AN10496

Vacuum cleaner with Philips P89LPC901

Rev. 01 — 10 August 2006 **Application note**

Document information

Info	Content
Keywords	P89LPC901, Vacuum Cleaner, Soft start, Harmonic suppression, Low cost
Abstract	A low cost P89LPC901 based vacuum cleaner system is introduced in this application note. Design hardware and software are fully discussed. This system can also guide the design of other universal motor driving systems that needs robust controlling and harmonic suppression.

PHILIPS

Revision history

Rev	Date	Description
01	20060810	Initial version

Contact information

For additional information, please visit: http://www.semiconductors.philips.com

For sales office addresses, please send an email to: sales.addresses@www.semiconductors.philips.com

1. Introduction

Universal motor control using microcontrollers is widely used in industrial applications and domestic appliances. Domestic appliance examples include vacuum cleaners. Industrial applications include power tools. Here we focus on a vacuum cleaner with the information being equally relevant to all the applications.

Today, vacuum cleaners may be found in nearly every household. They are designed to make life and work easier. The speed of the universal motor is controlled through a TRIAC. With a small current on the gate terminal, the TRIAC conducts the current that passes through the motor. This way the area of the current determines the motor's power and controls the motor's speed.

In low-end vacuum cleaners, the control circuit is very simple. This kind of simple circuit may introduce several problems including:

1. The startup current might be too high.
2. As the power of the motor increases, normally more than 1500 W, the none-full current waveform can produce high harmonics.

The above two faults may cause the device fail to meet the IEC61000-3-2 standard.

3. The non-linear inductive load may require continuous long lasting TRIAC fire pulses that will consume additional power.

In this application note, we will introduce a vacuum cleaner application controlled by the Philips P89LPC901 microcontroller driving an AC 1800 W universal motor through TRIAC.

The following applications will be provided in this demo:

1. A soft start algorithm to minimize the surge current at start up.
2. Soft switching when increasing or decreasing the motor's speed.
3. The TRIAC fire pulse is modified to suppress the harmonics brought by the not full sinusoidal current waveform. The measurement of harmonic components and motor power is done with an oscilloscope (TDS5054B with TCPA300/TCP305 together with the software -- power measurement) and a digital power meter (WT210). The results show much better performance than normal control methods.
4. Speed control and robust control, which will be described in detail below.

2. Design hardware

A vacuum cleaner reference design is shown in Fig 1, and a brief description of the circuit operation follows. For more detail see the schematics in appendix A.

The three I/O ports of the P89LPC901 are used to generate the TRIAC drive waveform and control the speed of the motor. The gate negative trigger current of TRIAC BT139-800 is 35 mA. Three port pins can provide sufficient trigger current to drive the TRIAC directly with each I/O port putting out 20 mA current.

Two keys are used to get the speed for the motor. The MCU reads the keys' status using two I/O pins and then adjusts the motor speed. A single port pin is used with a Key Pad Interrupt (KBI) function to synchronize to the AC line. This input port current that injects into the MCU is limited using a large value resistor.

The MCU power supply current is taken directly from the mains supply. A capacitor, plus a resistor dropper circuit, is used for voltage and current dropping. The current of the MCU power supply is limited by the size of the AC line dropper capacitor. A high-voltage capacitor and a high-speed switching diode 1N4148 are needed to filter out the AC current and supply a DC current for the MCU. Between the V_{DD} and the 1N4148, a 3.9 V Zener diode is used for the MCU voltage regulation. Testing shows that such a low cost MCU power supply circuit can provide enough stability. In most applications a quartz crystal or ceramic resonator supplies the MCU clock. In this application, for cost reasons, the P89LPC901 on-chip oscillator generates the system clock. The ± 1 % on-chip oscillator can provide sufficient precision for this application.

Note: EXTREME CAUTION should be taken because there is NO isolation circuit on the board. The whole board is directly connected to the mains supply, which can be at a high voltage. When testing the hardware, an isolating transformer should be introduced to the power supply of the board for safety.

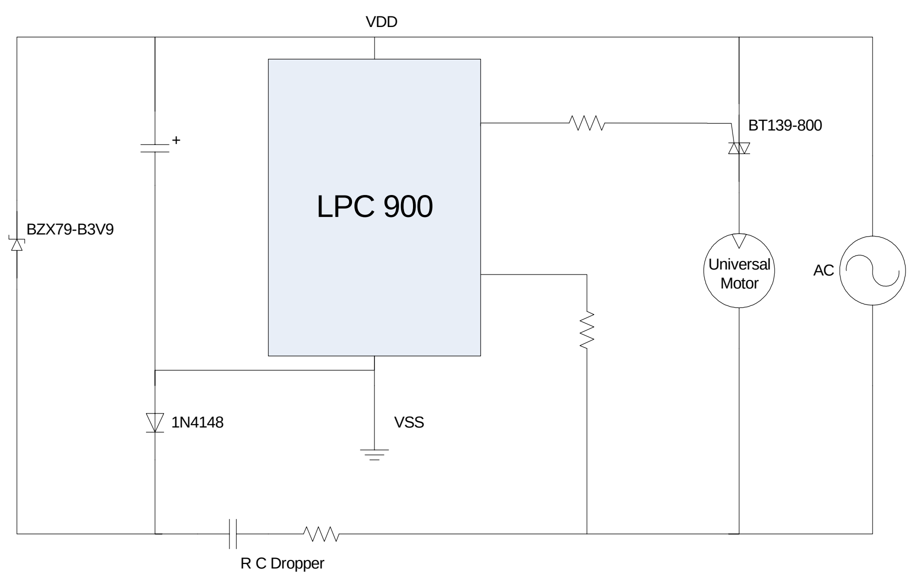

Fig 1. Vacuum cleaner reference design board

3. System Design

This section describes the design features of the universal motor control system. It is intended to help you to understand the design basics and to use those features as a basis for developing your own motor drive and to adapt it to your own requirements.

The section is organized as follows: Speed control, TRIAC drive control, soft start, and harmonic suppression.

3.1 Speed control

Universal motor speed control is based on phase angle control. When the current passes zero crossing, the TRIAC will not conduct until sufficient current triggers the gate terminal. The TRIAC will then continue conduction until next current zero crossing. The average power of the motor is now proportional to the area of the current waveform. By controlling the firing angle of the TRIAC, we can determine the average power of the load, including the universal motor or a lamp.

3.2 TRIAC drive control

According to the data sheet of the BT139-800, the gate terminal turn on time is about 2 μs. For robust controlling, we set the TRIAC firing pulse to be 200 μs. Once conducted, the TRIAC will stay on until the next zero crossing. So the trigger current at gate terminal can be withdrawn. As we know, most loads are not pure impedance loads, e.g., a universal motor. A universal motor is an inductive load. That is, the current of the load will lag the voltage. When the voltage reaches zero crossing, the current may continue to go for some degrees until cross its zero. If we fire the TRIAC near the zero voltage crossing point with a pulse as we used at other phase, the TRIAC may not be conducted as desired. Some method needs to be implemented to trigger the pulse of the TRIAC at those phases.

In this application, we apply a long fire pulse at the phase close to the ZVC. For long fire pulse, the trigger pulse is set to be 400 μs, twice the fire pulse at other angle. 400 μs are suitable for current lagging not exceeding 7 degrees.

