Control of Multiple PWM Servos by a Single Programmable Timer

Michael Collier(高理颢), Jin-Wei SUN(孙进伟)
(*College of Information and Electrical Engineering, Shandong University of Science & Technology, Qingdao* 266510, *China*)

Abstract – An algorithm for control of several servo motors by a microcontroller is presented. The limited number of programmable timers on the majority of microcontrollers presents a problem for multiple generation of timing pulses. Two software approaches are discussed in the paper and experimental results given for operation of a set of small servos using a single timer.

Key words – *PWM*; *servo*; *multiple*; *timer*

Manuscript Number: 1674-8042(2011)01-0060-04

dio: 10.3969/j.issn.1674-8042.2011.01.15

1 Introduction

The control of small servo motors for rotational positioning is being increasingly implemented by means of microcontrollers. Such a servo is most commonly controlled by a single wire carrying a Pulse-Width-Modulation (PWM) signal.

In cases where it is desired to control several servos from one microcontroller, the hardware interfacing is extremely simple since each motor can be connected to a separate port pin. However, the software production of the necessary variable pulse widths demands the use of the programmable timers on the microcontroller. These being limited in number (usually to two or three) the control of multiple servos becomes more complicated. This paper describes an algorithm to meet this situation.

The need for this development has arisen as part of a larger project to implement fuzzy-logic control of sailing vessels, and the development work is being undertaken on small-scale model sailing yachts to facilitate experimentation in the testing of the fuzzy algorithms. Among the requirements of such a system is the need to mechanically control several facilities on the boat from a single microcontroller. These operations concern the manipulation of the sails and rudder of the boat in response to changing wind conditions, and are derived from measurements of compass heading and relative wind direction.

2 PWM servo principles

Small servo motors usually have a three-wire connection, comprising positive supply, ground and signal pins. Position control is achieved by variation of pulse lengths over an active range. A typical servo, the S3003 from Futaba, used widely for model control applications is shown in Fig. 1. This motor[1] has been chosen for the fuzzy logic control project.

Fig. 1 Futaba S3003 servo motor

The arrangement is particularly suitable for microcontroller operation, since only one digital output is required and the signal voltage comprises accurately-timed pulses derived from the software. Referring to Fig. 2, it can be seen that the motor shaft of the S3003 moves between its angular extremities for a change of pulse width from approximately 500 μs to 1500 μs.

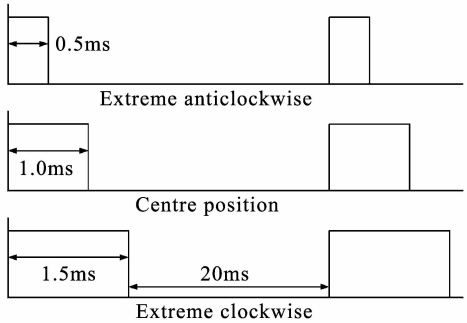

Fig. 2 Pulse widths for various positions of the S3003

The spacing between pulses is nominally 20 ms, but measurement has shown that this is not critical, with spacings from 6 ms to 42 ms producing acceptable performance.

Generation of appropriate pulse trains can be imple-

* **Received**: 2010-08-11
Corresponding author: Michael Collier(collier1942@yahoo.co.uk)

mented by software loading and enabling of one of the microcontroller on-chip timers.

Experimental measurements for the linearity of the angular movement against pulse width are shown in Fig.3, indicating that the relationship has a high degree of proportionality.

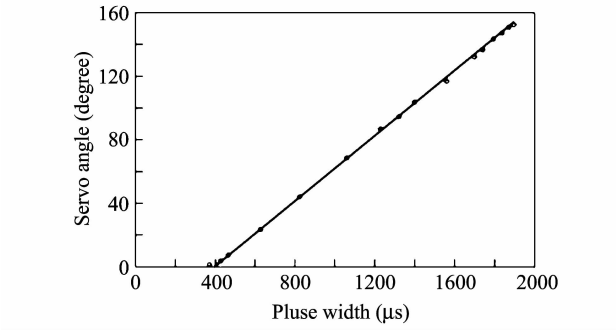

Fig.3 Angular position against pulse width for the S3003

3 Methods of multi-servo control

3.1 Previously proposed methods

Several ways of achieving multiple control have been reported, ranging from personal computer connection[2], PLCs[3], special interface boards[4], and the use of multiple timers[5]. The present project required a single timer, since the other timers on the chip were already bespoke for other processes.

3.2 Sequential pulse generation

This method creates the pulses for the motors at distinct non-overlapping times, as shown in Fig.4. The microcontroller keeps a table of the pulse times currently required by the individual motors, and uses the timer interrupt to determine the time positions of the pulses.

