VE373 Project Report
Eat-on-Time: Auto Feeding Machine for Cats

Jingbo Li 519370910040
Runxi Wang 519021911166
Yichen Cai 519021911200

August 5, 2022

Contents

1 Introduction

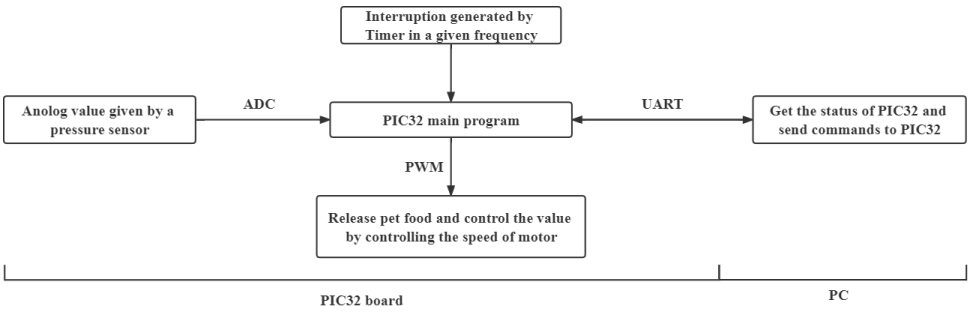


Figure 1: Functional diagram

In this project, we made use of components in PIC32 to implement a simple feeding machine. Our goal is to realize three main functions,

1. On-time feeding: Supported by timer interrupt and PWM. A motor controlled by a PWM mode output compare component is used to drive the food container switch. And timer will interrupt for every preset time period.

2. User-triggered feeding: Supported by PWM and UART module. Users can use a PC equipped with UART-USB converter to send command to PIC32 and control food releasing as asynchronous to pre-defined time period. Users can send different command to trigger different food releasing amount with the help of PWM.

3. Remain detection: Supported by ADC module. A pressure sensor is used to detect the remain on the food plate. When there is too much remain, the on-time feeding function will be disabled until when the plate is cleared out.

The relations between each component on PIC32 and each function is shown in Figure 1. And the system design with detailed connections between each hardware components are shown in 2.

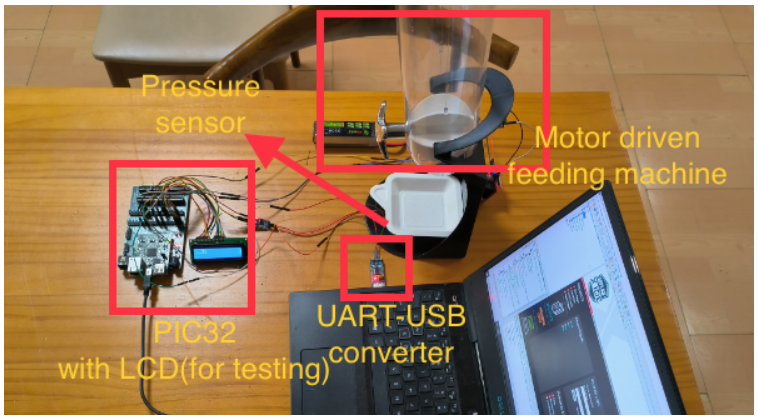


Figure 2: System design and hardware setup

2 Function Implementation

2.1 Timer Part

Timer is used in this project to implement the on-time feeding function. For real-life usages, the timer will trigger food releasing for every 8 hours. But for demo time, to visualize the function in a short time, we set the timer to interrupt for every 30 seconds.
We use timer 3 for this part. The clock source for timer 3 is `PBCLK`and the prescale parameter is `1:256`. For initialization, we set `TMR3` as 0 and `PR3` as either the pre-defined period we want or the maximum value of this 16-bit timer (in this case each interrupt coming from this timer will be accumulated to calculate the time passed). So only when the accumulated period of time is reached, the interrupt of this timer will trigger a food releasing.

2.2 Motor Driving Part

We used a motor to rotate the turbine in the food container. But the motor is not directly controlled by PIC32. We use a motor driver L298N (Figure 3) here to control the rotating pattern of the motor. The power supply of the motor is connected to L298N by the `12V` and `GND` port as shown in Figure 3. Port `p1` and `p2` are connected to `OC1pin` and `GND` in PIC32 respectively. Finally, port `m1` and `m2` are connected to the two pins on the motor (either order is fine because we do not care about the rotating direction).