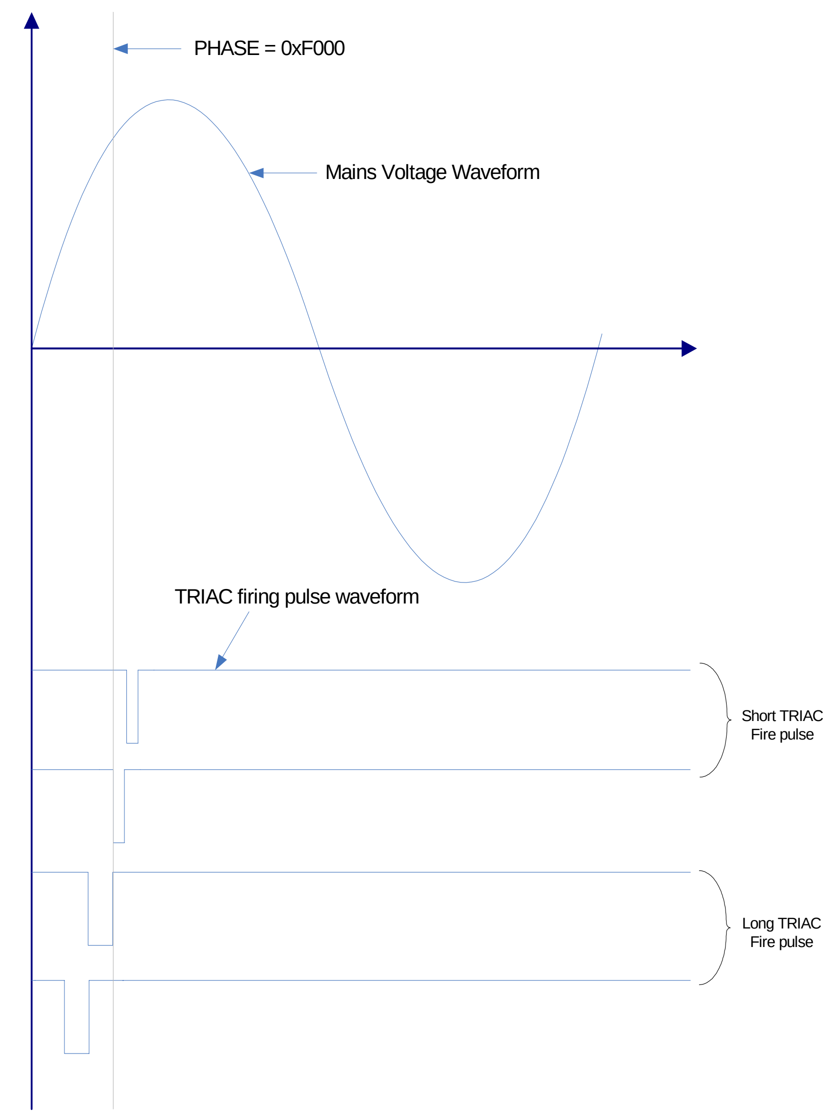

Fig 2. Long fire waveform

3.3 Start up delay

The start up delay feature can reduce the startup surge current of the universal motor.

At start up, when charged with mains supply, there will be very high amplitude current among the motor that may not comply with the limitation of IEC61000-3-2 standard.

The startup delay stays at a speed point until it is stable and then shifts into the next level. Finally, the motor will reach the lowest power level of the vacuum cleaner.

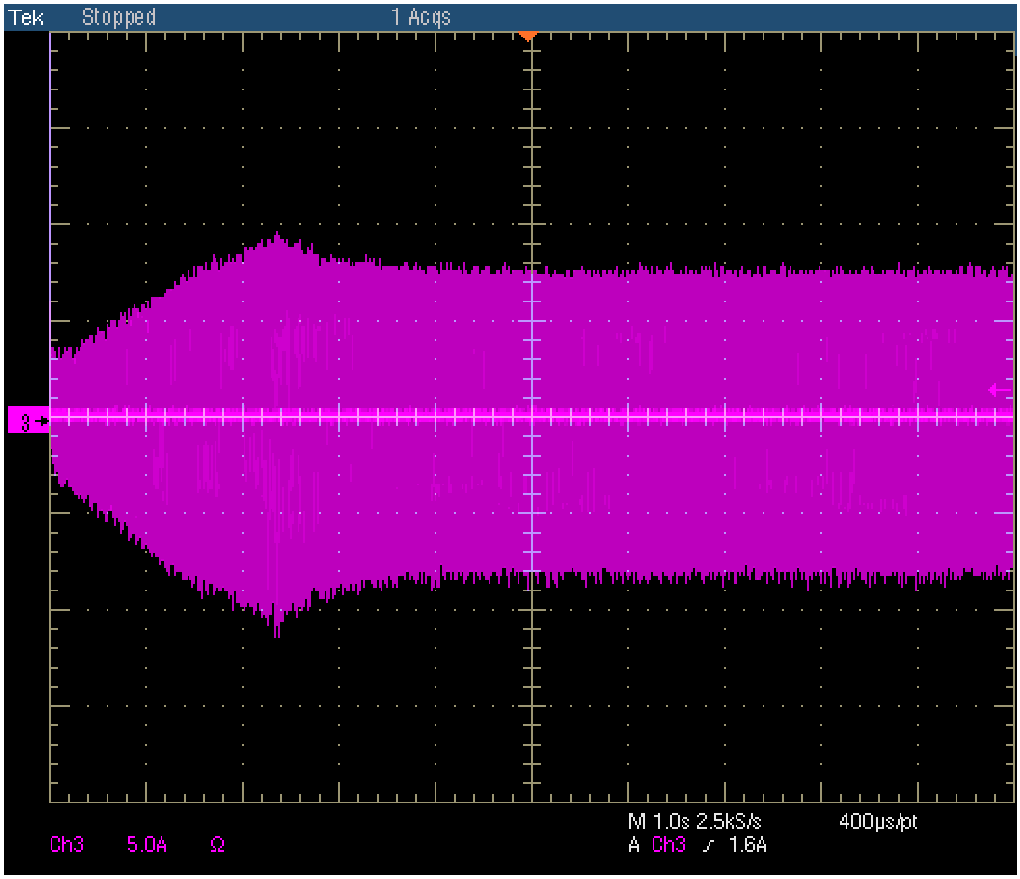

Fig 3. Start-up delay demo

3.4 Soft switch

The soft switch algorithm allows controlling the speed smoothly when changing speeds. Appendix F shows the flow diagram of the soft start subroutine.

By switching the speed, the soft switch scheme will prevent the current from changing dramatically. If the desired speed is faster or slower than current speed for more than one-step span, the software will get to the desired speed step by step and manage to smoothen the speed switching. Each step will hold on for an “update rate” period to stabilize the current and then move to next speed level. An experiment has shown that 35 steps from minimum to maximum speed are enough for this application. Such an algorithm provides robust control of the motor and prolongs the life of the motor.

The software is compact, efficient, and suitable for any P89LPC900 series microcontroller.

3.5 Harmonic suppression

Harmonic suppression is one of the most important features of the design. In this application, we apply the KURZ phase control method. This method modulates the universal motor current with one long phase trigger full wave and one short phase trigger full wave.

The performance of the method is shown in Fig 4. The universal motor is V1J-PH29 1800 W/230 V from Suzhou CINDERSON. Channel 1 is the AC mains voltage waveform; channel 2 is the motor current waveform.

This method has already been patented by KURZ. The patent number is DE 19705907C1 (German Patent) and EP 0859452B1 (European Patent).