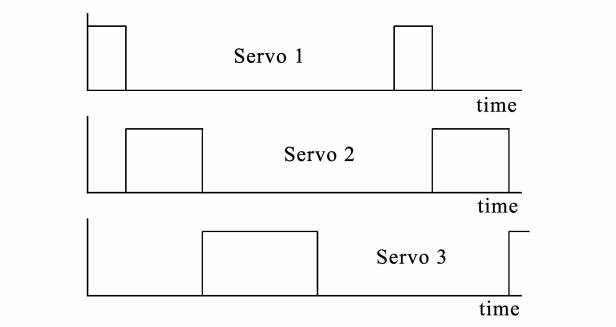

Fig.4 Time diagrams for the sequential method

When the timer interrupt service routine is called by the overflow of the timer/counter, the software of the ISR will terminate the pulse for one motor, and immediately reload the timer to start the pulse for the next motor in the cyclic chain. This reload value is read from the pulse time table which is being constantly updated by other processes running on the hardware.

3.3 Simultaneous pulse generation

A second method of control is to start all the pulses to the various motors at the same time. Whenever a timer overflow occurs the ISR will recalculate the time remaining until the end of the next pulse. Fig.5 shows the timing arrangement.

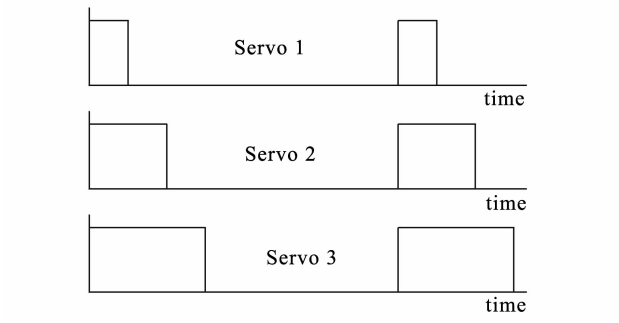

Fig.5 Timing diagrams for the simultaneous method

4 Software design and implication

4.1 Program structure

The process for controlling the servo motors is intended to form part of a multitasking system. The whole software package is programmed in the C51 language, using global variables for inter-process communication. The inputs for the servo control module take the form of arrays of 8-bit variables which are constantly being updated by other processes on the system. The programs have been written for control of five servo motors, but can be extended to more.

The whole servo control process is entirely interrupt-driven, once the initialisation function has set the timer running.

4.2 C51 code for the sequential method

```
unsigned char pulsetime[5];
                //Updated from separate process
unsigned char current_servo=0;      //Start with servo 0
void main()
{
    P1=0x00;        //All output lines low
    EA=1;
    TMOD=0x01;        //Use Timer 0 in mode 1
    ET0=1;
    TH0=(65536-pulsetime[current_servo]*9+603)/
256;
    TL0=(65536-pulsetime[current_servo]*9+603)%
256;
    P1=0x01;          //Start pulse for servo 0
    TR0=1;          //Start timer
    while(1);         //Initialisation complete
}
```

```
timer0_ISR() interrupt 1
{
    TF0=0;
    if (current_servo=4)    //After all servos 20 ms space
    {
      TH0=(65536-301150)/256;    //Preload for 20 ms
      TL0=(65536-301150)%256;
      current_servo=99;    //Indicate a 20 ms space
      P1=0x00;
      TR0=1; //Start 20 ms space
    }
    else
    {
      if (current_servo=99)
      {
        current_servo=0;    //Reset for servo 0
        P1 = 0x01;
      }
      else
      {
        P1=P1<<1;        //Move to next output pin
        current_servo++;
      }
      TH0=(65536-pulsetime[current_servo]*9+603)/256;
      TL0 = (65536-pulsetime[current_servo]*9+603)%256;
      TR0=1;        //Restart timer
    }
}
```

4.3 Algorithm for the simultaneous method

The procedure is similar to that shown for the previous method, with the modification that all the pulses are started at the same time, and the ISR reloads the timer with the lowest value of the remaining pulse times. This "shortest-pulse-first" strategy causes the interrupts to occur at each of the required pulse trailing edges, thereby terminating the pulses at the correct times.

The timer ISR is programmed to perform the following:

(1) Make the output pin to the current servo low;

(2) Subtract the previous pulse time value from all the values in the cycle time table;

(3) Remove from the table any values that have reached zero;

(4) Make the servo with the shortest time to be the current servo;

(5) Reload the timer with the remaining pulse time of the current servo;

(6) If all pulse times have reached zero, then reload the pulse time table, and load the timer for 20 ms;

(7) Restart the timer.

Fig.6 shows the timer reload values at each stage of the operation.

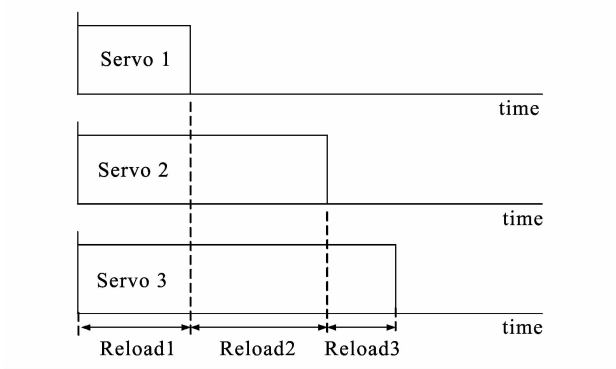

Fig.6 Time reload values for the sequential method

4.4 Analysis of the algorithms

An inherently possible problem with the sequential algorithm is that a large number of servos might produce a low pulse repetition rate for each motor. The pulse spacing should nominally be 20 ms, but would obviously increase with the number of devices. However, as mentioned above, the experimental evidence is that the servo circuitry is tolerant of a considerable range of values. Nevertheless the advantage of the simultaneous algorithm is that all servo pulses go high at the same time, and therefore the pulse start times are separated by exactly 20 ms, regardless of the number of devices being controlled.

