Detail of

Omni-directional Quadruped Robot

Catalogue

Overview

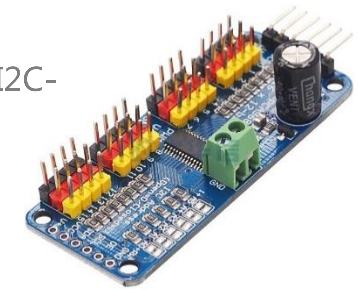
- **3D Printed Body**

- **180° MG90S servo**

  Each leg 3 servos.
  Advantage: more stable than SG90: metal gear.

- **PCA9685**

  PCA9685 16-channel, 12-bit PWM Fm+ I2C-bus LED controller.



- **Rocking bar**

  Working principle similar to VR.
  Get (X,Y) value from 0 to 4095 by ADC.
  Disadvantage: not accurate.

- **Gait control algorithm**

- **Able to move in 8 directions currently**

Design flow: Servo-Control with Timer

Intuition:

3~4 Timers, 12 Channels, Same pre-scale and period

Generate 12 PWM signal to control 12 servo.

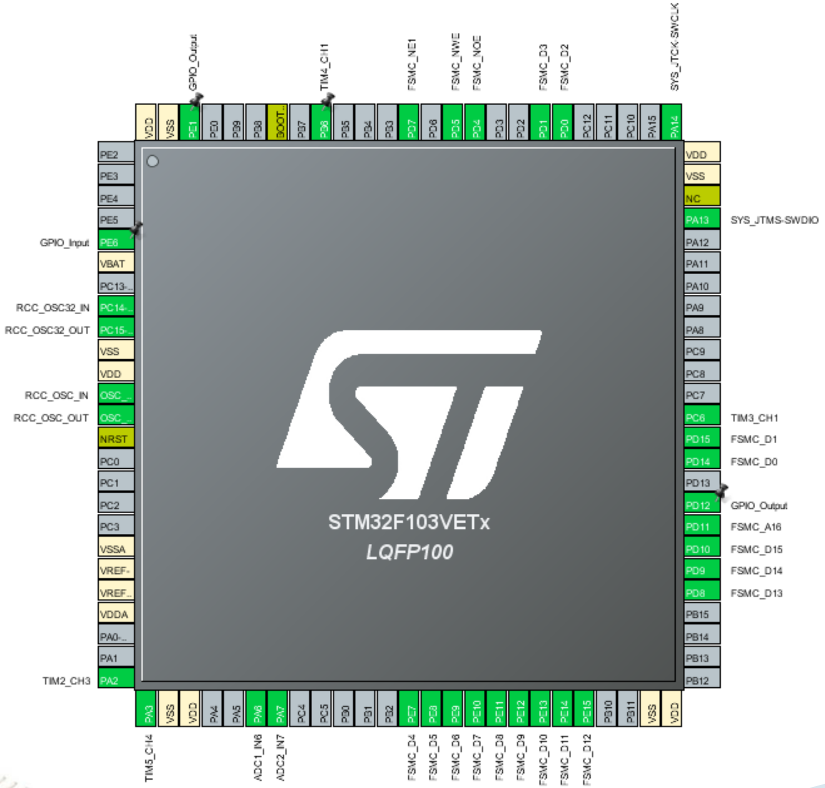
Problem: In test mode, when using more than one timer, the program will finalize in the line: HardFault_handler() ---->empty while(1) loop.

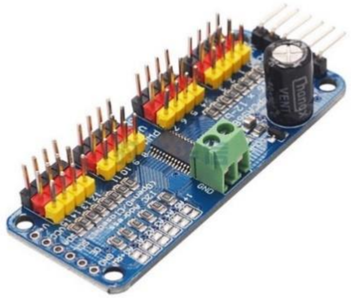
Our Solution:

1. Power supply not enough: using lab power supplier
2. No timer interrupt: tried according to instruction online.

Decision:

1. Use PCA9685 16-channel, 12-bit PWM Fm+ I2C-bus LED controller.





Design flow: Servo-Control with Timer

PCA9685

16-channel, 12-bit PWM Fm+ I²C-bus LED controller

Rev. 4 — 16 April 2015 Product data sheet

1. General description

The PCA9685 is an I²C-bus controlled 16-channel LED controller optimized for Red/Green/Blue/Amber (RGBA) color backlighting applications. Each LED output has its own 12-bit resolution (4096 steps) fixed frequency individual PWM controller that operates at a programmable frequency from a typical of 24 Hz to 1526 Hz with a duty cycle that is adjustable from 0 % to 100 % to allow the LED to be set to a specific brightness value. All outputs are set to the same PWM frequency.