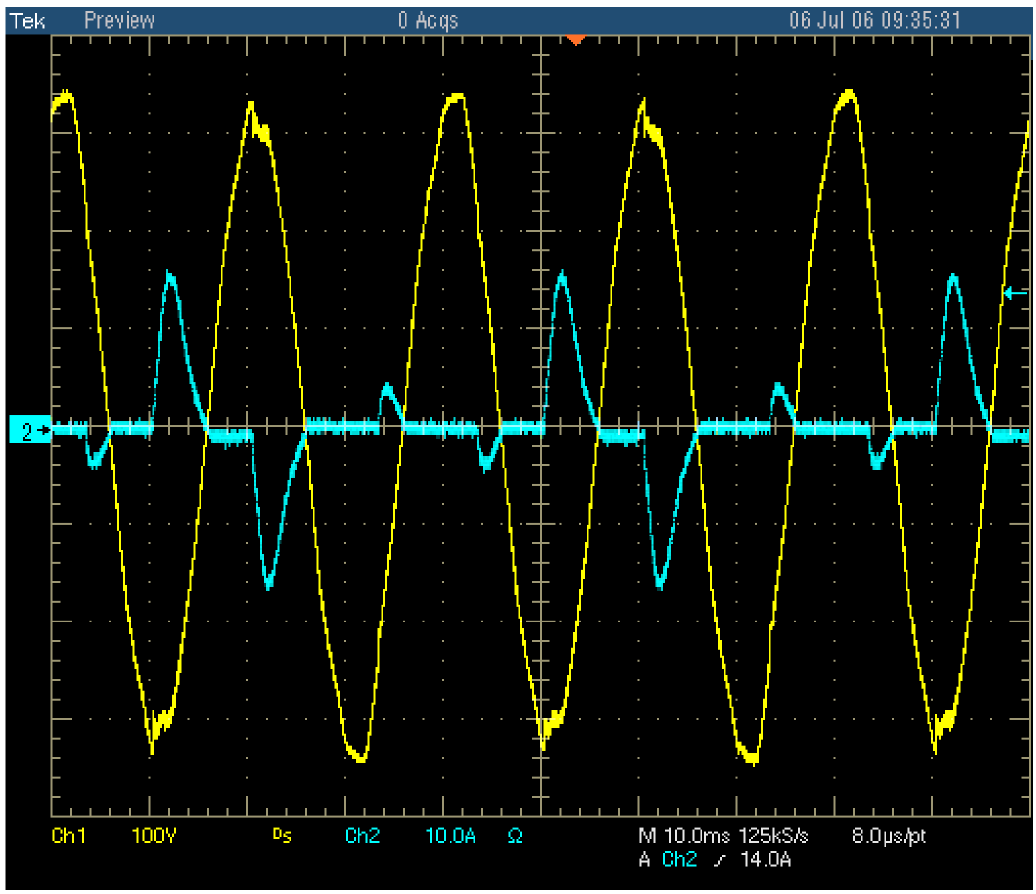

Fig 4. Harmonic reducing demo

Table 1. Testing results for the V1J-PH29 1800 W/230 V universal motor from Suzhou CINDERSON with the KURZ method

POWER	Harmonic order and corresponding current (A)			
(W)	3	5	7	9
700	1.475	0.372	0.307	0.250
780	1.475	0.455	0.452	0.351
820	1.869	0.561	0.516	0.322
860	1.814	0.537	0.496	0.337
900	1.800	0.497	0.517	0.359
940	1.773	0.498	0.564	0.352
970	1.713	0.543	0.586	0.338
1060	1.714	0.552	0.643	0.349
1160	1.768	0.531	0.689	0.323
1270	1.863	0.486	0.566	0.181
1340	1.963	0.491	0.478	0.058
1450	1.943	0.603	0.360	0.012
1560	1.904	0.359	0.092	0.169
1680	1.804	0.286	0.256	0.152
1700	1.605	0.264	0.175	0.197
700	1.949	0.642	0.534	0.255

4. Vacuum cleaner software

In this section, we will discuss the whole structure of the vacuum cleaner software. This software is developed for the P89LPC901, and it will run on any Philips P89LPC900 MCU with simple modifications. This MCU has Key Pad Interrupt functions that enable the mains zero voltage crossing detection. The two timers provide all the necessary timing control for the software. Timer 0 is used for TRIAC pulse generator. Timer 1 is configured as keys status sampler.

The P89LPC901 also features an internal oscillator and a small 8-pin package.

First, the MCU processes the initialization. A start up delay is added to ensure configuration operation and waits for the start up current to stabilize. The main function is ended with an endless while(1) loop.

The non-time critical events are harmonic waveform generation, soft switch, and timer value conversion which all can be performed in the while(1) loop. Meanwhile, the zero voltage crossing detection, TRIAC pulse generation, and key status sampling, which require in time operation events, can be handled by the interrupt.

4.1 Main loop

The main loop contains no time critical functions.

When entering the main routine, init() function is processed to initialize global variables and I/O ports. Other hardware initialization of the MCU, such as KBI, timer, interrupt, and on-chip RC Oscillator settings, are also implemented in this function.

After configuration, the main routine comes to the while(1) loop. Subroutine get_speed() processes the control of updating the global variable PHASE. PHASE in this software is used for Timer0 TRIAC fire time transferring. The get_speed() function is the combination of four subroutines: get_ADC(), softswitch(), harm_reduce () and phase2timer(). Each subroutine performs a basic service as shown in the flow diagram in Appendix D.

4.2 KBI routine

This application note details the KBI interrupt subroutine because of its complexity and importance to the whole software. Other subroutines can be easily understood from the flow diagrams in Appendix F, Appendix G and Appendix H.

Pin 6 of the P89LPC901 is configured as the KBI interrupt input pin. This pin is used as the zero voltage crossing detection.

The main features of the KBI routine include: AC line synchronization, Timer 0 TRIAC fire angle loading, harmonic suppressing waveform controlling, and soft switch update rate controlling.

As shown in Fig 6, the KBI subroutine is invoked when a falling or rising edge event occurs on Pin 6. When entered, the first thing is to disable the global interrupt and not allowing other interrupts to take place while the KBI routine is running. In order to reenter KBI on the next zero voltage crossing point, inversing the P89LPC901 KBI interrupt pattern is needed. That is, if current invoke event is falling edge (1 to 0), the KBI interrupt pattern should be set as 1 so that next rising edge (0 to 1) will invoke the KBI interrupt. For more detail please refer to the P89LPC901 user manual.

Thanks to the flexible configuration of P89LPC900 microcontroller, the software can be simple and robust. This saves time for the CPU to perform other functions and makes the whole software more synchronized to the AC mains.

5. Conclusion

In this application, we introduce a cost saving P89LPC901 microcontroller based vacuum cleaner system that can be a guide for other controlling designs like universal motor control design or lamp or power tools design. The hardware implementation is simple and cost effective. The five most important system design points are discussed. They include: speed control, TRIAC drive control, start up delay, soft switch, and Harmonic suppression. The software has been introduced with main loops and KBI interrupt routine.

Results have shown good performance of the systems. The 1800 W vacuum cleaner demo system controlled by P89LPC901FN can pass the IEC61000-3-2 standard at startup and each speed checkpoint.