Another cause for concern is that the high frequency of interrupts may cause a time overhead in the running of the overall system, since the repeated raising of ISRs may adversely affect the ability to complete other processes in a realistic time frame. Experimental measurements with a system clock of nominally 16 MHz have revealed that the timer ISR executes in about 8 μs. This indicates that the interrupt latency is small compared with the pulse durations, and is considered to have negligible affect on system performance.

5 Experimental results

The above methods have been programmed into a C8051F206 microcontroller[6] to assess the effectiveness of the techniques. Outputs from the chip were connected to five independent servo motors of the S3003 type. Variable analog inputs were applied to the microcontroller to operate the individual servos. Since these tests formed part of the sailing boat control project, the five motors used were those already installed in the model yachts, as shown in Fig.7.

It was observed that each servo followed the corresponding input closely, and that no measurable interference between the motors was detected.

Fig.8 and Fig.9 show the waveforms for two of the servos, using the sequential method, illustrating the cases where the servos are at the extreme limits of their range of movement. These positions correspond to pulses of 400 μs

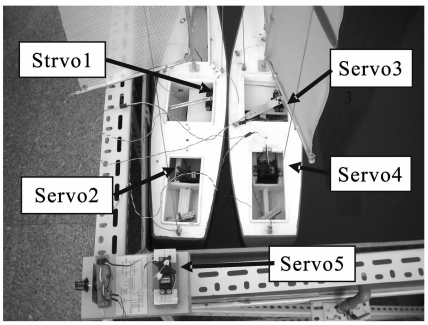

Fig. 7 Positions of the test servos in the model yachts

and 1900 μs.

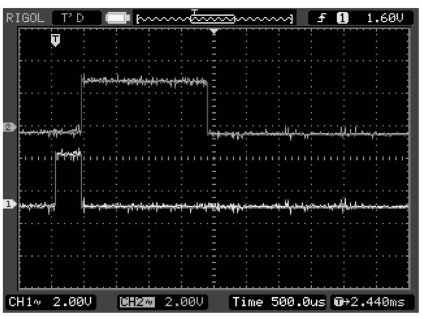

Fig. 8 Waveforms for two servos at extremities

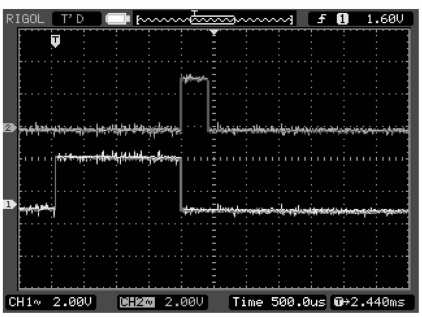

Fig. 9 Waveforms for the same servos at opposite extremes

When the simultaneous method was used, the waveform of Fig. 10 was produced.

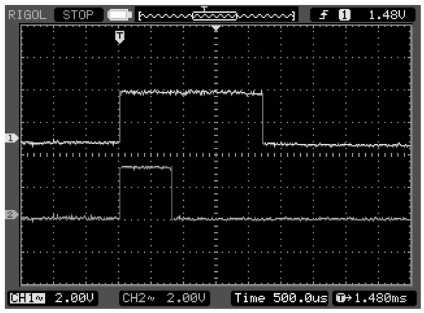

Fig. 10 Waveforms for the simultaneous method

6 Conclusion

The two algorithms given above have provided a method of control for several servo motors without the need to allocate one programmable timer to each device. The interrupt-driven nature of the programs means that they can be effectively run in the background in a multi-tasking environment. Empirical evidence suggests that the degradation of control accuracy is very slight for the increased number of servos, and that cross-talk between the motor controllers is minimal.

References

[1] Futaba Corporation. Futaba S3003 Servo Standard, http: // www. futaba-rc. com.

[2] Principia Labs, Arduina-Python 4-Axis Servo Control, http:// principialabs. com/arduino-python-4-axis-servo-control/

[3] Method for one digital control shaft controlling multiple servo shafts and shaft expansion control device, Ningjiang Machine Tool Group Co., Ltd., 2007.

[4] Pololu 16-servo controller kit-0, http://www. pololu. com/catalog/product/240

[5] Emerald Automation Controller-EMC-2100, http://www. iis-servo. com/IISAutomationSystems/EmeraldIntro/tabid/73x

[6] C8051F206 Datasheet, CYGNAL Integrated Products, 2002.