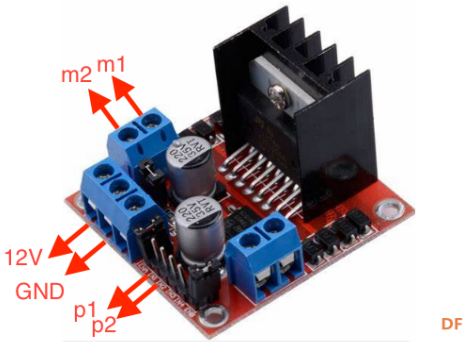
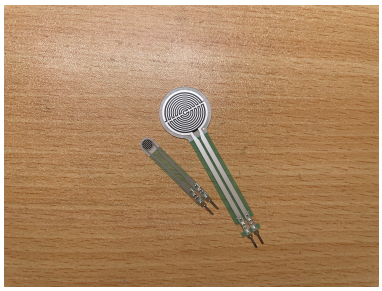


Figure 3: L298N Motor Driver

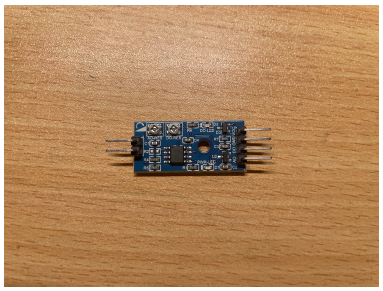
2.3 PWM Part

As described above, port `p1` of the motor driver is connected to `OC1pin` on PIC32, which means that the motor is controlled by the output compare module in PIC32. Since in our feeding functions, we need to implement different rotating speed of the turbine inside the food container, we need to apply the characteristics of PWM, whose output pulse width can be changed flexibly.
`OC1` uses 16-bit mode timer 2 (clock source `PBCLK`prescaled 1:256), and is set as PWM mode. We initialize `OC1RS` as 0 because the motor is initially stopped. If we want to change the rotating speed of the motor, we just change the duty cycle or `OC1RS` to do so. As the value of `OC1RS` increases, the motor rotates faster.

2.4 ADC Part



(a) Pressure sensitive resistor



(b) Pressure converter

Figure 4: Pressure sensor

The ADC implementation is shown in our function that is related to the pressure sensor. The expected function is that the feeding machine will not release food when the pressure sensor detects enough food on the plate.
The pressure sensor is composed of a pressure sensitive resistor and a pressure converter module shown in Figure 4. The pressure converter module will output an analog signal according to the resistance of the resistor. Then the ADC will convert this analog signal to a digital signal.
The `AD1` module in PIC32 board is selected as our ADC. It is configured as manual sample and auto convert mode. The ADC will start to sample when the `SAMP` bit of `AD1CON1` is 1, and when ADC finishes sampling, it will automatically convert the signal to a digital signal.
After the digital signal is ready and written to the buffer, the value is read out and compared to a preset value. If it exceeds the value, which means the food on the plate is enough, other functions like timer and PWM will be influenced so that the motor will not rotate.

2.5 UART Part

The UART implementation is shown in out project that is related to the PC control function.
In this part, we use a USB-UART converter to translate the signal between UART ports and USB ports. A picture of the converter is shown in figure 5.

