```
void timer_init (uint8_t high, uint8_t low)
{
	TH1 = high;
	TL1 = low;
	TMOD = B00010001;			// T1 MOD 16timer
	TR1 = 1;				// T1 start
	ET1 = 1;				// enable T1 interrupt
}

void timer_stop (void)
{
	TR1 = 0;
	ET1 = 0;
}
```

```
{
    for (; us != 0; us--);
}

void start_step (uint16_t cycles, uint16_t freq)
{
    uint8_t high, low;
    uint16_t high_speed, high_speed_num, i;

    // 计算最大频率在加速表中的位置
    high_speed = 0xffff - (uint16_t)(1000000L / (uint32_t)freq);
    for (i = 0; i < MAX_TABLE; i++) {
        if (high_speed < table[i])
            break;
    }
    if (i <= MIN_SPEED)
        high_speed_num = 1;
    else
        high_speed_num = i - MIN_SPEED;

    step_ground = GROUND;

    // 自动换算调速数据，这里我们用 1/3-1/3-1/3 方案
    roll_steps = cycles / GROUND;
    speed_up_steps = roll_steps / 3;
    if (speed_up_steps > high_speed_num)
        speed_up_steps = high_speed_num;
    speed_up_steps_B = speed_up_steps;          // 加减速运转步数
    roll_steps = cycles - speed_up_steps * 2;   // 匀速运转部数
    step_add = MIN_SPEED;                       // 启动速度

    high = *(p_table + step_add * 2);
    low = *(p_table + step_add * 2 + 1);

    timer_init (high, low);
    speed_up = 1;                       // 加速过程使能
    out_enable = 1;                         // 输出使能
    finished = 0;                       // 清结束标志
    while (!finished)
        WMCON |= WDTRST_;
    out_enable = 0;                         // 输出禁止
    timer_stop ();
}
```

```
{

    StepOut = 1;
//  TH1 = table[step_add * 2];
//  TL1 += table[step_add * 2 + 1];
    TH1 = *(p_table + step_add * 2);
    TL1 = *(p_table + step_add * 2 + 1);
    delay_us (10);
    StepOut = 0;

    if (!speed_up) {                        // 降速和匀速段
        if (speed_up_steps == 0) {          // 匀速段
            roll_steps--;
            if (roll_steps == 0) {          // 已减完，转入降速段
                speed_up_steps = speed_up_steps_B;
                step_ground = GROUND;
            }
        }
        else {                              // 降速段
            if (--step_ground == 0) {
                step_add--;
                if (--speed_up_steps == 0) {  // 整个输出结束
                    finished = 1;
                    out_enable = 0;
                }
                else
                    step_ground = GROUND;
            }
        }
    }
    else {
        if (--step_ground ==0) {            // 加速段
            step_add++;
            if (--speed_up_steps == 0) {    // 加速段运行完毕
                speed_up = 0;
                roll_steps--;
            }
            else
                step_ground = GROUND;
        }
    }
}

void delay_us (uint8_t us)
```

```
44
