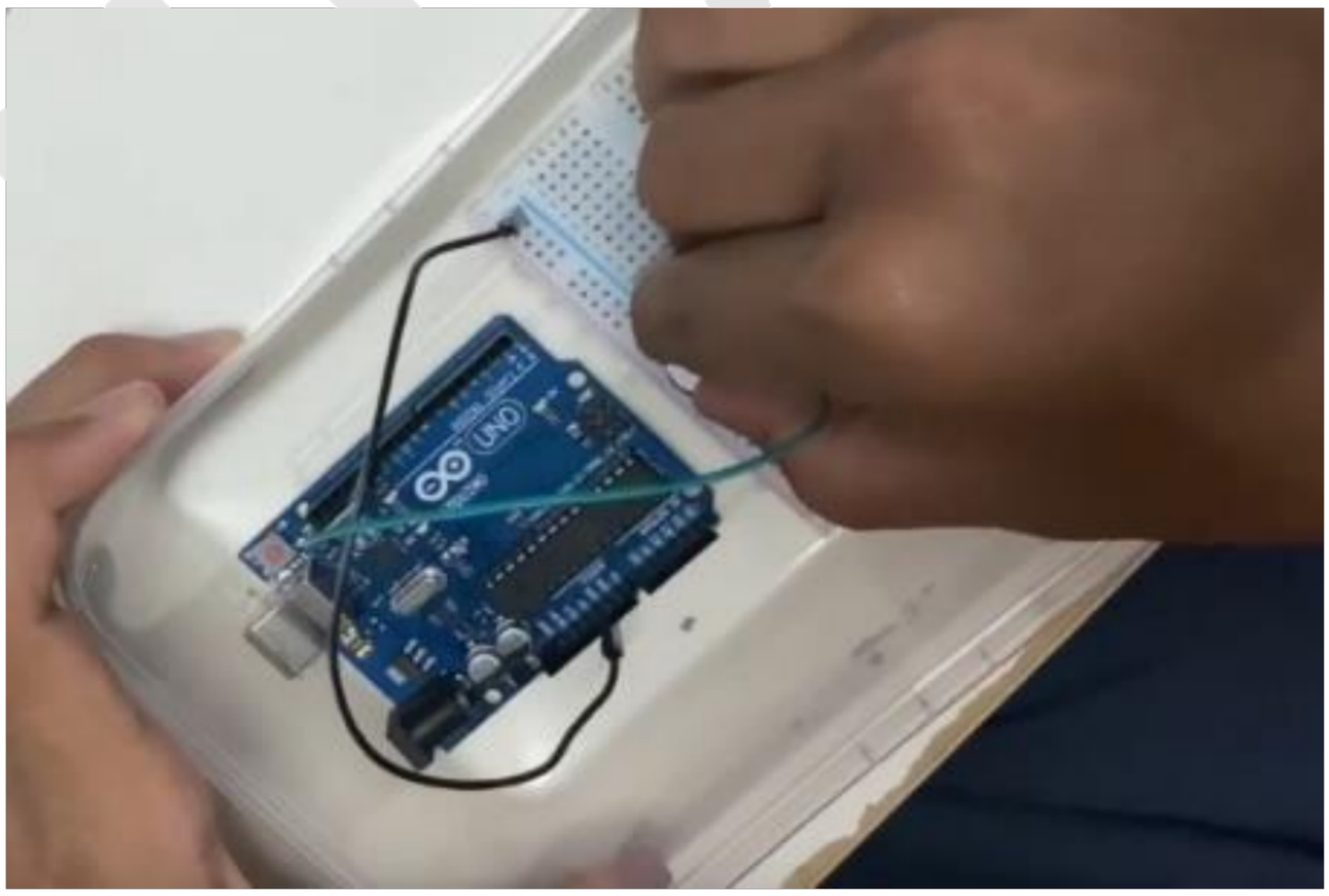
6. Last step in mechanical part is to glue food container between aluminium profile stands.