6. Appendix A

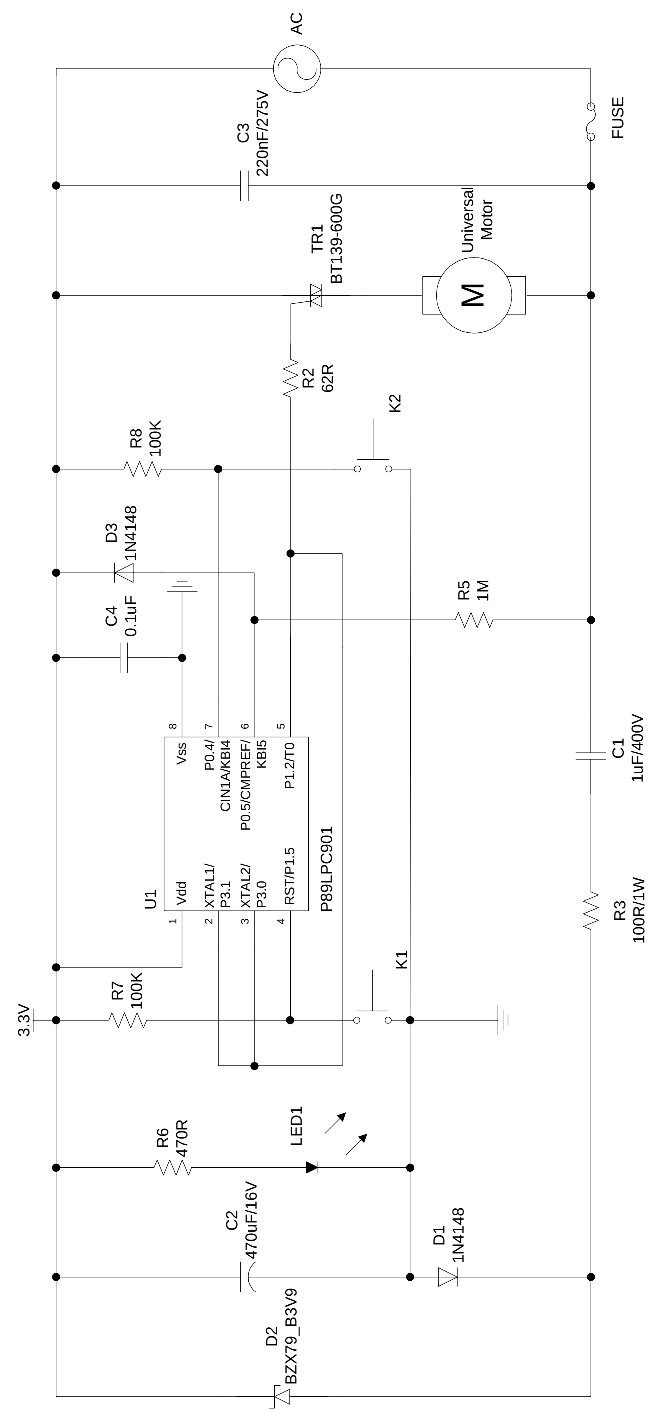

Fig 5. Detailed schematics of the vacuum cleaner circuit using the P89LPC901 microcontroller

Table 2. LPC901 Vacuum cleaner demo board Bill of Materials (BOM)

Item	Quantity	Reference	Part	Manufacturer
1	1	U1	P89LPC901FN	Philips Semiconductor
2	1	TR1	BT139-800	Philips Semiconductor
3	1	D2	1	Philips Semiconductor
4	2	D1, D3	1N4148	Philips Semiconductor
5	1	R5	1M Ω	
6	1	R2	62 Ω	
7	2	R7, R8	100KΩ	
8	1	R3	100 Ω/1W	
9	1	R6	470 Ω	
10	1	C1	1uF/400V	
11	1	C2	470uF/16V	
12	1	C3	220nF/275V	
13	1	C4	0.1uF	
14	1	CON_1	V1J-PH29	CINDERSON
15	1	F1	10A FUSE	