0xfe3b, 0xfe44, 0xfe4d, 0xfe55, 0xfe5d, 0xfe65, 0xfe6c, 0xfe73, 0xfe7a, 0xfe81, 0xfe88,        //
55
0xfe8e, 0xfe94, 0xfe9a, 0xfe9f, 0xfea5, 0xfeaa, 0xfeaf, 0xfeb4, 0xfeb9, 0xfebe, 0xfec3,    // 66
0xfec7, 0xfecb, 0xfed0, 0xfed4, 0xfed8, 0xfedc, 0xfedf, 0xfee3, 0xfee7, 0xfeea, 0xfeee,    // 77
0xfef1, 0xfef4, 0xfef7, 0xfefa, 0xfefd, 0xff00, 0xff03, 0xff06, 0xff09, 0xff0c, 0xff0e,       // 88
0xff11, 0xff13, 0xff16, 0xff18, 0xff1b, 0xff1d, 0xff1f, 0xff22, 0xff24, 0xff26, 0xff28,       // 99
0xff2a, 0xff2c, 0xff2e, 0xff30, 0xff32, 0xff34, 0xff36, 0xff37, 0xff39, 0xff3b, 0xff3d,       // 110
0xff3e, 0xff40, 0xff42, 0xff43, 0xff45, 0xff46, 0xff48, 0xff4a, 0xff4b, 0xff4c, 0xff4e,       // 121
0xff4f, 0xff51, 0xff52, 0xff53, 0xff55, 0xff56, 0xff57, 0xff59, 0xff5a, 0xff5b, 0xff5c,       // 132
0xff5d, 0xff5f, 0xff60, 0xff61, 0xff62, 0xff63, 0xff64, 0xff65, 0xff66, 0xff67, 0xff68,       // 143
0xff69, 0xff6a, 0xff6b, 0xff6c, 0xff6d, 0xff6e, 0xff6f, 0xff70, 0xff71, 0xff72, 0xff73,       // 154
0xff74, 0xff74, 0xff75, 0xff76, 0xff77, 0xff78, 0xff79, 0xff79, 0xff7a, 0xff7b, 0xff7c,       // 165
0xff7c, 0xff7d, 0xff7e, 0xff7f, 0xff7f, 0xff80, 0xff81, 0xff82, 0xff82, 0xff83, 0xff84,       // 176
0xff84, 0xff85, 0xff86, 0xff86, 0xff87, 0xff88, 0xff88, 0xff89, 0xff89, 0xff8a, 0xff8b,       // 187
0xff8b, 0xff8c, 0xff8c, 0xff8d, 0xff8e, 0xff8e, 0xff8f, 0xff8f, 0xff90, 0xff90, 0xff91,       // 198
0xff91, 0xff92, 0xff92, 0xff93, 0xff94, 0xff94, 0xff95, 0xff95, 0xff96, 0xff96, 0xff97,       // 209
0xff97, 0xff97, 0xff98, 0xff98, 0xff99, 0xff99, 0xff9a, 0xff9a, 0xff9b, 0xff9b, 0xff9c,       // 220
0xff9c, 0xff9c, 0xff9d, 0xff9d, 0xff9e, 0xff9e, 0xff9f, 0xff9f, 0xff9f, 0xffa0, 0xffa0,       // 231
0xffa1, 0xffa1, 0xffa1, 0xffa2, 0xffa2, 0xffa3, 0xffa3, 0xffa3, 0xffa4, 0xffa4, 0xffa4,       // 242
0xffa5, 0xffa5, 0xffa5, 0xffa6, 0xffa6, 0xffa7, 0xffa7, 0xffa7, 0xffa8, 0xffa8, 0xffa8,       // 253
0xffa9, 0xffa9, 0xffa9, 0xffaa, 0xffaa, 0xffaa, 0xffab, 0xffab, 0xffab, 0xffab, 0xffac,       // 264
0xffac, 0xffac, 0xffad, 0xffad, 0xffad, 0xffae, 0xffae, 0xffae, 0xffae, 0xffaf, 0xffaf,    // 275
0xffaf, 0xffb0, 0xffb0, 0xffb0, 0xffb0, 0xffb1, 0xffb1, 0xffb1, 0xffb2, 0xffb2, 0xffb2,       // 286
0xffb2, 0xffb3, 0xffb3, 0xffb3, 0xffb3, 0xffb4, 0xffb4, 0xffb4, 0xffb4, 0xffb5, 0xffb5,       // 297
0xffb5, 0xffb5, 0xffb6, 0xffb6, 0xffb6, 0xffb6, 0xffb7, 0xffb7, 0xffb7, 0xffb7, 0xffb7,       // 308
0xffb8, 0xffb8, 0xffb8, 0xffb8, 0xffb9, 0xffb9, 0xffb9, 0xffb9, 0xffb9, 0xffba, 0xffba,       // 319
0xffba, 0xffba, 0xffbb, 0xffbb, 0xffbb, 0xffbb, 0xffbb, 0xffbc, 0xffbc, 0xffbc, 0xffbc,       // 330