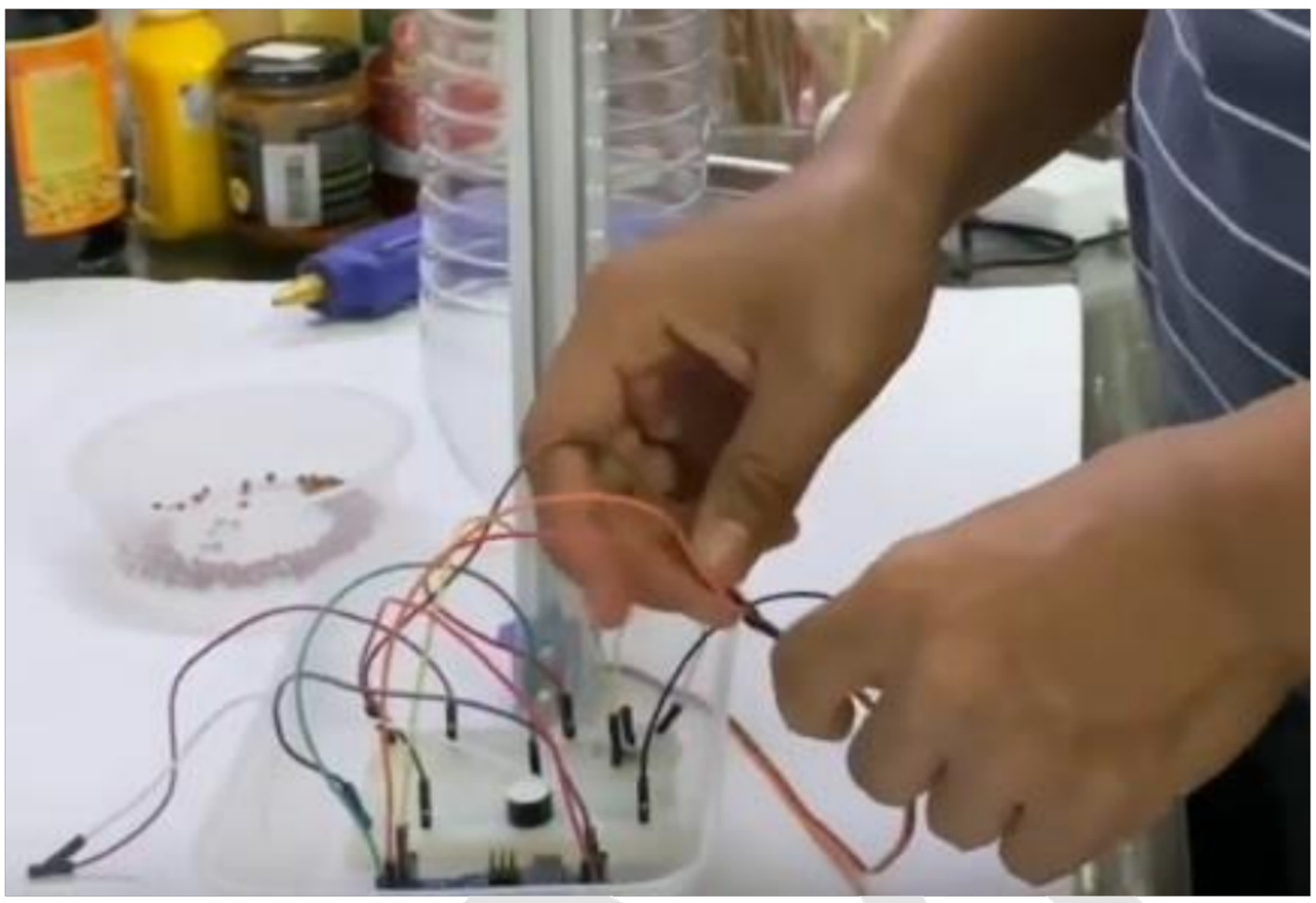
7. The step in the day before was being continued in the next day to construct the box to place the controller, breadboard, and the other electrical components. Starting by making a hole at the side of the box.