7. Appendix B

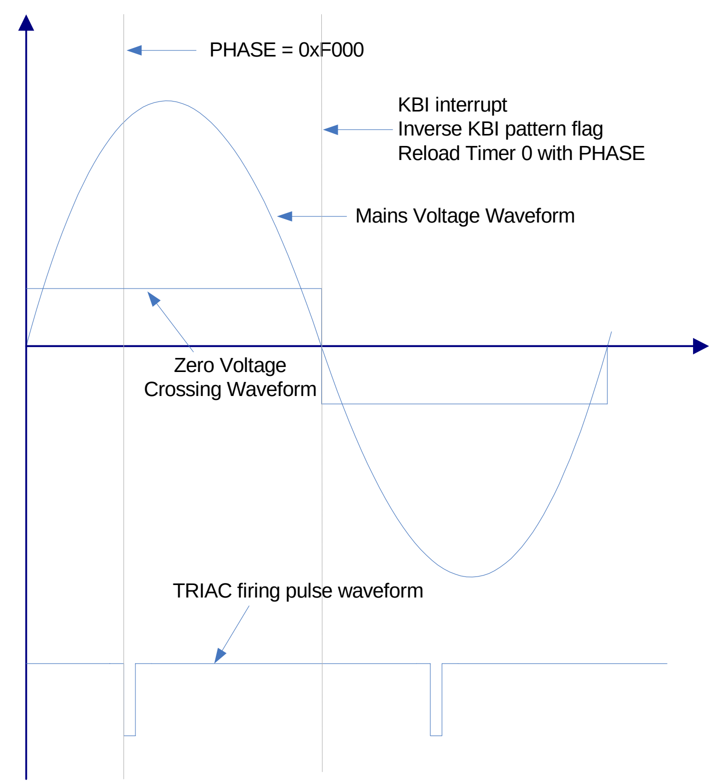

Fig 6. Detailed KBI controlling waveform

8. Appendix C

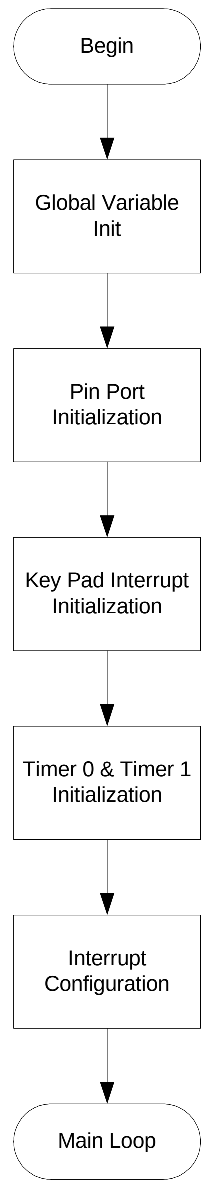

Fig 7. Init subroutine flowchart

9. Appendix D

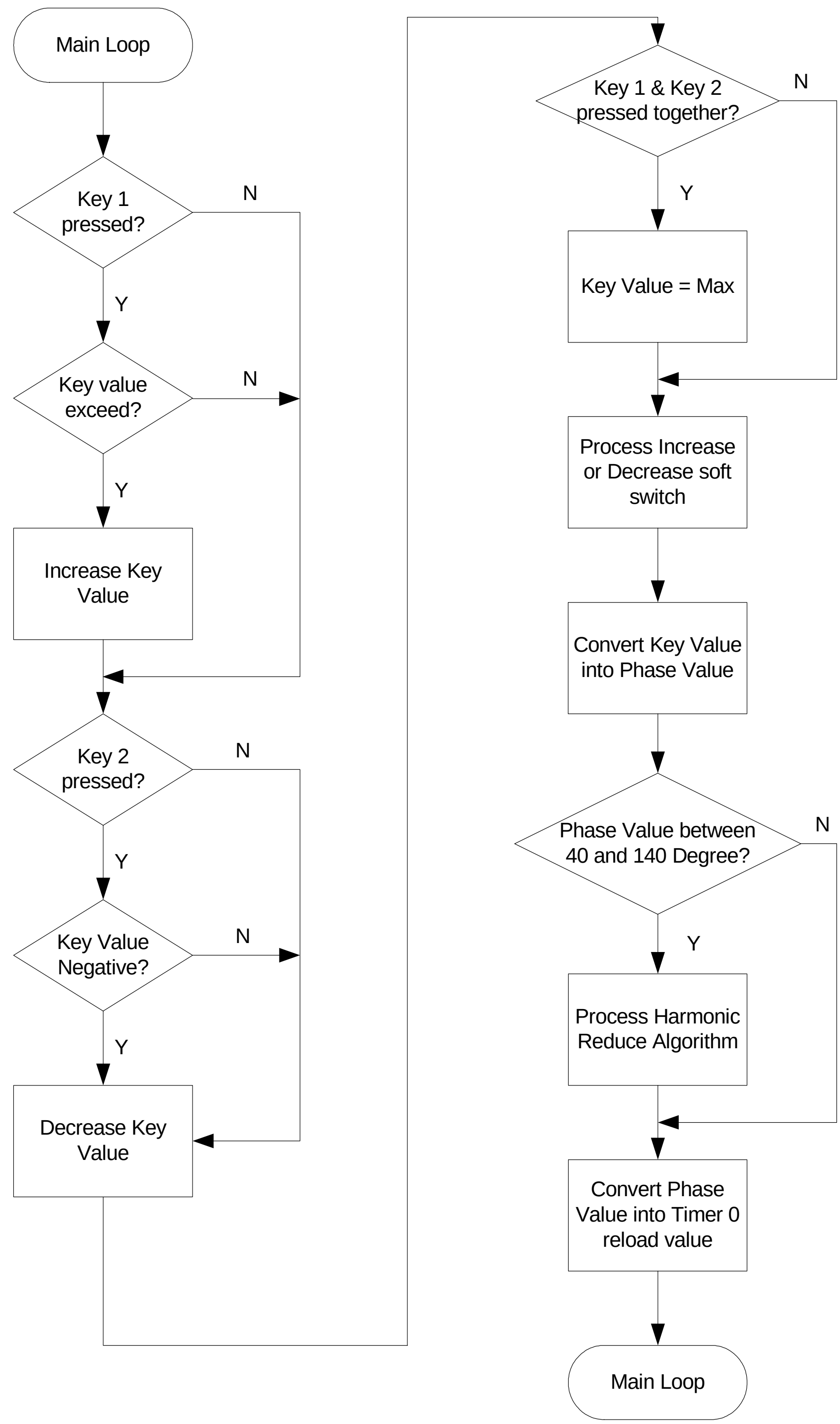

Fig 8. Main loop flowchart

10. Appendix E

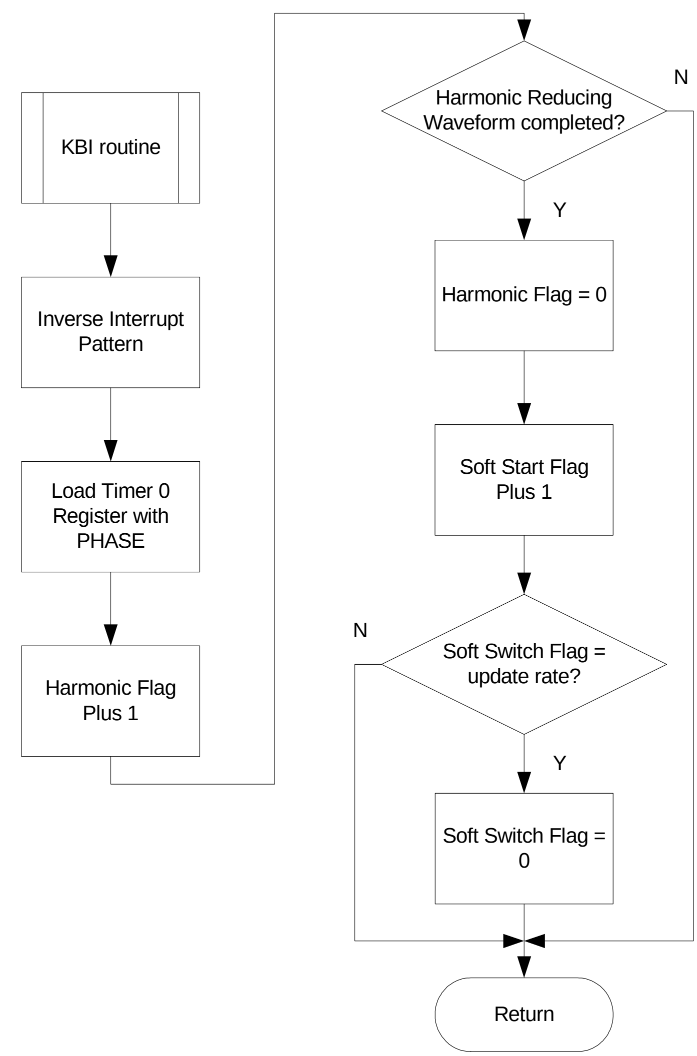

Fig 9. KBI subroutine flowchart

11. Appendix F

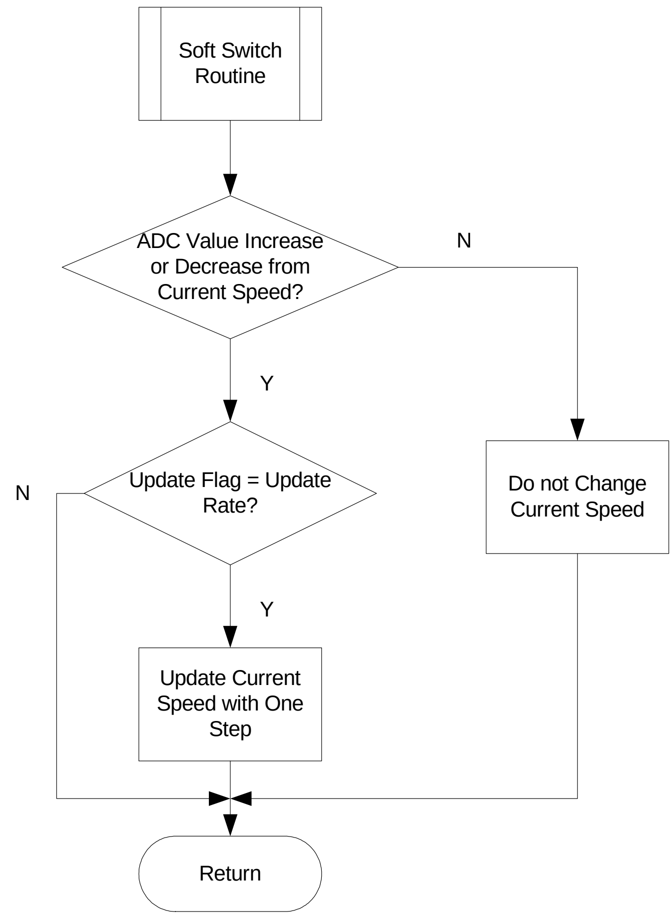

Fig 10. Softswitch subroutine flowchart

12. Appendix G

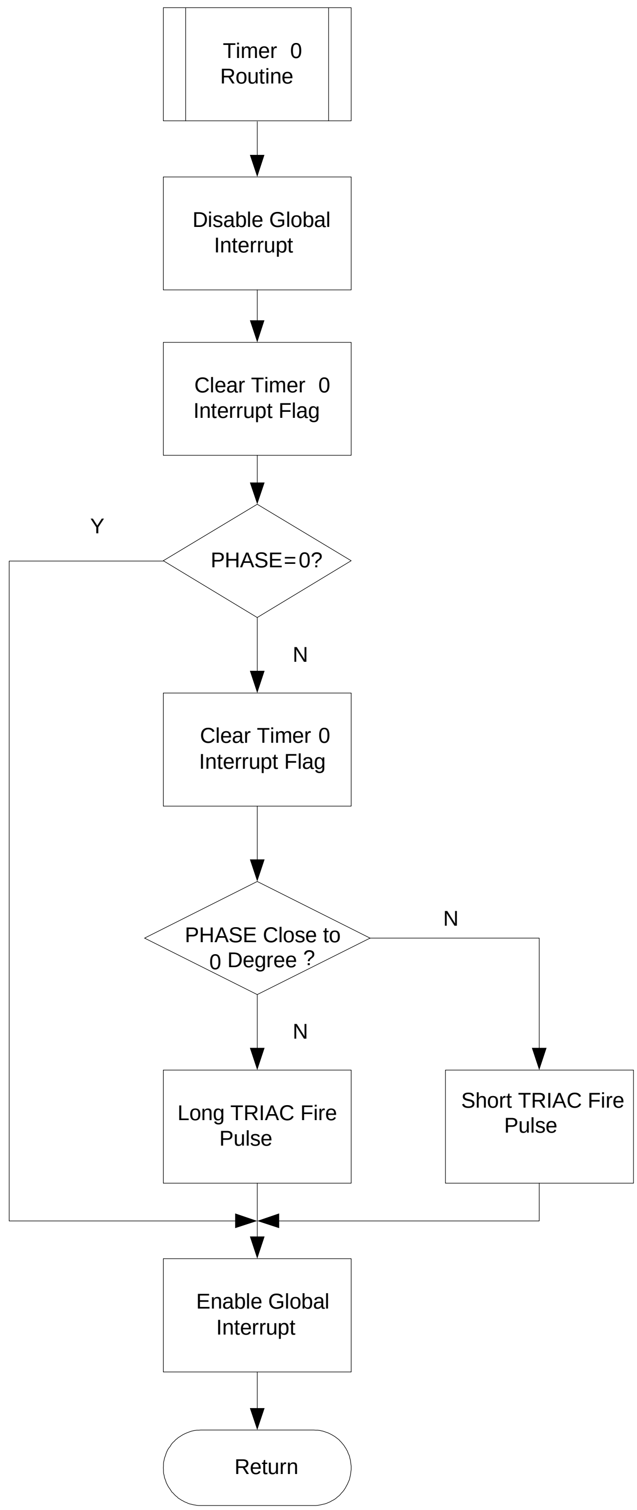

Fig 11. Timer0 subroutine flowchart

13. Appendix H

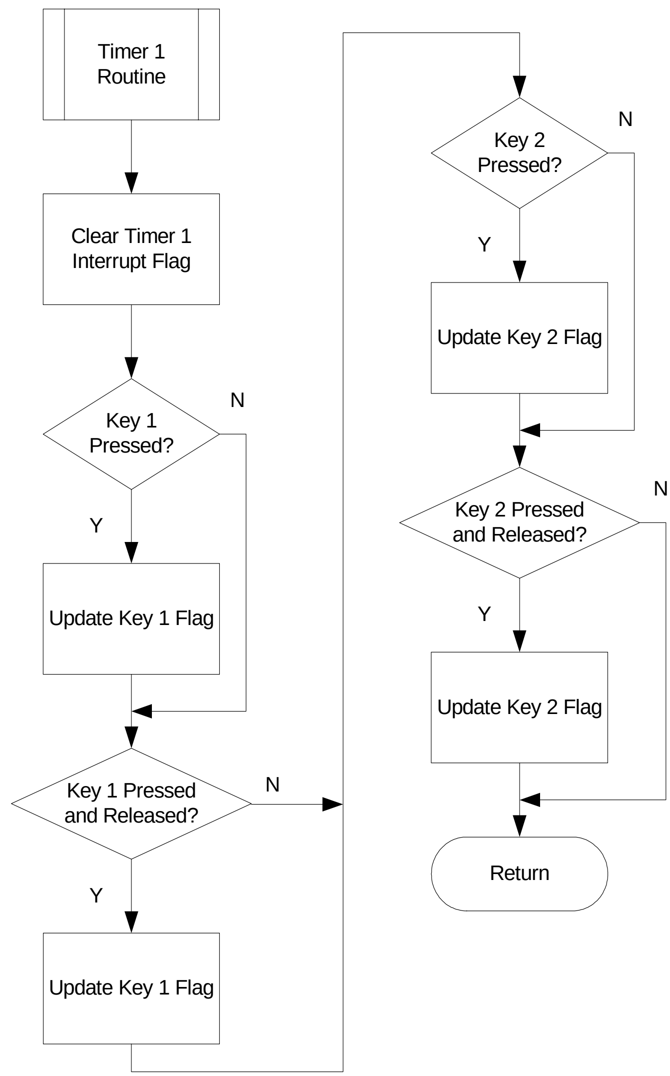

Fig 12. Timer1 subroutine flowchart

14. Appendix I

```
//****************************************************************************
//*
//* vacuum.c
//* Date : July 2005
//* Description : Vacuum Cleaner demo program with Philips P89LPC901
//*
//*******************Revise History*****************************************
//* Date : July 2005
//* Description : Creat
//****************************************************************************
#include <REG901.H>             // register definition
//-----------------------------------------------
//Port Pin Definitions
//-----------------------------------------------
sbit key1 = P0^4;
sbit key2 = P1^5;
sbit Port1 = P1^2;
sbit Port2 = P3^0;
sbit Port3 = P3^1;
//-----------------------------------------------
//Global Variable Definitions
//-----------------------------------------------
volatile unsigned char curr_flag;
volatile unsigned char syn;
volatile unsigned int PHASE;
volatile unsigned int adc_old;
volatile unsigned char one_step;
volatile unsigned char harm_flag;
volatile unsigned char update;
volatile unsigned char key1_flag;
volatile unsigned char key2_flag;
volatile unsigned char key_value;
volatile unsigned char speed_max;
//-----------------------------------------------
//Constant Definitions
//-----------------------------------------------
#define update_rate  2

//****************************************************************************
//* Functions
//****************************************************************************
void init(void);
void startupdelay (unsigned int degree, unsigned int cnt);
unsigned int phase2timer(unsigned int phase_value);
void KBI_ISR(void);
void T0_ISR(void);
void T1_ISR(void);
void fire_triac(void);
unsigned int get_speed(void);
unsigned int harm_reduce(unsigned int phase_value);
```

```
unsigned int get_ADC(void);
unsigned int softswitch(unsigned int adc_value);
void delay (unsigned int cnt);
//*****************************************************************************
//* Name: main()
//* Input(s) : none.
//* Returns : none.
//* Description : main loop
//*****************************************************************************
void main()
{
      init();
      startupdelay(140,8);                   //wait for startup stable
      while(1)
      {
            PHASE = get_speed();
      }
}
//*****************************************************************************
//* Name: init()
//* Input(s) : none.
//* Returns : none.
//* Description : initialization of P89LPC901
//*****************************************************************************
void init(void)
{
curr_flag = 0;
syn = 0;
PHASE = 0;
adc_old = 0x0;
one_step = 1;
harm_flag = 0;
update = 0;
key1_flag = 0;
key2_flag = 0;
key_value = 0x0;
speed_max = 46;  //from 0~140 degree, set as 46 levels, so 3 degree a level
/* Pin configuration */
P0M1 = 0x20;
P0M2 = 0x0;     //pin 7 (P0.4) as Quasi-bidir. & pin 6 (P0.5) as input only
P1M1 = 0x0;
P1M2 = 0x04;               //pin 5 (P1.2) as Push-Pull
P3M1 = 0x0;
P3M2 = 0x03;               //pin 2, 3 (P3.1, P3.0) as Push-Pull
/* KBI configuration */
KBMASK = 0x20;       //P0.5 as keypad interrupt
KBPATN = 0x20;       //pattern is high-level
KBCON = 0x0; //when signal in P0.5 is not equal to high-level, generate interrupt
/* Timer configuration */
      TMOD = 0x11;         //Timer0 as Mode 1 and Timer1 as Mode 1
      TAMOD = 0x0;         //16 bit mode
```

```
     TH0 = 0x0;           //init Timer0 value to be maximum(about 18ms)