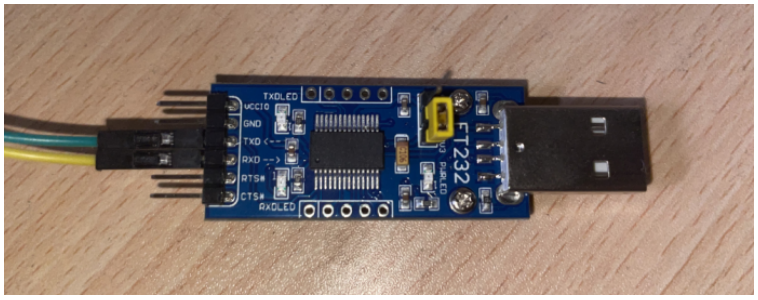


Figure 5: USB-UART Converter

The converter has 1 USB port, and also 6 other ports. In this project, we only use the USB port, TXD port, and RXD port. The USB port is connected to the PC, while TXD and RXD are connected to the U1RX and U1TX port on the PIC32 board, doing the job of receiving and sending data respectively.
As for the embedded programming part, we implement an interrupt handler work as a UART receiver. This handler will be called every time the receive buffer of the UART module is not empty. The handler will read out the data in the receive buffer, and call the corresponding functions according to the data. The handler will clear the receive buffer and also clear the interrupt flag after all the reactions have taken.
A function for sending out data through UART is also implemented in this project. Given a list of unsigned chars, the function will send out all the characters through the UART module, so that PC can receive the data showing the status of the PIC32 system.
For the software on the PC side, we use a tool which can interact with the USB port in Windows 10 environment. A screenshot of the software is shown as figure 6. We can pre-define some commands on the right. Data received are also shown automatically on the left side.
After all these implementations, we can now use UART to realize the communication between PC and PIC32 board. PC can control the PWM mode, the timer mode, and also the switch of food releasing. PC can also request the status of the PIC32 system.

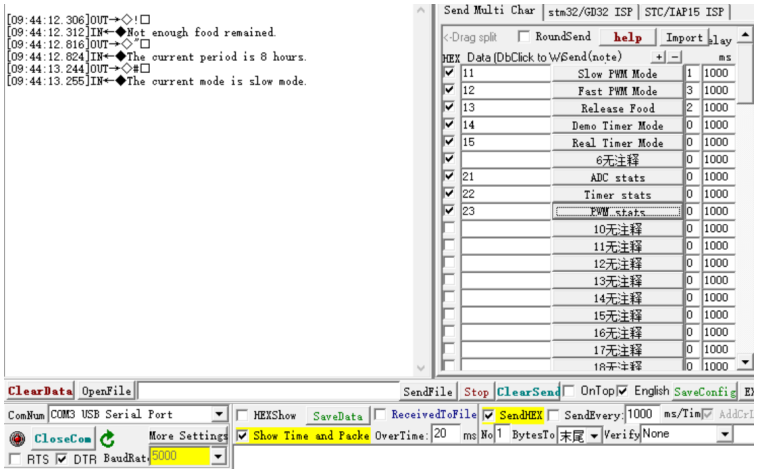


Figure 6: Screenshot of USB Port Interact Software

3 Assemble & Testing

The test plan for this project can be briefly divided into two parts. First, we test each modules independently. After all the modules work well, we assemble all the components and modules together and test as a whole. The modules involved in this project can be divided into five parts, including timer, PWM, motor, ADC, and UART.

3.1 Timer

We can use the timer interrupt handler to switch on/off the RD0 light. We can then measure the duration of on/off. This duration should be the same as the expected time period we set.

3.2 PWM

We set the PWM output to RD0 to test the PWM module. The lightness will indicate the duty cycle of the PWM signal. We can control the duty cycle with button. A brighter light represents a larger duty cycle, and vice versa.

3.3 Motor

The capability of the motor to drive the turbine in the food container was tested first. The result shows that one 30-cycles/min motor is enough for driving the turbine. And then we tested the whole motor driving system, which composes of PWM in PIC32, a battery, a motor driver, and the motor itself. The connections are shown in 7. We set different pulse width for `OC1` output and see whether the turbine is rotating with different speed.

3.4 ADC

To test ADC module, we connect a LCD module to the PIC32 board. The values collected from the ADC module will be shown on the LCD module. We then stick the plate onto the pressure sensitive resistor, and put some amount of cat food in the plate. If the value shown on LCD increases, it represents that our ADC module works correctly.