8. Then, both breadboard and Arduino were glued on the box.

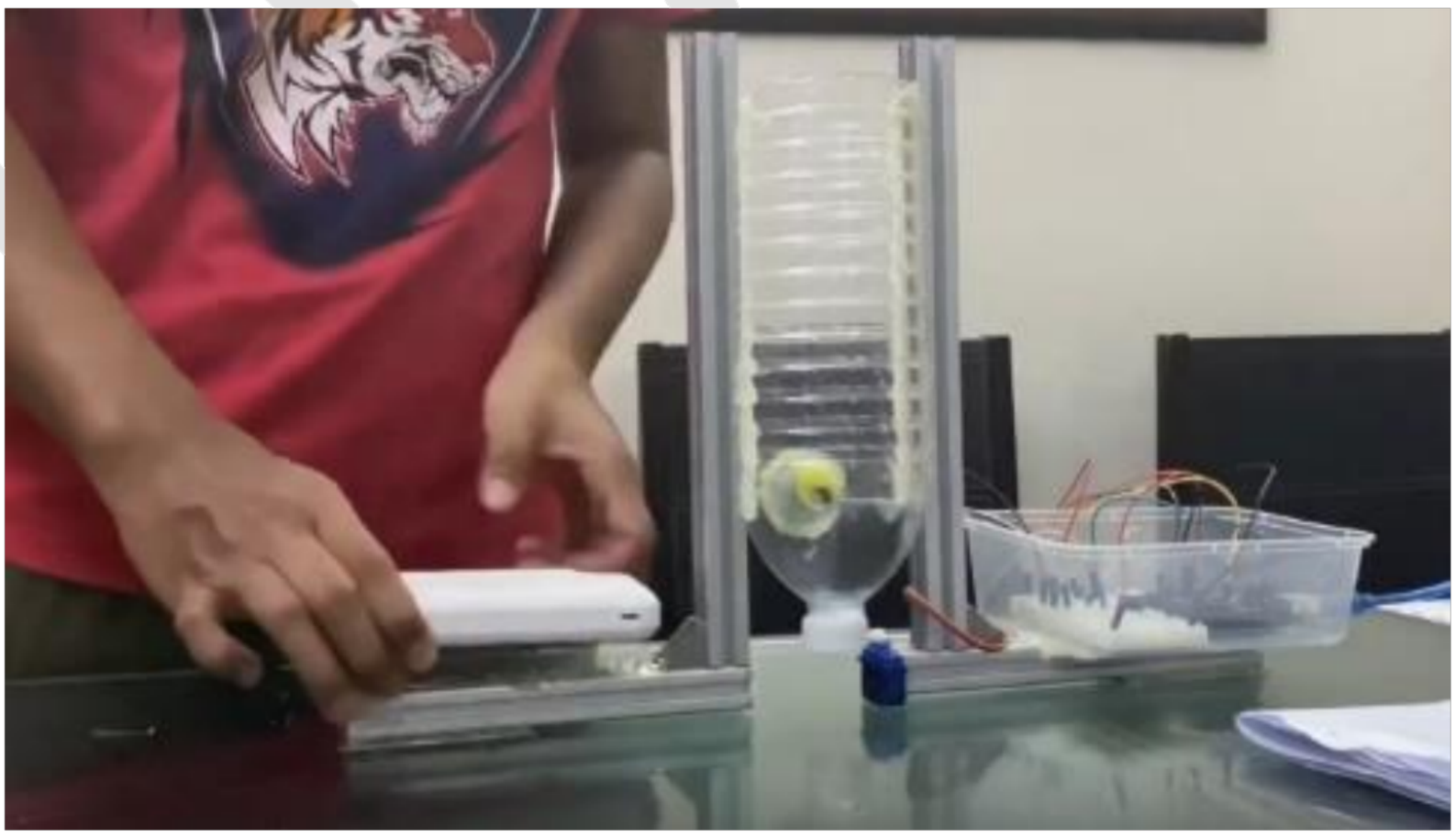
9. And then the circuit was being constructed by connecting the components into the right pins with proper wiring on the breadboard and Arduino board.



10. The circuit box then glued into the aluminium profile.



11. Lastly, the power source which is the power bank was glued on the aluminium profile.



12. The programming code was built based on the proposed flowchart and get tested on the system until the programming code does not has any mistake and the system done the task according to the programmed code.

