華南理工大學
South China University of Technology

MCU Experiment Report

Taxi Fare Meter

Faculty	**School of Design**
Professional	**Product Design**
Student's Name	**Huang Zejun**
Student ID	**202130670058**
Instructor	**Zhu Yanzhao**
Start Date	**2025.2.28-2025.4.6**

I. Design Requirements

(1) Design a schematic diagram in Proteus that displays distance, speed, and total fare in real time.
(2) Use a single button to control the billing process: press to start billing, and release to stop billing and display the current distance and fare.
(3) Simulate taxi wheel signals using a signal generator.
(4) Submit the lab report and project files. A final evaluation will be conducted through a one-on-one defense with the instructor.
(5) Assume the circumference of the taxi tire is 1.83 meters. The fare is calculated as follows: ¥8 for the first 2 kilometers, and ¥2.6 per additional kilometer. Other charges are not considered.
(6) The program must be written in assembly language; C language is not allowed.
(7) The lab report must clearly explain the algorithm and describe the meaning and use of each memory variable.
(8) Carefully watch the reference video provided, and ensure your final result achieves the same functionality and appearance as shown in the video.

II. Design Overview

This project implements a simplified taxi fare meter based on the AT89C52 microcontroller. The system utilizes timer interrupts and external interrupts to respectively handle time-based counting and wheel pulse detection, enabling real-time calculation and 7-segment display of distance traveled, current speed, and total fare.

The overall architecture follows an "interrupt + main loop" structure. In the MAIN routine, buffer registers such as cache and count are first initialized, and Timer 1 is configured in Mode 1 (16-bit timer mode) with an initial value that generates an interrupt every 5ms. External interrupt 0 (INT0) is enabled to receive simulated wheel signals. Once started, the system enters a main loop that continuously responds to interrupts and processes data.

Each INT0 external interrupt corresponds to one full wheel rotation and increments the pulse counter. In the interrupt service routine (INT0_HANDLER), a flag TR_FLAG is also set to notify the main loop that new data is available for processing.

Timer 1 generates an interrupt every 5ms. Once it triggers 200 times (i.e., one second), the main loop calls CALC_SPEED to compute the current speed, and resets both cache and count to prepare for the next second.

In the main loop, if TR_FLAG is set, the system calls UPDATE_PATH to accumulate distance and UPDATE_PRICE to update the fare. Every second, CALC_SPEED is called to compute the real-time speed based on pulse count, which is then converted to BCD format for display.

All display data—including fare, kilometers, and speed—is stored in BCD format. The CONVERT_BCD routine separates each byte into individual digits and stores them in cDisplayBuffer. The Display routine then scans through 12 digits, looks up corresponding segment codes from DisplayTable, and drives the 7-segment display dynamically.

2.1 Speed

Speed is calculated based on the number of pulses received within a 1-second interval. Timer 1 generates an interrupt every 5ms, and a counter accumulates until 200 ticks indicate one second has passed. At that point, the main loop calls CALC_SPEED.

The theoretical formula used is: Speed = pulses × 1.83m × 3600s / 1000m = pulses × 6.588 (km/h)

Since this value can easily overflow standard registers, a fixed-point approximation is used. For instance, 100 pulses should yield 658.8 km/h. To simulate this in assembly without floating-point support, the value is approximated by splitting 6.588 into two factors: Speed ≈ pulses × 227 (E3H) + pulses × 65 (41H)

This is implemented via two successive multiplications in the program. The final binary result is stored in result, then converted to BCD using the BinDec routine for display.

2.2 Distance Calculation

Distance is calculated by counting wheel pulses via external interrupt INT0. Each pulse represents one full wheel rotation. In the interrupt service routine, the pulse count (cache) is incremented and TR_FLAG is set to notify the main loop.

The main loop detects the flag and calls UPDATE_PATH, which adds the equivalent distance of 1.83 meters per pulse to the BCD-based memory cells BCD+2 and BCD+3. Proper BCD carry operations and DA (Decimal Adjust) instructions ensure correctness. The final kilometer value is stored in the BCD region for display.

