Final Lab: ARM MCBSTM32C Finite State Machine with timers

Yang Du

46884803

**Overview**

The final project performs the behavior of a finite state machine using timers in the ARM MCBSTM32C development board. We use the case of a Mealy machine. Mealy machine, as one of the finite state machine, its output values are determined by the current state and the input value. Finite state machine (FSM) is an [abstract machine](https://en.wikipedia.org/wiki/Abstract_machine) that can be in exactly one of a finite number of [states](https://en.wikipedia.org/wiki/State_(computer_science)) at any given time. The FSM can change from one state to another in response to some external inputs; the change from one state to another is called a transition. This project has five states which are Neutral, Gear 1, Gear 2, Gear 3, and Gear 4. We use Timer to control how long it would stay in each state as well as four buttons (wake up, user, tamper, and reset) are used to change the input which can change the state of the system. The Fig.1 shows the state routine of the system.

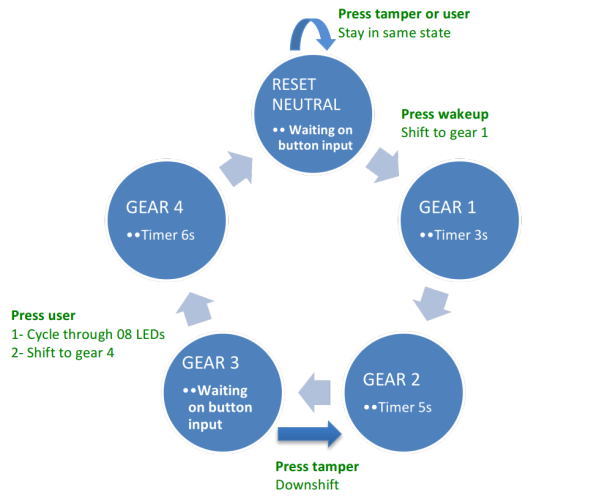


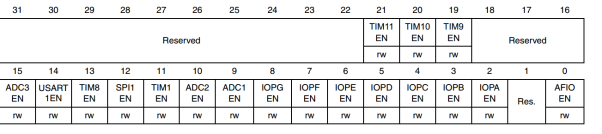
Fig.1 State diagram

In my code, **switch() case** is used to change the state, **TIMx->CNT** and **TIMx->CR1** control the time period, and **if()** estimates the input buttons. The following sections would illustrate exactly what does work in my design.

**GPIO configuration**

A General Purpose Input/output (GPIO) is an interface available on most modern microcontrollers (MCU) to provide an ease of access to the devices internal properties. Generally there are multiple GPIO pins on a single MCU for the use of multiple interaction so simultaneous application.

In this project, we need do configuration for LED and Timers which are connected to GPIOB and GPIOE. The Fig.2 shows the detail of APB2 peripheral clock enable register and Port configuration register. In this part, we need set GPIOE as output to control the LED. Otherwise, we have to make the four buttons which are connected to GPIOA(B C) as input. Moreover, we should initial the clock of the GPIO and AFIO. The AFIO is used for timer.



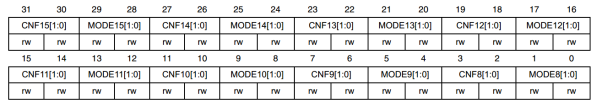


Fig.2 RCC\_APB2ENR and GPIOx-CRH

The following code is for GPIO configuration.

**/\*Enable GPIOA GPOIB GPIOC clock\*/**

RCC->APB2ENR |= 1 << 2; RCC->APB2ENR |= 1 << 3; RCC->APB2ENR |= 1 << 4;

**/\* Configure the GPIOA GPIOB GPIOC for Push Buttons \*/**

GPIOA->CRL &= 0xFFFFFFF0; GPIOA->CRL |= 0x00000004; GPIOB->CRL &= 0x0FFFFFFF; GPIOB->CRL |= 0x40000000;

GPIOC->CRH &= 0xFF0FFFFF; GPIOC->CRH |= 0x00400000;

**/\* Setup GPIO for LEDs, enable GPIOE and port configuration \*/**

RCC->APB2ENR |= 1 << 6; GPIOE->CRH = 0x33333333;

After the configuration of GPIO, we can use 08LED which are connected to GPIOE. In this project, there are two methods to blink LEDs which are read-modify-write and Atomic method. In this case, we use atomic method to cycle the LEDs.

for(num=0; num<8; num++)

{ GPIOE->BSRR = led\_mask[num]; for (i = 0; i < 500000; i++);

GPIOE->BSRR = led\_mask[num] << 16; for (i = 0; i < 500000; i++);

}

**Timer configuration**

Timers are peripherals that generate periodic events in a microcontroller. A timer can be used as a PWM generator, or a trigger for a motor, or to control an ADC. The main parts of a timer are: a counter-, a reload-, and a match-register.