     TL0 = 0x0;           //note: The timer counts up
     TR0 = 0x01;          //run Timer0
     TH1 = 0x0;           //init Timer1 value to be maximum(about 18ms)
     TL1 = 0x0;
     TR1 = 0x01;          //run Timer1
     /* Interrupt configuration */
     EA = 0x01;           //enable global interrupt
     ET0 = 0x1;           //enable Timer0 interrupt
     EKBI = 0x1;          //enable KBI interrupt
     ET1 = 0x01;          //enable Timer1 interrupt as key input
     IP1H = 0x02;         //set KBI interrupt priority as level 3 (highest)
     IP1 = 0x02;
     IP0H = 0x02;         //set Timer0 interrupt priority as level 3 (highest)
     IP0 = 0x02;          //and Timer1 interrupt priority as level 0 (lowest)
     /* close all other interrupt */
     EBO = 0x0;
     EWDRT = 0x0;
     EC = 0x0;
     /* RC Oscillator */
     DIVM = 0x00;              //Fcpu = Fosc / (2 * DIVM)
}
//****************************************************************************
//* Name: startupdelay()
//* Input(s) : unsigned int degree, unsigned int cnt.
//* Returns : none.
//* Description : provide softstart function
//****************************************************************************
void startupdelay (unsigned int degree, unsigned int cnt)
{
     unsigned int i,j;
     for(i=170;i > degree;i--)
     {
          for(j=0;j<cnt;j++)
          {
               while(!syn);
               syn = 0;
          }
          PHASE = phase2timer(i);
     }
}
//****************************************************************************
//* Name: T0_ISR()
//* Input(s) : none.
//* Returns : none.
//* Description : Interrupt from Timer 0
//****************************************************************************
void T0_ISR(void) interrupt 1    //T0 interrupt vector address is 000Bh
{
     EA = 0x0;                  //disable global interrupt
     while(TF0 != 0x0)
```

```
    {
        TF0 = 0x0;              //clear TF0 bit
    }
    //fire TRIAC
    if(PHASE != 0x0000)
    {
        fire_triac();
    }
    ET0 = 0x0;                  //disable Timer0 interrupt
    EKBI = 0x1;                 //enable KBI interrupt
    EA = 0x01;                  //enable global interrupt
}
//*****************************************************************************
//* Name: T1_ISR()
//* Input(s) : none.
//* Returns : none.
//* Description : Interrupt from Timer 1
//*****************************************************************************
void T1_ISR(void) interrupt 3   //T1 interrupt vector address is 001Bh
{
    TF1 = 0x0;              //clear TF1 bit
    /* key1 sampling */
    if ((key1 == 0) && (key1_flag == 0))
    {
        key1_flag = 1;
    }
    if ((key1 == 0) && (key1_flag == 1))
    {
        key1_flag = 2;
    }
    if ((key1 == 1) && (key1_flag == 2))
    {
        key1_flag = 3;
    }
    /* key2 sampling */
    if ((key2 == 0) && (key2_flag == 0))
    {
        key2_flag = 1;
    }
    if ((key2 == 0) && (key2_flag == 1))
    {
        key2_flag = 2;
    }
    if ((key2 == 1) && (key2_flag == 2))
    {
        key2_flag = 3;
    }
    TH1 = 0x0;                  //reload Timer1 value to be maximum(about 18ms)
    TL1 = 0x0;
}
//*****************************************************************************
```

```
//* Name: KBI_ISR()
//* Input(s) : none.
//* Returns : none.
//* Description : Interrupt from key pad pins
//****************************************************************************
void KBI_ISR(void) interrupt 7   //KBI interrupt vector address is 003Bh
{
      EA = 0x0;                               //disable global interrupt
      while(KBCON & 0x01 != 0x0)
      {
            KBCON = 0x0;                //clear KBIF bit
      }
      //Inverse the interrupt edge
      TL0 = PHASE & 0xff;
      TH0 = (PHASE >> 8) & 0xff;  //set Timer0 value
      if (curr_flag == 0)                     //falling edge caused interrupt
      {
            curr_flag = 1;
            while(KBPATN != 0x0)
            {
                  KBPATN = 0x0;
            }
      }
      else if (curr_flag != 0)          //rising edge caused interrupt
      {