Each LED output can be off or on (no PWM control), or set at its individual PWM controller value. The LED output driver is programmed to be either open-drain with a 25 mA current sink capability at 5 V or totem pole with a 25 mA sink, 10 mA source capability at 5 V. The PCA9685 operates with a supply voltage range of 2.3 V to 5.5 V and the inputs and outputs are 5.5 V tolerant. LEDs can be directly connected to the LED output (up to 25 mA, 5.5 V) or controlled with external drivers and a minimum amount of discrete components for larger current or higher voltage LEDs.

The PCA9685 is in the new Fast-mode Plus (Fm+) family. Fm+ devices offer higher frequency (up to 1 MHz) and more densely populated bus operation (up to 4000 pF).

Although the PCA9635 and PCA9685 have many similar features, the PCA9685 has some unique features that make it more suitable for applications such as LCD or LED backlighting and Ambilight:

- The PCA9685 allows staggered LED output on and off times to minimize current surges. The on and off time delay is independently programmable for each of the 16 channels. This feature is not available in PCA9635.
- The PCA9685 has 4096 steps (12-bit PWM) of individual LED brightness control. The PCA9635 has only 256 steps (8-bit PWM).
- When multiple LED controllers are incorporated in a system, the PWM pulse widths between multiple devices may differ if PCA9635s are used. The PCA9685 has a programmable prescaler to adjust the PWM pulse widths of multiple devices.
- The PCA9685 has an external clock input pin that will accept user-supplied clock (50 MHz max.) in place of the internal 25 MHz oscillator. This feature allows synchronization of multiple devices. The PCA9635 does not have external clock input feature.
- Like the PCA9635, PCA9685 also has a built-in oscillator for the PWM control. However, the frequency used for PWM control in the PCA9685 is adjustable from about 24 Hz to 1526 Hz as compared to the typical 97.6 kHz frequency of the PCA9635. This allows the use of PCA9685 with external power supply controllers. All bits are set at the same frequency.
- The Power-On Reset (POR) default state of LEDn output pins is LOW in the case of PCA9685. It is HIGH for PCA9635.

I2C COMMUNICATION

Similar to Lab6

01 Set output frequency

Write per-scale to register 254.

Internal 25 MHz oscillator.

254	FE	1	1	1	1	1	1	1	0	PRE_SCALE[1]	read/write

```c
void pca_setfreq(float freq)
{
  uint8_t prescale, oldmode, newmode;
  double prescaleval;
  prescaleval = 25000000; // internal clock with 25MHz
  prescaleval /= 4096;
  prescaleval /= freq;
  prescaleval -= 1;
  prescale = floor(prescaleval + 0.5f);

  HAL_I2C_Mem_Write(&hi2c2, 0x80, pca_model, 1, &oldmode, 1, 100);
  newmode = (oldmode & 0x7F) | 0x10; // sleep
  HAL_I2C_Mem_Write(&hi2c2, 0x80, pca_model, 1,&newmode, 1, 100); // go to sleep
  HAL_I2C_Mem_Write(&hi2c2, 0x80, pca_pre, 1, &prescale, 1, 100); // set the prescaler
  HAL_I2C_Mem_Write(&hi2c2, 0x80, pca_model, 1, &oldmode, 1, 100);
  HAL_Delay(2);
  newmode = (oldmode | 0xa1);
  HAL_I2C_Mem_Write(&hi2c2, 0x80, pca_model, 1, &newmode, 1, 100);
}
```