0xffbc, 0xffbd, 0xffbd, 0xffbd, 0xffbd, 0xffbd, 0xffbe, 0xffbe, 0xffbe, 0xffbe, 0xffbe,       // 341
0xffbe, 0xffbf, 0xffbf, 0xffbf, 0xffbf, 0xffbf, 0xffc0, 0xffc0, 0xffc0, 0xffc0, 0xffc0,    // 352
0xffc0, 0xffc1, 0xffc1, 0xffc1, 0xffc1, 0xffc1, 0xffc1, 0xffc2, 0xffc2, 0xffc2, 0xffc2,       // 363
0xffc2, 0xffc2, 0xffc3, 0xffc3, 0xffc3, 0xffc3, 0xffc3, 0xffc3, 0xffc4, 0xffc4, 0xffc4,       // 374
0xffc4, 0xffc4, 0xffc4, 0xffc5, 0xffc5, 0xffc5, 0xffc5, 0xffc5, 0xffc5, 0xffc5, 0xffc6,       // 385
0xffc6, 0xffc6, 0xffc6, 0xffc6, 0xffc6, 0xffc6, 0xffc7, 0xffc7, 0xffc7, 0xffc7, 0xffc7,       // 396
0xffc7, 0xffc7, 0xffc8, 0xffc8, 0xffc8, 0xffc8, 0xffc8, 0xffc8, 0xffc8, 0xffc9, 0xffc9,       // 407
0xffc9, 0xffc9, 0xffc9, 0xffc9, 0xffc9, 0xffc9, 0xffca, 0xffca, 0xffca, 0xffca, 0xffca,   // 418
0xffca, 0xffca, 0xffca, 0xffcb, 0xffcb, 0xffcb, 0xffcb, 0xffcb, 0xffcb, 0xffcb, 0xffcb,       // 429
0xffcc, 0xffcc, 0xffcc, 0xffcc, 0xffcc, 0xffcc, 0xffcc, 0xffcc, 0xffcc, 0xffcd, 0xffcd,    // 440
0xffcd, 0xffcd, 0xffcd, 0xffcd, 0xffcd, 0xffcd, 0xffcd
};

uint8_t code *p_table = (uint8_t code *)table;

void timer_int (void) interrupt 3 using 1
```

```
#include <atmel/at898252.h>
#include <intrins.h>
#include "config.h"
#include "hardware.h"

#define MAX_TABLE   446
#define GROUND      8
#define MIN_SPEED   4

void delay_us (uint8_t us);
void start_step (uint16_t cycles, uint16_t freq);
void timer_init (uint8_t high, uint8_t low);
void timer_stop (void);

#ifndef STEPADDR
#define STEPADDR    P1^0
#endif
sbit StepOut    = STEPADDR;

uint8_t bdata   status;                 // 综合位变量寄存字节
sbit speed_up  = status^0;
sbit out_enable     = status^1;
sbit finished   = status^2;

uint8_t         step_ground;        // 台阶内部数记录变量
uint16_t  step_add;       // 当前速度记录变量
uint16_t  speed_up_steps,           // 当前速度计算所得变量
          speed_up_steps_B, // 当前速度计算所得变量 b 备份
          roll_steps;           // 填充脉冲数变量

//////////////////////////////////////////////////////////////////////////
// 晶振频率=12MHz; 预分频率=12
// 参数设定值: V0 = 20; 折点 1= 100; 折点 2 = 2200 极限转速 = 3000 rpm
//              Point0= 0;第一折点= 12;第二折点= 100; 终点步数= 160
// 电机转动一周对应脉冲数=200; 编程响应(CTn rewrite)补偿步数 = 8
//////////////////////////////////////////////////////////////////////////
uint16_t code table[] = {
0xec77, 0xf05f, 0xf2ef, 0xf4be, 0xf61c, 0xf729, 0xf803, 0xf8b6, 0xf949, 0xf9c7, 0xfa34,      //
11
0xfa93, 0xfae6, 0xfb30, 0xfb72, 0xfbad, 0xfbe3, 0xfc12, 0xfc3e, 0xfc66, 0xfc8b, 0xfcac,      //
22
0xfccc, 0xfce9, 0xfd04, 0xfd1d, 0xfd35, 0xfd4b, 0xfd60, 0xfd74, 0xfd87, 0xfd98, 0xfda9,      //
33
0xfdb8, 0xfdc7, 0xfdd5, 0xfde3, 0xfdf0, 0xfdfc, 0xfe08, 0xfe13, 0xfe1e, 0xfe28, 0xfe32,      //
```