            curr_flag = 0;
            while(KBPATN != 0x20)
            {
                  KBPATN = 0x20;
            }
      }
      //harmonic reducing flag
      harm_flag++;
      if (harm_flag > 3)
      {
            harm_flag = 0;
      }
      //soft start control
      if (harm_flag == 3)
      {
            update++;
            if (update > update_rate)
            {
                  update = 0;
            }
      }
      syn = 1;
      ET0 = 0x1;                        //enable Timer0 interrupt
      EKBI = 0x0;                       //disable KBI interrupt
      EA = 0x01;                        //enable global interrupt
}
```

```
//****************************************************************************
//* Name: fire_triac()
//* Input(s) : none.
//* Returns : none.
//* Description : fire TRIAC with P1.2, P3.0 and P3.1 together
//****************************************************************************
void fire_triac(void)
{
     Port1 = 0;
     Port2 = 0;
     Port3 = 0;
     if (PHASE > 0xF000)
     {
          delay(400);          //long fire pulse delay
     }
     else
     {
          delay(200);          //short fire pulse delay
     }
     Port1 = 1;
     Port2 = 1;
     Port3 = 1;
     return;
}
//****************************************************************************
//* Name: get_speed()
//* Input(s) : none.
//* Returns : unsigned int phase_value.
//* Description : control routine in geting adc value, process softstart and look
//* up phase value according to the adc value.
//****************************************************************************
unsigned int get_speed(void)
{
     unsigned int adc_value, timer_value, phase_value;
     adc_value = get_ADC();
     adc_value = softswitch(adc_value);
     //Converse adc value into timer reload value
     phase_value = 140 - (adc_value * 3); //adc value to fire phase
     if ((phase_value > 40)&&(phase_value < 140))
     {
          phase_value = harm_reduce(phase_value);
     }
     timer_value = phase2timer(phase_value);
     return timer_value;
}
//****************************************************************************
//* Name: get_ADC()
//* Input(s) : none.
//* Returns : unsigned int adc_value.
//* Description : get adc value from the I/O port P0.4 and P1.5,
//*              if press Key1 only, the adc_value will increas
```

```
//*             if press Key2 only, the adc_value will decrease
//*             if press Key1 and Key2 together, the adc_value will be given
//*             with the maxiam value. That will preform softstart demo.
//*****************************************************************************
unsigned int get_ADC(void)
{
      unsigned int adc_value;
      if ((key1_flag == 3) && (key2_flag != 3) && (key2_flag != 2))
      {
            if (key_value < speed_max)
            {
                  key_value++;     //if key1 is pressed, speed increase.
            }
            key1_flag = 0;
      }
      if ((key2_flag == 3) && (key1_flag != 3) && (key1_flag != 2))
      {
            if (key_value > 0x0)
            {
                  key_value--;     //if key2 is pressed, speed decrease.
            }
            key2_flag = 0;
      }
      if ((key1_flag == 2) && (key2_flag == 2))
      {
key_value = speed_max;      //if press two keys together, go directly to maximum
//speed.
            key1_flag = 0;
            key2_flag = 0;
      }
      adc_value = key_value;
      return adc_value;
}
//*****************************************************************************
//* Name: phase_lookup()
//* Input(s) : unsigned int adc_value.
//* Returns : unsigned int phase_value.
//* Description : convert adc_value into Timer 0 reload phase value
//*****************************************************************************
unsigned int harm_reduce(unsigned int phase_value)
{
      unsigned int new_phase_value;
      switch (harm_flag)
      {
            case 0:
                  new_phase_value = phase_value - 30;

                  break;
            case 1:
                  new_phase_value = phase_value - 30;