3.3 CIRCUIT CONFIGURATION

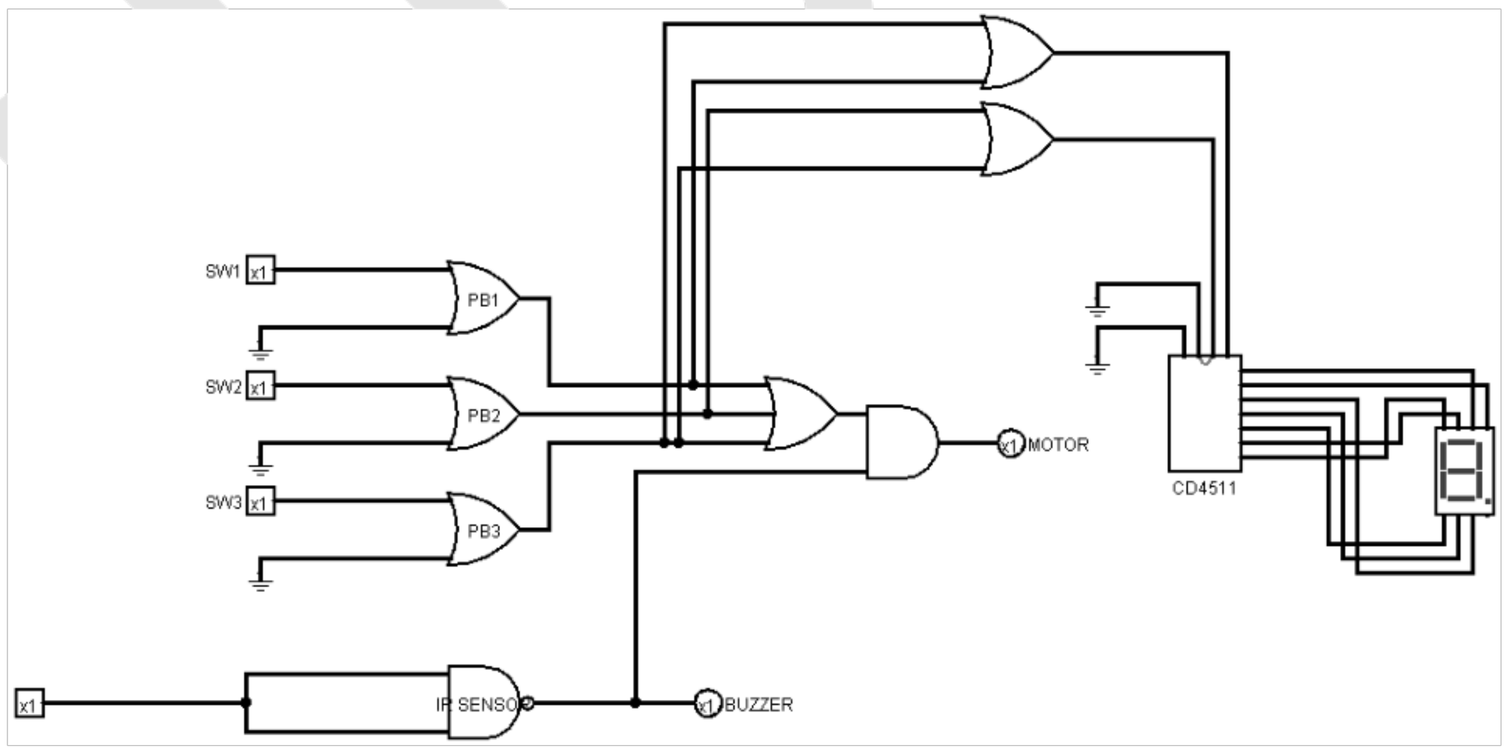


Figure 8 Digital logic circuit of our system

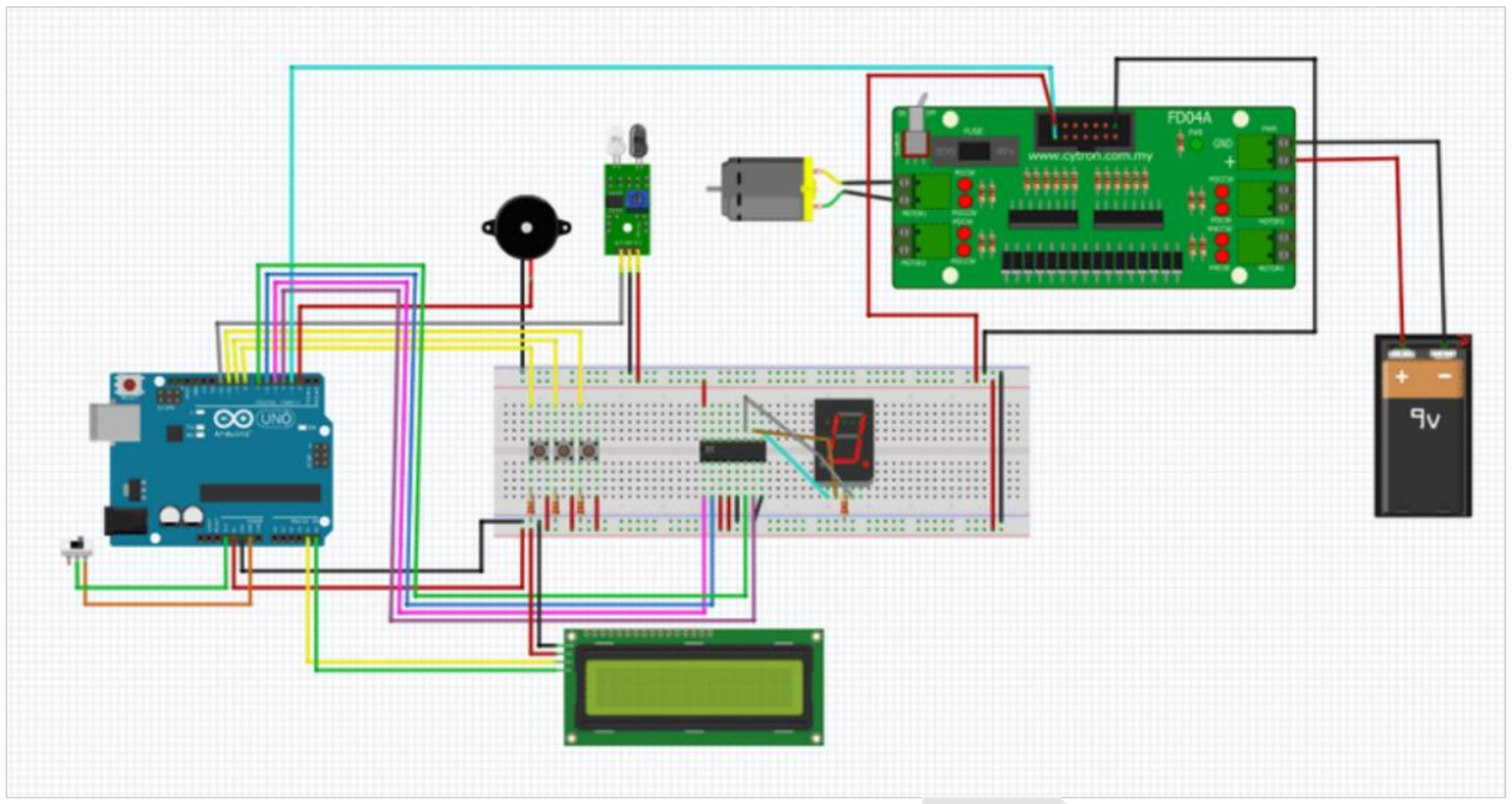


Figure 9 Schematic diagram of the system (Digital Logic Circuit and Microprocessor)

The pin connection of the circuit can be seen in Table 1 below: -

Table 2 Pin configuration

Components	Arduino Pin
IR Sensor	PB3
Pushbutton 1	PB0
Pushbutton 2	PB1
Pushbutton 3	PB2
7-Segment IN1	PD4
7 -Segment IN2	PD5
7 -Segment IN3	PD6
7 -Segment IN4	PD7
Buzzer	PD2
Motor	PD3
LCD_I2C-SDA	PC4
LCD_I2C-SCL	PC5

4.0 DISCUSSION AND CONCLUSION

In conclusion, our group has successfully built the Automatic Fish Feeder for pet fish at home by implementing DLD and Microprocessor's knowledge into our project's operating system. There are three modes that can be chosen by the users, which will make their life easier to feed the fish automatically. Besides, all of our objectives also have been achieved in this Mini project. Nevertheless, some problems occurred during the process of developing this system, which can be described as following: -

The problem that we encounter are:

1. Problem : The output of the LCD is hanging or haywire when changing sentence or word output to display

   Solution : Set a delay before displaying the next output on the LCD

2. Problem : The container door did not stop exactly at the right initial position when closing after released food.

   Solution : Create a stopper to ensure that the container doorstop at the exact initial position.

3. Problem : When the food capacity is more than half of the food container, the motor could not open or move the container door.

   Solution : Replace the current motor with a motor that has higher torque.