I 3 bits LED Pendulum

Contreras R Orlando[1,2*] and Lara H Kevin[1,2*]

[3]Department of Electronic Systems, UAA, Av. Universidad 940, Aguascalientes, 20131, Aguascalientes, México.

*Corresponding author(s). E-mail(s): {al348390,al464376}@edu.uaa.mx;

Abstract

The objetive of this lab practice is to create a 3/8 bits LED pendulum in C language on the STM32F446ZET6 board, the main purpose is to handle configuration and delays.

Keywords: C, ARM, Shift, Registers, Vector

1 Introduction

In this practice we will develop an algorithm in c to shift 3 leds in a ledbar of 8 bits, in adition we will implement two buttons to change the speed. We use a vector with all of the hex values to send them to ouput port .

2 Objetives

- Design a code using vectors and for cycles with variable speed buttons (+/-).
- Correct use of GPIO .
- Implement an effective polling.
- Develop an optimiced algorithms for space and time.

3 Method

3.1 Pseudocode

Here is the pseudo code that represents the algorithm, also it is included the configuration as masks or

Algorithm 1 GPIO Configuration Algorithm

Input: None

Output: GPIO configuration for LED outputs and button inputs

```
confGPIO() // Configure GPIOE pins PE8-PE15 as output
mask ← 0x5555 ≪ 16    // Binary: 0101 0101 0101 0101 shifted left by 16 bits
GPIOE_MODER ← GPIOE_MODER OR mask  // Set MODER bits 01 (output) for pins 8-15
// Configure GPIOA pins PA5 and PA6 as input with pull-down resistors
mask ← 0xA ≪ 10    // Binary: 1010 shifted left by 10 bits (bits 10-13)
GPIOA_PUPDR ← GPIOA_PUPDR OR mask            // Set PUPDR bits 10 (pull-down) for PA5 and PA6
```

Algorithm 2 LED Control with Pattern Movement and Speed Adjustment

Input: None

Output: LED control with predefined pattern movement and button-adjustable speed

```
confRCC() confGPIO()
delay_val ← 50                                                    // Initial delay value
Define veclit[19] ← {0x3800, 0x1C00, 0x0E00, 0x0700, 0x0300, .., 0x7000}    // LED pattern array
while true do
    for i ← 0 to 17 do
        GPIOE_ODR ← veclit[i]                  // Output current pattern to LEDs
                                                                  // Check button inputs
        if GPIOA_IDR & (1 ≪ 5) and delay_val < 100 then
            delay_val ← delay_val + 1                  // Increase delay (slow down)
        if GPIOA_IDR & (1 ≪ 6) and delay_val > 10 then
            delay_val ← delay_val − 1                   // Decrease delay (speed up)
        delay_ms(delay_val)
```

3.2 Register Address

- **RCC**: Base address of clock control register.
- **RCC_AHB1ENR**:Register used to enable port clocks.
- **Registers MODER, PUPDR**: Used to configurate GPIOs.
- **Registers IDR y ODR**: Input and ouput data registers .

3.3 System Initialization

The main code begins by enabling peripheral's clock by the `RCC_AHB1ENR` register.

3.4 Configuration of GPIO

- The pins PE8 to PE15 are in output configuration.
- The pins PA5 and PA6 are configurated on input and pull-down mode.

3.5 Vector

Here's how we get the vector using excel [Figure 2], we had to add to 0's after the hex because we use the pins from 15 to 8

V[i]	7	6	5	4	3	2	1	0	HEX	PORT
0	0	0	0	1	1	1	0	0	1C	0x1C00,
1	0	0	0	0	1	1	1	0	0E	0x0E00,
2	0	0	0	0	0	1	1	1	07	0x0700,
3	0	0	0	0	0	0	1	1	03	0x0300,
4	0	0	0	0	0	0	0	1	01	0x0100,
5	0	0	0	0	0	0	1	1	03	0x0300,
6	0	0	0	0	0	1	1	1	07	0x0700,
7	0	0	0	0	1	1	1	0	0E	0x0E00,
8	0	0	0	1	1	1	0	0	1C	0x1C00,
9	0	0	1	1	1	0	0	0	38	0x3800,
10	0	1	1	1	0	0	0	0	70	0x7000,
11	1	1	1	0	0	0	0	0	E0	0xE000,
12	1	1	0	0	0	0	0	0	C0	0xC000,
13	1	0	0	0	0	0	0	0	80	0x8000,
14	1	1	0	0	0	0	0	0	C0	0xC000,
15	1	1	1	0	0	0	0	0	E0	0xE000,
16	0	1	1	1	0	0	0	0	70	0x7000,

Fig. 1 Vector of Ledbar

3.6 Control Logic

The code increments a counter with a for cycle and index a value in an array each value is a port value, this would do a left-right pendulum.