```

```
                break;
            case 2:
                new_phase_value = phase_value + 30;
                break;
            case 3:
                new_phase_value = phase_value + 30;
                break;
            default:
                harm_flag = 0;
                break;
        }
        return new_phase_value;
}
//*****************************************************************************
//* Name: softswitch()
//* Input(s) : unsigned int adc_value.
//* Returns : unsigned int adc_new.
//* Description : This method process soft-switch algorithm of the vacuum
//*****************************************************************************
unsigned int softswitch(unsigned int adc_value)
{
        unsigned int adc_new;
        if ((adc_old > adc_value) || (adc_value > adc_old) && (update ==
         update_rate))
        {
            if (adc_old > adc_value)
            {
                adc_new = adc_old - one_step;
            }
            else
            {
                adc_new = adc_old + one_step;
            }
            update = 0;
        }
        else
        {
            adc_new = adc_old;
        }
        adc_old = adc_new;
        return adc_new;
}
//*****************************************************************************
//* Name: delay()
//* Input(s) : unsigned int cnt.
//* Returns : none.
//* Description : process delay function
//*****************************************************************************
void delay (unsigned int cnt)
{
  while (--cnt);
```

```
}
//****************************************************************************
//* Name: phase2timer()
//* Input(s) : unsigned int phase_value.
//* Returns : unsigned int timer_value.
//* Description : This method process conversion from phase value into timer0
//* reload value
//****************************************************************************
unsigned int phase2timer(unsigned int phase_value)
{
     unsigned int timer_value;
     timer_value = 0xffff - (phase_value * 0xba);
     return timer_value;
}


/*--------------------------------------------------------------------------
REG901.H

Header file for Philips 89LPC901
--------------------------------------------------------------------------*/

#ifndef __REG901_H__
#define __REG901_H__

/*  BYTE Registers  */
sfr P0     = 0x80;
sfr P0M1   = 0x84;
sfr P0M2   = 0x85;

sfr P1     = 0x90;
sfr P1M1   = 0x91;
sfr P1M2   = 0x92;

sfr P3     = 0xB0;
sfr P3M1   = 0xB1;
sfr P3M2   = 0xB2;
//------------------
sfr PSW    = 0xD0;
sfr ACC    = 0xE0;
sfr B      = 0xF0;
sfr SP     = 0x81;
sfr DPL    = 0x82;
sfr DPH    = 0x83;
//------------------
sfr AUXR1  = 0xA2;
sfr CMP1   = 0xAC;
sfr DIVM   = 0x95;

sfr FMADRH = 0xE7;
sfr FMADRL = 0xE6;
```

```
sfr FMCON  = 0xE4;
sfr FMDATA = 0xE5;

sfr IEN0   = 0xA8;
sfr IEN1   = 0xE8;

sfr IP0    = 0xB8;
sfr IP0H   = 0xB7;
sfr IP1    = 0xF8;
sfr IP1H   = 0xF7;

sfr KBCON  = 0x94;
sfr KBMASK = 0x86;
sfr KBPATN = 0x93;

sfr PCON   = 0x87;
sfr PCONA  = 0xB5;
sfr PCONB  = 0xB6;

sfr PT0AD  = 0xF6;
sfr RSTSRC = 0xDF;

sfr RTCCON = 0xD1;
sfr RTCH   = 0xD2;
sfr RTCL   = 0xD3;

sfr TAMOD  = 0x8F;
sfr TCON   = 0x88;
sfr TL0    = 0x8A;
sfr TL1    = 0x8B;
sfr TH0    = 0x8C;
sfr TH1    = 0x8D;
sfr TMOD   = 0x89;
sfr TRIM   = 0x96;

sfr WDCON  = 0xA7;
sfr WDL    = 0xC1;
sfr WFEED1 = 0xC2;
sfr WFEED2 = 0xC3;

/*  BIT Registers  */
/*  PSW   */
sbit CY   = PSW^7;
sbit AC   = PSW^6;
sbit F0   = PSW^5;
sbit RS1  = PSW^4;
sbit RS0  = PSW^3;
sbit OV   = PSW^2;
sbit F1   = PSW^1;
sbit P    = PSW^0;

```

```
/*  TCON  */
sbit TF1    = TCON^7;
sbit TR1    = TCON^6;
sbit TF0    = TCON^5;
sbit TR0    = TCON^4;

/*  IEN0   */
sbit EA     = IEN0^7;
sbit EWDRT  = IEN0^6;
sbit EBO    = IEN0^5;
sbit ET1    = IEN0^3;
sbit ET0    = IEN0^1;

/*  IEN1   */
sbit EC     = IEN1^2;
sbit EKBI   = IEN1^1;

/*  IP0   */
sbit PWDRT  = IP0^6;
sbit PB0    = IP0^5;
sbit PT1    = IP0^3;
sbit PT0    = IP0^1;

/*  P0  */
sbit KB5    = P0^5;
sbit CMPREF = P0^5;
sbit KB4    = P0^4;
sbit CIN1A  = P0^4;

/*  P1  */
sbit RST     = P1^5;
sbit T0      = P1^2;

/*  P3  */
sbit XTAL1= P3^1;
sbit XTAL2= P3^0;

#endif
```

15. Legal information

15.1 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, Philips Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — Philips Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — Philips Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a Philips Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. Philips Semiconductors accepts no liability for inclusion and/or use of Philips Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

15.2 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

DE 19705907C1 (German Patent) — owned by Gerhard Kurz Gmbh

EP 0859452B1 (European Patent) — owned by Gerhard Kurz Gmbh

15.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

16. Contents

Date of release: 10 August 2006
Document identifier: AN10496_1