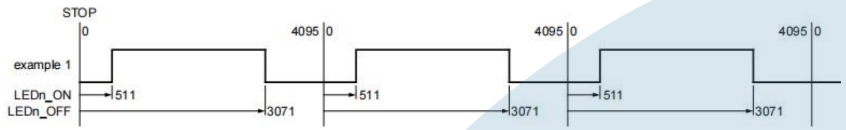
02 Set PWM

4 registers for each PWM:

LEDn_ON_H, LEDn_ON_L

LEDn_OFF_H, LEDn_OFF_L

```c
void pca_setpwm(uint8_t num, uint32_t on, uint32_t off)
{
  uint8_t LED_ON_L = on, LED_ON_H = on>>8;
  uint8_t LED_OFF_L = off, LED_OFF_H = off>>8;
  HAL_I2C_Mem_Write(&hi2c2, 0x80, LED0_ON_L + 4*num, 1, &LED_ON_L, 1, 100);
  HAL_I2C_Mem_Write(&hi2c2, 0x80, LED0_ON_H + 4*num, 1, &LED_ON_H, 1, 100);
  HAL_I2C_Mem_Write(&hi2c2, 0x80, LED0_OFF_L + 4*num, 1, &LED_OFF_L, 1, 100);
  HAL_I2C_Mem_Write(&hi2c2, 0x80, LED0_OFF_H + 4*num, 1, &LED_OFF_H, 1, 100);
}
```



Design flow: Gait with Thread-Scheduling

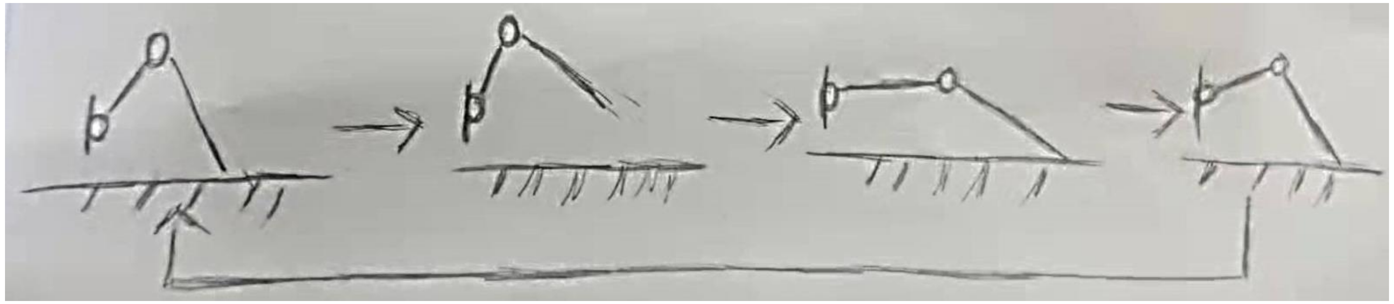
Coding

Initial state.

Ready state.

Walking state.

4 stage in each step loop. →



Define 3 most fundamental movements:

Direction: rotate forward or backward.

Stride: Stretching or putting back .

Height: lift up or put down.

Combined them to achieve each stage in gait.

Label servos and **map** them with actual PWM channel.

Expand one direction to all other directions by **swapping** servos` label.

Use same algorithm, but **different parameter sets**.

Design flow: Gait with Thread-Scheduling

uCOSiii: concurrency of legs

01 **System tick** for timing, counting on time when a systick interrupt is triggered