3.7 Functions

- **confRCC**: Set the GPIO clocks .
- **confGPIO**: Set input and output ports.
- **delayms**: Do a delay of 1 ms .

4 Results Discussion

4.1 Code

Este es el código original en ensamblador que corresponde al pseudocódigo anterior:

```
// ------- Main Library -------
#include <stm32f446xx.h>

// --------- Function ---------
void confRCC(void);
void confGPIO(void);
void delay_ms(uint32_t delay);
// ---------- Class ----------
uint16_t veclit [19] = {0x3800,0x1C00,0x0E00,0x0700,0x0300,
0x0100,0x0300,0x0700,0x0E00,0x1C00,0x3800,0x7000
,0xE000,0xC000,0x8000,0xC000,0xE000,0x7000};
// -------- Variables --------
// ----------- Main -----------

int main(void){
  confRCC();
  confGPIO();
  int delay_val = 50;
```

```
  while(1){
  for(uint8_t i = 0;i<18;i++){
  GPIOE->ODR = veclit[i];

  if(GPIOA->IDR  & (1 << 5) && delay_val < 100)
  {
  delay_val = delay_val + 1;
  }
  if(GPIOA->IDR  & (1 << 6) && delay_val >10)
  {
  delay_val = delay_val - 1;
}
  delay_ms(delay_val);
  }

  }}
/*
  if(GPIOA->IDR & (1<<0)){}
  lecPA0 = (uint8_t)GPIOA-IDR;
  if(lecPA= & (1<<0))
*/

void confRCC(void){
  RCC->AHB1ENR |= (1 << 0) | (1 << 4); // A  E
}

void confGPIO(void){
  GPIOE->MODER |= (0x5555 << 16); // PE8 -> PE15 Output
  GPIOA->PUPDR |= (0xA << 10); //PA5 y PA6 como entrada PA5  + PA6 -
}
// 3 bits en 8
// Input PA5 PA6

void delay_ms(uint32_t delay){
  for(uint32_t i=0;i<delay;i++){
  for(uint32_t j=0;j<11500;j++);
  }
}
```

4.2 Wiring Diagram

Here's our circuit implementation, you can notice that we use a resistor bar of 220 Ω with a ledbar.

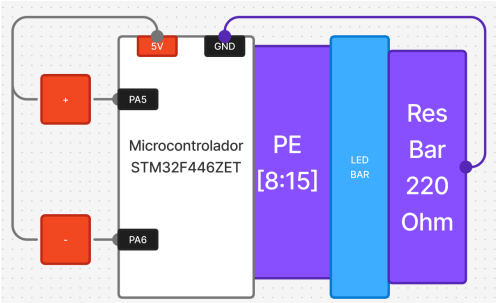

Fig. 2 Vector of Ledbar

4.3 Engineering Evaluation

Q1: How did you implement debouncing and why?
Debouncing was implemented using a delay of 10 ms.

Q2: Why use a `mask << pos` rather than shifting the pattern directly to maintain a window of N LEDs?
We didn't use in the code implementation because we thought that would be easier with a vector

Q3: How would you sync the effect to an external clock/PWM?
Synchronization can be achieved by implementing in an ADC pin and potetiometer. With a ADC can be readed and with a PWM can be taked out .

Q4: What impact does keeping multiple LEDs on have on current consumption? How do you reduce it?
Keeping multiple LEDs on increases current consumption, we would implement a bigger resistor to reduce it,

Q5: How to convert button polling to EXTI-based interrupts?
We could do by an EXTI setting the required configuration by enabling the interruptions on the pin. This would be more efective and optimiced for timing, also

Q6: What problems arise if selected pins are multiplexed for other peripherals (ADC, UART)? How to avoid it?
It could be posible but should be implemented by increasing the clock due to timing errors, also would be necesary to do multiple mask.

5 Conclusion

This practice wasn't so hard because we already are related to high level languages, also were less instructions that in assembly, the configuration part was the same that in assembly. There wasn't any complications than typos.