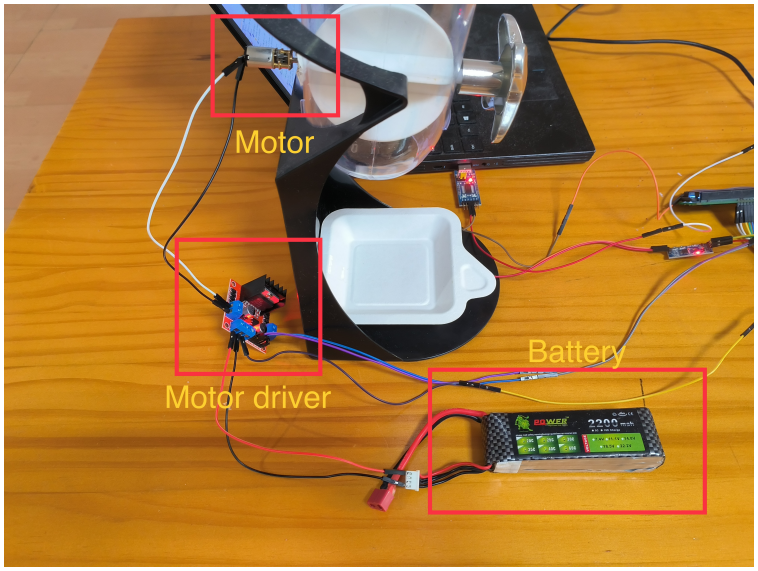


Figure 7: Detailed motor setup

3.5 UART

To test UART module, we first connect the USB-UART converter to the PC, and open the USB port interact software. We send a command to the PIC32 board, and if the module works well, it should return some data back to the PC. The data should be shown on the screen of PC.

3.6 Assemble

After assembling all the modules together, we can test the overall functions.

1. Every 30 seconds, the motor should rotate for some seconds to release food.
2. After sending a command from PC, the rotate speed of the motor should be changed in the next time.
3. PC can directly control the motor to rotate.
4. PC can also request status of the system. System will return the information back to PC.
5. After placing some cat food in the plate, the motor will not release food until some are removed.

4 Final Project Timeline

Our final project implementation timeline is shown in Figure 8.