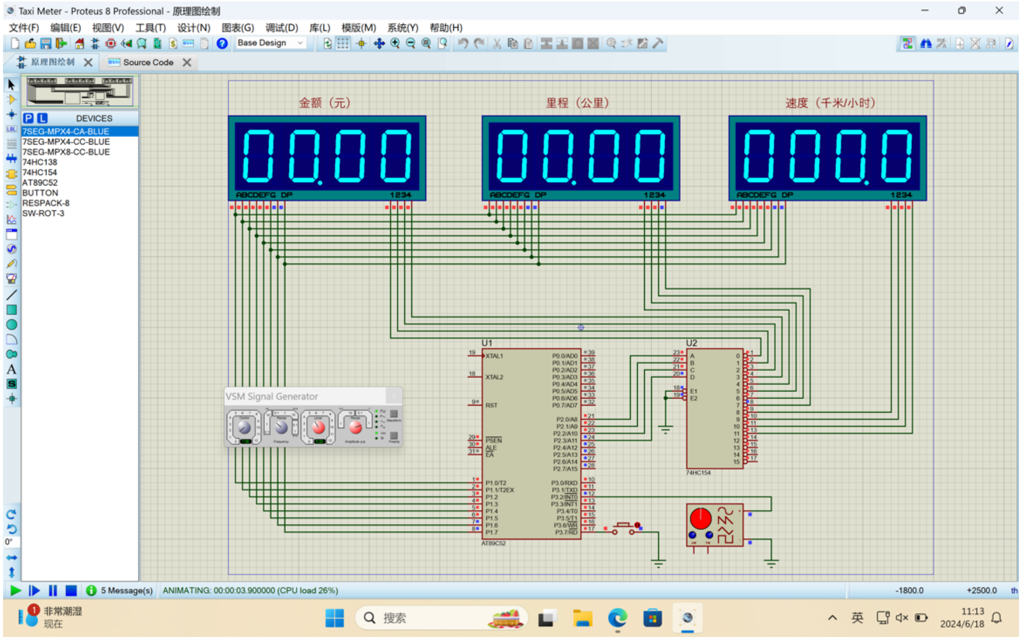
2.3 Fare Calculation

Fare is calculated based on the distance, with a base rate and tiered pricing. If the total distance is under 2 kilometers, the fare is fixed at ¥8. If the distance exceeds 2 kilometers, each additional increment (approximately 0.00183 km per pulse) is charged at ¥2.6 per kilometer.

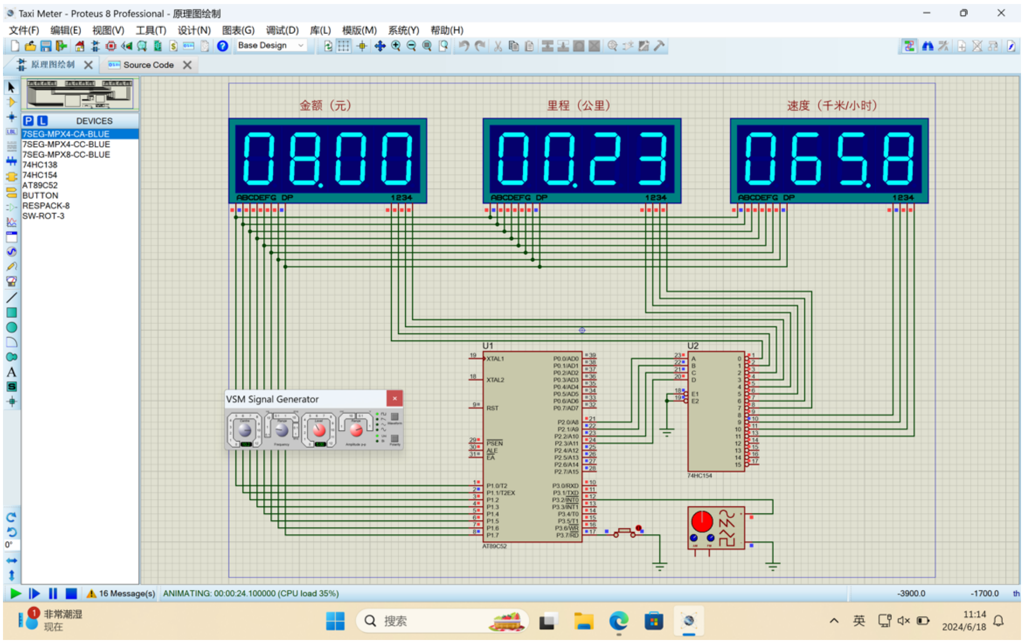
This logic is handled in the UPDATE_PRICE routine. Similar to distance accumulation, intermediate results are temporarily stored in cache+3 and cache+4, and then carried into the BCD area representing the fare (BCD, BCD+1) using BCD-compliant addition and adjustment. This allows the fare to be dynamically and accurately updated for display.