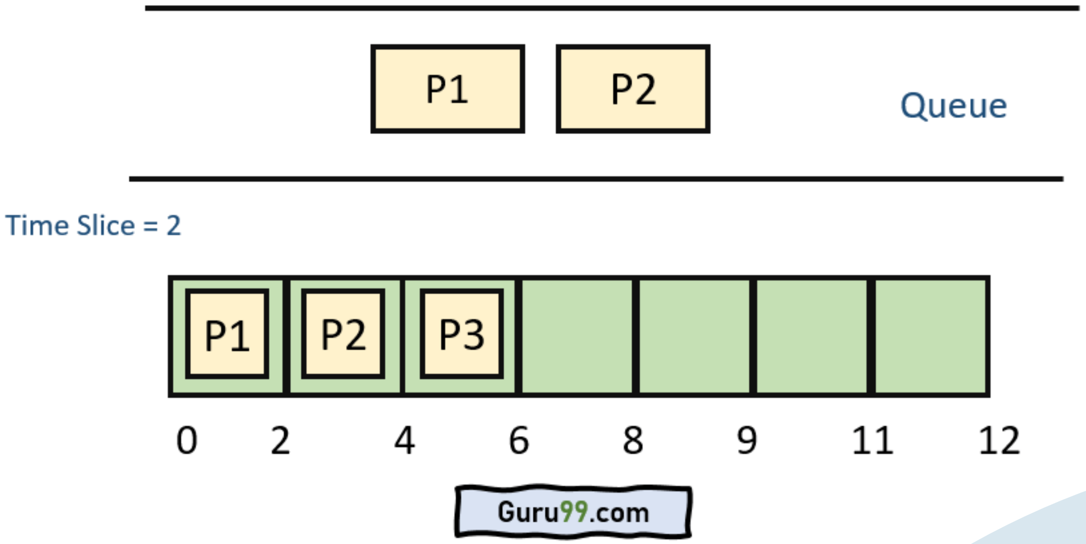
```c
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  /* USER CODE BEGIN SysTick_IRQn 1 */
  CPU_SR_ALLOC();


  //CPU_CRITICAL_ENTER();
  //OSIntNestingCtr++;
  //CPU_CRITICAL_EXIT();


  if(OSRunning==1)
  {
    OSIntEnter();
    OSTimeTick();
    OSIntExit();
  }

  /* USER CODE END SysTick_IRQn 1 */
}
```

02 **PendSV_handler()** to suspend context switch if there is interrupt haven`t been handled yet

```asm
PendSV_Handler
    CPSID   I                                   ; Prevent interruption during context switch
    MRS     R0, PSP                             ; PSP is process stack pointer
    CBZ     R0, OS_CPU_PendSVHandler_nosave     ; Skip register save the first time

    SUBS    R0, R0, #0x20                       ; Save remaining regs r4-11 on process stack
    STM     R0, {R4-R11}

    LDR     R1, =OSTCBCurPtr                    ; OSTCBCurPtr->OSTCBStkPtr = SP;
    LDR     R1, [R1]
    STR     R0, [R1]

OS_CPU_PendSVHandler_nosave
    PUSH    {R14}                               ; Save LR exc_return value
    LDR     R0, =OSTaskSwHook                   ; OSTaskSwHook();
    BLX     R0
    POP     {R14}

    LDR     R0, =OSPrioCur                      ; OSPrioCur   = OSPrioHighRdy;
    LDR     R1, =OSPrioHighRdy
    LDRB    R2, [R1]
    STRB    R2, [R0]

    LDR     R0, =OSTCBCurPtr                    ; OSTCBCurPtr = OSTCBHighRdyPtr;
    LDR     R1, =OSTCBHighRdyPtr
    LDR     R2, [R1]
    STR     R2, [R0]

    LDR     R0, [R2]                            ; R0 is new process SP; SP = OSTCBHighRdyPtr->StkPtr;
    LDM     R0, {R4-R11}                        ; Restore r4-11 from new process stack
    ADDS    R0, R0, #0x20

    MSR     PSP, R0                             ; Load PSP with new process SP
    ORR     LR, LR, #0x04                       ; Ensure exception return uses process stack
    CPSIE   I

    BX      LR                                  ; Exception return will restore remaining context
```

03 **Round Robin algorithm** to run tasks alternately

5 tasks, switch on every 2ms



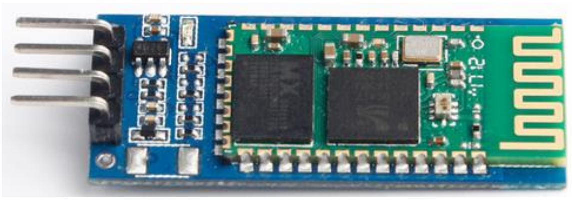
Further Development

Gyro sensor

Balance under normal circumstances

Distance sensor

Obstacle detection
Tracking



Bluetooth module

Remote control

THANKS

DING, Ziheng Chen, Siyu