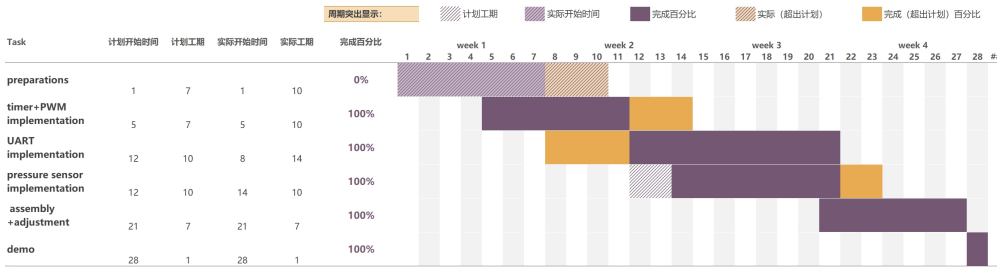


Figure 8: Final timeline arrangement

5 Source Code

```
#include <p32xxxx.h>
/* define macros for LCD instructions*/
#define PERIOD 800
#define LARGE 700
#define SMALL 300
#define LCD_IDLE 0x33
#define LCD_2_LINE_4_BITS 0x28
#define LCD_2_LINE_8_BITS 0x38
#define LCD_DSP_CSR 0x0c
#define LCD_CLR_DSP 0x01
#define LCD_CSR_INC 0x06
#define LCD_SFT_MOV 0x14
typedef unsigned char uchar;
#define RS PORTFbits.RF0
#define E PORTFbits.RF1
#define Data PORTE
#define DEMO_T 58590
#define REAL_T 56246400
#define WEIGHT 620

int T = DEMO_T;
int timer_full_times = 0; // accumulate times of TMR3 is full
int total_time = 0;

int motor_speed = SMALL;
int adc_flag = 1;

void MCU_init();
void PWM_init();
void LCD_init();
void ADC_init();
void UART_init();
void timer_init();
void GenMsec(void);
void DelayMsec(uchar num);
void LCD_putchar(uchar c);
void LCD_goto(uchar addr);
void print_result(int result);
void UART_transmit(const uchar *s);
void UART_handler();
void PWM_on();
void PWM_off();
void return_ADC();
void return_Timer();
void return_PWM();

int main(){
    ADC_init();
    MCU_init();
    timer_init();
    PWM_init();
    LCD_init();
    UART_init();
    while(1){
        AD1CON1bits.SAMP = 0x1;//start sampling
        AD1CON1bits.SAMP = 0;
        while(!AD1CON1bits.DONE){}//wait for conversion
        int result = ADC1BUF0;
        if (result >= WEIGHT) adc_flag = 0;
        else adc_flag = 1;
        print_result(result);
        int i = 0;
        while (i < 1000) i++;
    }
}
```

```
void MCU_init() {
    // configure GPIO to be output
    TRISECLR = 0xFF;// RE4-7
    TRISFCLR = 0x3;// RF0-1

    //UART
    TRISFCLR = 0x8; // Set U1TX output

    //PWM test
    TRISDCLR = 0x1; // Set RD0 output




    /*setup interrupt*/
    INTCONbits.MVEC = 1;
    asm("ei");
}

void timer_init() {
    // configure timer3 for time-specific feeding
    T3CON = 0x70; // use PBCLK as clock source, prescale 1:256
    TMR3 = 0;
    PR3 = (T > 0xffff) ? 0xffff : T;
    // interrupt configuration
    IEC0CLR = 0x1000;
    IFS0CLR = 0x1000;
    IPC3SET = 0x1C;// priority: 7
    // enable interrupt and start timer
    IEC0SET = 0x1000;
    T3CONSET = 0x8000;

    // configure timer4 for time-specific feeding
    T4CON = 0x70; // use PBCLK as clock source, prescale 1:256
    TMR4 = 0;
    PR4 = 0xffff;
}

void PWM_init() {
    // timer 2 configuration
    TMR2 = 0;
    PR2 = PERIOD;
    T2CON = 0x30; //use PBCLK as clock source, prescale 256, use 16-bit mode
    // PWM configuration
    OC1CON = 0x0;
    OC1R = 0;
    OC1RS = 0;
    OC1CON = 0x6; // use 16-bit timer 2, initialize as low

    T2CONSET = 0x8000;
    OC1CONSET = 0x8000;
}

void LCD_init() {
    T1CON = 0x0; //use PBCLK as clock source, prescale 1:1
    PR1 = 0xFFFF;

    DelayMsec(15); //wait for 15 ms
    RS = 0; //send command
    Data = LCD_IDLE; //function set - 8 bit interface
    DelayMsec(5); //wait for 5 ms
    Data = LCD_IDLE; //function set - 8 bit interface
    DelayMsec(2);
    Data = LCD_IDLE; //function set
    DelayMsec(5);
    Data = LCD_IDLE;
    DelayMsec(2);
```

```
    LCD_putchar(LCD_2_LINE_4_BITS);
    DelayMsec(2);
    LCD_putchar(LCD_DSP_CSR);
    DelayMsec(2);
    LCD_putchar(LCD_CLR_DSP);
    DelayMsec(5);
    LCD_putchar(LCD_CSR_INC);
}

void LCD_putchar(uchar c) {
    E = 1;
    Data = c; //sending higher nibble
    E = 0; //producing falling edge on E
    E = 1;
    Data <<= 4; //sending lower nibble through higher 4 ports
    E = 0; //producing falling edge on E
}

void GenMsec() {
    TMR1 = 0;
    T1CONSET = 0x8000;
    while (TMR1 < 500);
    T1CONCLR = 0x8000;
}
/* Call GenMsec() num times to generate num ms delay*/
void DelayMsec(uchar num) {
    uchar i;
    for (i=0; i<num; i++) {
        GenMsec();
    }
}
void LCD_goto(uchar addr) {
    addr = 0x80 + addr;
    RS = 0;
    E = 1;
    Data = addr; //sending higher nibble
    E = 0; //producing falling edge on E
    E = 1;
    Data <<= 4; //sending lower nibble through higher 4 ports
    E = 0; //producing falling edge on E