III. Screenshot of the work

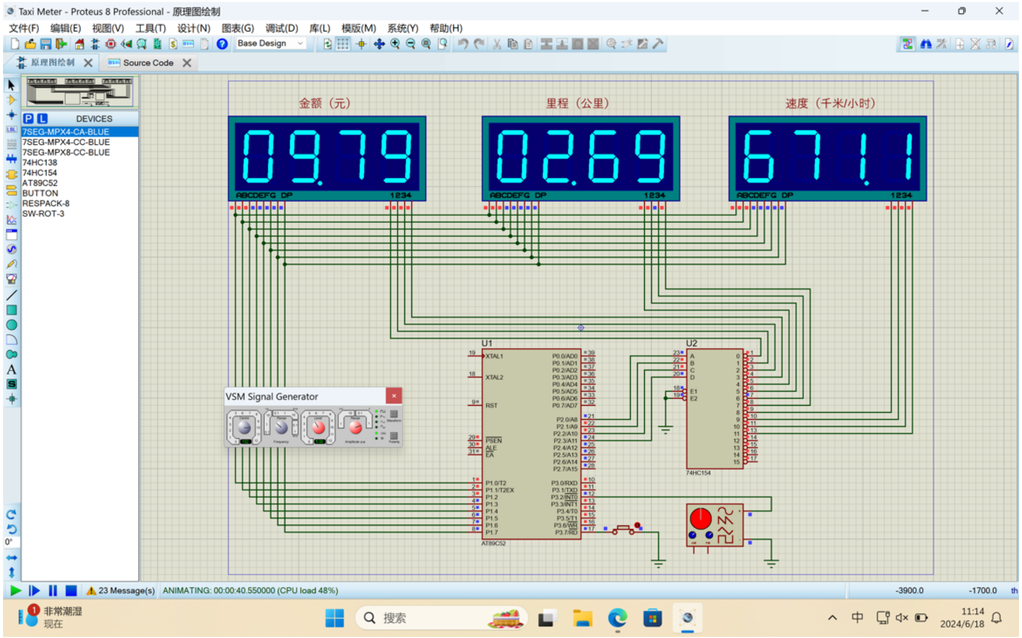
(1) Press the button to start billing



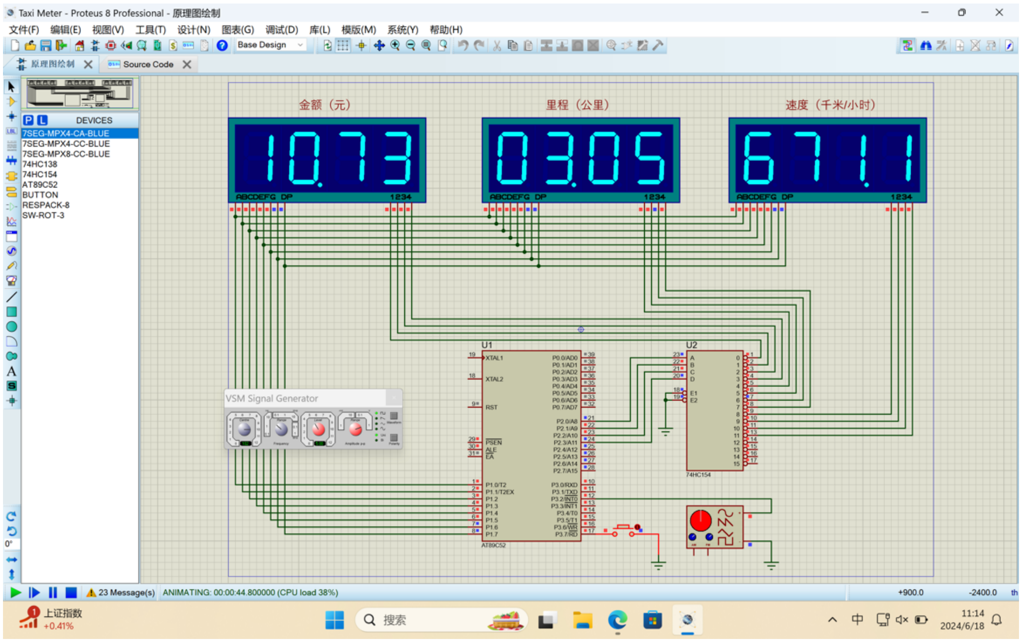
(2) Set the frequency to 10 and the speed will be displayed as 65.8