In this part, we use general purpose Timers to perform the functionality that the system need. In this case, we use TIM4 as our timers to achieve 3s, 5s and 6s timers. First of all, this project has to make three different time period which are 3s, 5s, and 6s. Secondly, we have to use HIS that is the internal clock as our clock. The HIS clock is 8MHz clock. By setting Clock configuration register (RCC\_CFGR) to select HIS clock. Also the LSB of Clock control registers (RCC\_CR) can enable and disable HIS clock. Otherwise, SMCR should equal 000. 000: Slave mode disabled - if CEN = ‘1’ then the pre-scalar is clocked directly by the internal clock. Thirdly, we should consider setting the smallest time period as our time unit. Assuming our time unit is 1 ms, we should set the value in PSC register to make sure its frequency is 1/1 ms = 1000Hz. According to the following formula

C:\Users\duy\AppData\Local\Temp\1494096135(1).png

I get the value of PSC register is 7999 in decimal. The following code illustrates the configuration of the Timer 4.

**/\* HSI selected as system clock \*/**

RCC-> CFGR &= 0000;

**/\* enable HSI clock \*/**

RCC->CR |= 1<<0;

**/\* Slave mode selection 000 \*/**

TIM4->SMCR &= (000 << 0);

**/\* TIM4 prescaler 7999 \*/**

TIM4->PSC &= 0x0000;

TIM4->PSC |=0x1F3F << 0;

**/\* counter initialize \*/**

TIM4->CNT &=0x0000;

After the configuration, out time unit is 1 ms. Therefore, we use TIM4->CNT register to perform timers. In this case, we use count up which is achieved in bit 4 of TIMx\_CR1. Also, Counter value is configured to 3000 for accomplish 3s timer.

**Functionality claims**

I have achieved all the functionality in this project, the following form would be as a checklist to shows which function that I have implemented.

According to the form above, we achieve full functionality in this design.

|  |  |  |
| --- | --- | --- |
|  | Project requirements | Design implementation |
| Neutral  State | Press tamper or user Stay in same state | Valid |
| Press wakeupshift to gear 1 | Shift to Gear 1 |
| Gear 1 | Timer 3s | Timer 3s then shift to Gear 2 |
| Gear 2 | Timer 5s | Timer 5s then shift to Gear 3 |
| Gear 3 | Press tamper shift to Gear 2 | Valid |
| Press user  LEDs blink and shift to Gear 4 | 08 LED blink in a cycle then shift to Gear 4 |
| Gear 4 | Timer 6s | Timer 6s then shift to Neutral state |

It can repeat by the cycle in Fig.1, and it shows that both current state and current input can affect the result of states.

**Conclusion**

In this report, the implementation of state machine is presented. The next state can be affected by both current state and input value. We use function **switch ()** and **case** to determinate the entry of each state. Also, we use function **if ()** and **else** to estimate the value of the counter whether meets the time we need. Function **if ()** is used to estimated the buttons as well. Otherwise, at the beginning and end of each state, we empty the value in the counter register to make sure its value equals zero at next counting task. Moreover, we also need unable the HIS clock when one state is expired, on the other hand, we should enable the HIS clock at the beginning of the counting task. It can assure the accuracy of the clock. As for display task, I modified several times, because we need clean the line sometimes to prevent the overlay of the characters. However, I failed in using function **GLCD\_ClearLn()**. Therefore, for achieving cleaning the line in LCD, I use function **GLCD\_DisplayString()** by inserting the blank.

I have understood how to use the timer registers in this project and the interfacing of MCBSTM32C. For LCD display, I learned how to troubleshoot when it does not display the right information. Also, I understand function if() and else in the usage of estimating buttons.

In conclusion, this project accomplished all the requirements. It can change states by input and timers accurately.