    DelayMsec(5);
}

void ADC_init(){
    TRISB=0x0; //config AN0 ~ AN15 as input
    TRISDCLR = 0x1; // set RD0 as light output
    AD1CON1 = 0xe4; //turn off ADC, 16bit integer output, auto convert, SAMP begins sampling
    AD1CON2 = 0x0; //use AVDD & AVSS, no scan, one Buffer, use MUX A
    AD1CON3 = 0x1F00; //use TPB, sample time 31TAD, TAD=2TPB
    AD1PCFG = 0x0; //all analog input
    AD1CHSbits.CHONA = 0x0; //MUX A negative input is VR-
    AD1CHSbits.CHOSA = 0x0; //MUX A positive input is AN0
    AD1CON1bits.ON=0x1;//turn on ADC
}

void print_result(int result) {
    RS = 1;
    float temp = result / 1024.0 * 3.3;
    uchar c1 = 0x30 + (int)temp / 1;
    uchar c2 = 0x30 + (int)(temp * 10) / 1 - (int)temp / 1 * 10;
    uchar c3 = 0x30 + (int)(temp * 100) / 1 - ((int)(temp * 10) / 1 - (int)temp / 1 * 10)*10 - (int)temp / 1 *
        100;
    LCD_putchar(c1);
    DelayMsec(5);
    LCD_putchar(0x2e);
    DelayMsec(5);
    LCD_putchar(c2);
    DelayMsec(5);
```

```
    LCD_putchar(c3);
    DelayMsec(5);
    LCD_goto(0x00);// keep the display in the same place
}

void UART_init() { // 8-bit data, no parity, 1 stop bit,
    U1BRG = 24; // Baud rate = 5000 at PBCLK = 500k and high speed
    U1AMODEbits.BRGH = 1; // High-Speed mode
    U1ASTAbits.UTXEN = 1; // Enable UART1 transmitter
    IEC0bits.U1ARXIE = 1; // Enable U1RX interrupt
    IFS0bits.U1RXIF = 0; // Clear UART1 receiver interruption bit
    IPC6bits.U1IP = 5; // Set priority 5
    IPC6bits.U1IS = 3; // Set sub-Priority 3
    U1ASTAbits.URXEN = 1; // Enable UART1 receiver

    U1AMODEbits.ON = 0; // Reset UART module
    U1AMODEbits.ON = 1; // Enable UART module
}

void UART_transmit(const uchar *s) {
    while (*s) {
        U1ATXREG = *(s++);
        while(!U1ASTAbits.TRMT);
    }
}

#pragma interrupt T3_handler ipl7 vector 12
void T3_handler() {
    IEC0CLR = 0x1000;
    total_time += PR3;
    if (total_time >= T) {
        total_time = 0;
        // it is time to release food
// OC1RS = SMALL;// default setting
        PWM_on();
        TMR3 = 0;
        while (TMR3 < 3900);// total releasing time: 2 seconds
// OC1RS = 0;// stop releasing
        PWM_off();
        TMR3 = 0;
    }
    IFS0CLR = 0x1000;
    IEC0SET = 0x1000;
}

#pragma interrupt UART_handler ipl5 vector 24
void UART_handler() {
    switch (U1ARXREG){
        case 0x11:
            motor_speed = SMALL;
            break;
        case 0x12:
            motor_speed = LARGE;
            break;
// case 0x13:
// PORTDbits.RD0 = ~PORTDbits.RD0;
//// PORTDbits.RD1 = ~PORTDbits.RD1;
// const uchar *msg_ptr = "Hello world";
// UART_transmit(msg_ptr);
// break;
        case 0x13:
            PWM_on();
            TMR4 = 0;
            T4CONSET = 0x8000;
            while(TMR4 < 3900);
            PWM_off();
            T4CONCLR = 0x8000;
            break;
```

```
        case 0x14:
            T = DEMO_T;
            break;
        case 0x15:
            T = REAL_T;
            break;
        case 0x21:
            return_ADC();
            break;
        case 0x22:
            return_Timer();
            break;
        case 0x23:
            return_PWM();
            break;
    }

    U1STAbits.OERR = 0;
    IFS0bits.U1RXIF = 0; // Clear UART receiver interruption bit
}

void PWM_on() {
    if (adc_flag) OC1RS = motor_speed;
}

void PWM_off() {
    OC1RS = 0;
}

void return_ADC() {
    const uchar *msg_adc_1 = "Not enough food remained.";
    const uchar *msg_adc_2 = "Enough food remained.";
    if (adc_flag) UART_transmit(msg_adc_1);
    else UART_transmit(msg_adc_2);
}

void return_Timer() {
    const uchar *msg_timer_1 = "The current period is 30 seconds.";
    const uchar *msg_timer_2 = "The current period is 8 hours.";
    if (T == DEMO_T) UART_transmit(msg_timer_1);
    else UART_transmit(msg_timer_2);
}

void return_PWM() {
    const uchar *msg_PWM_1 = "The current mode is slow mode.";
    const uchar *msg_PWM_2 = "The current mode is fast mode.";
    if (motor_speed == SMALL) UART_transmit(msg_PWM_1);
    else UART_transmit(msg_PWM_2);
}
```