(3) Set the frequency to above 100 and quickly increase the mileage



(4) Release the button, stop billing, display current mileage and amount



IV. Code

```
ORG 0000H
SJMP MAIN
ORG 0003H
LJMP INT0_HANDLER
ORG 001BH
LJMP TIMER1_ISR

cDisplayBit    EQU 2CH
count         EQU 2DH
cache         EQU 2EH

bcd           EQU 40H
bcdBuf        EQU 46H
result        EQU 49H

cDisplayBuffer EQU 50H
TR_FLAG       BIT 00H
MARK          BIT 01H

MAIN:
   MOV cache,#00H
   MOV count,#00H
   MOV TMOD,#10H
   MOV TH1,#0DCH
   MOV TL1,#00H
   SETB EA
   SETB IT0
   SETB EX0
   SETB ET1
   SETB TR1
   LCALL CCLR
M1:
   JNB TR_FLAG,M2
   LCALL UPDATE_PATH
   LCALL UPDATE_PRICE
   CLR TR_FLAG
M2:
   LCALL SHOW
   MOV A,count
   XRL A,#200
   JNZ M1
```

```
   LCALL CALC_SPEED
   MOV count,#0
   MOV cache,#0
   SJMP M1

INT0_HANDLER:
   JB P3.7,RESET_MARK
   JB MARK,HANDLE_PULSE
   LCALL CCLR
   SETB MARK
HANDLE_PULSE:
   INC cache
   SETB TR_FLAG
   RETI
RESET_MARK:
   CLR MARK
   RETI

CCLR:
   MOV R1,#60H
   MOV R0,#20H
C1:
   MOV @R0,#00H
   INC R0
   DJNZ R1,C1
   RET

TIMER1_ISR:
   MOV TH1,#0DCH
   MOV TL1,#00H
   INC count
   RETI

CALC_SPEED:
   MOV A,cache
   MOV B,#0E3H
   MUL AB
   MOV result+1,B
   MOV B,#41H
   MOV A,cache
   MUL AB
   ADD A,result+1
   MOV result+1,A
   MOV A,B
```

```
    ADDC A,#0
    MOV result,A
    MOV R0,#result
    MOV R1,#bcdBuf
    LCALL BinDec
    MOV bcd+5,bcdBuf+2
    MOV bcd+4,bcdBuf+1
    RET

BinDec:
    CLR A
    MOV @R1,A
    INC R1
    MOV @R1,A
    INC R1
    MOV @R1,A
    PUSH 7
    MOV R7,#16
BD1:
    CLR C
    INC R0
    MOV A,@R0
    RLC A
    MOV @R0,A
    DEC R0
    MOV A,@R0
    RLC A
    MOV @R0,A
    PUSH 1
    MOV A,@R1
    ADDC A,@R1
    DA A
    MOV @R1,A
    DEC R1
    MOV A,@R1
    ADDC A,@R1
    DA A
    MOV @R1,A
    DEC R1
    MOV A,@R1
    ADDC A,@R1
    DA A
    MOV @R1,A
    POP 1
```

```
    DJNZ R7,BD1
    POP 7
    RET

UPDATE_PATH:
    CLR C
    MOV A,#30H
    ADDC A,cache+2
    DA A
    MOV cache+2,A
    MOV A,#18H
    ADDC A,cache+1
    DA A
    MOV cache+1,A
    MOV A,bcd+3
    ADDC A,#0
    DA A
    MOV bcd+3,A
    MOV A,bcd+2
    ADDC A,#0
    DA A
    MOV bcd+2,A
    RET

UPDATE_PRICE:
    MOV A,bcd+2
    CLR C
    SUBB A,#02H
    JC BASE_PRICE
    MOV A,#58H
    ADD A,cache+4
    DA A
    MOV cache+4,A
    MOV A,#47H
    ADDC A,cache+3
    DA A
    MOV cache+3,A
    MOV A,#00H
    ADDC A,bcd+1
    DA A
    MOV bcd+1,A
    MOV A,#00H
    ADDC A,bcd
    DA A
```

```
    MOV bcd,A
    RET
BASE_PRICE:
    MOV bcd,#08H
    RET

SHOW:
    LCALL CONVERT_BCD
    LCALL Display
    RET

CONVERT_BCD:
    MOV R5,#06H
    MOV R0,#bcd
    MOV R1,#cDisplayBuffer
BCD_SPLIT:
    MOV A,@R0
    ANL A,#0F0H
    SWAP A
    MOV @R1,A
    INC R1
    MOV A,@R0
    ANL A,#0FH
    MOV @R1,A
    INC R0
    INC R1
    DJNZ R5,BCD_SPLIT
    RET

DisplayTable: DB 3FH,06H,5BH,4FH,66H,6DH,7DH,07H,7FH,6FH

Display:
    MOV R5,#0CH
D1:
    LCALL Delay
    MOV A,cDisplayBit
    MOV P2,A
    MOV DPTR,#DisplayTable
    MOV A,#cDisplayBuffer
    ADD A,cDisplayBit
    MOV R0,A
    MOV A,@R0
    MOVC A,@A+DPTR
    MOV P1,A
```

```
    INC cDisplayBit
    DJNZ R5,D1
    MOV cDisplayBit,#00H
D2:
    LCALL Delay
    MOV P2,#01H
    MOV P1,#80H
    LCALL Delay
    MOV P2,#05H
    MOV P1,#80H
    LCALL Delay
    MOV P2,#0AH
    MOV P1,#80H
    RET

Delay:
    MOV R0,#10
    MOV R1,#10
    DJNZ R1,$
    DJNZ R0,$-4
    RET

END
```