Important notice

Dear Customer,

On 7 February 2017 the former NXP Standard Product business became a new company with the tradename **Nexperia**. Nexperia is an industry leading supplier of Discrete, Logic and PowerMOS semiconductors with its focus on the automotive, industrial, computing, consumer and wearable application markets

In data sheets and application notes which still contain NXP or Philips Semiconductors references, use the references to Nexperia, as shown below.

Instead of http://www.nxp.com, http://www.philips.com/ or http://www.semiconductors.philips.com/, use **http://www.nexperia.com**

Instead of sales.addresses@www.nxp.com or sales.addresses@www.semiconductors.philips.com, use **salesaddresses@nexperia.com** (email)

Replace the copyright notice at the bottom of each page or elsewhere in the document, depending on the version, as shown below:
- © NXP N.V. (year). All rights reserved or © Koninklijke Philips Electronics N.V. (year). All rights reserved
Should be replaced with:
- **© Nexperia B.V. (year). All rights reserved**.

If you have any questions related to the data sheet, please contact our nearest sales office via e-mail or telephone (details via **salesaddresses@nexperia.com**). Thank you for your cooperation and understanding,

Kind regards,

Team Nexperia

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.

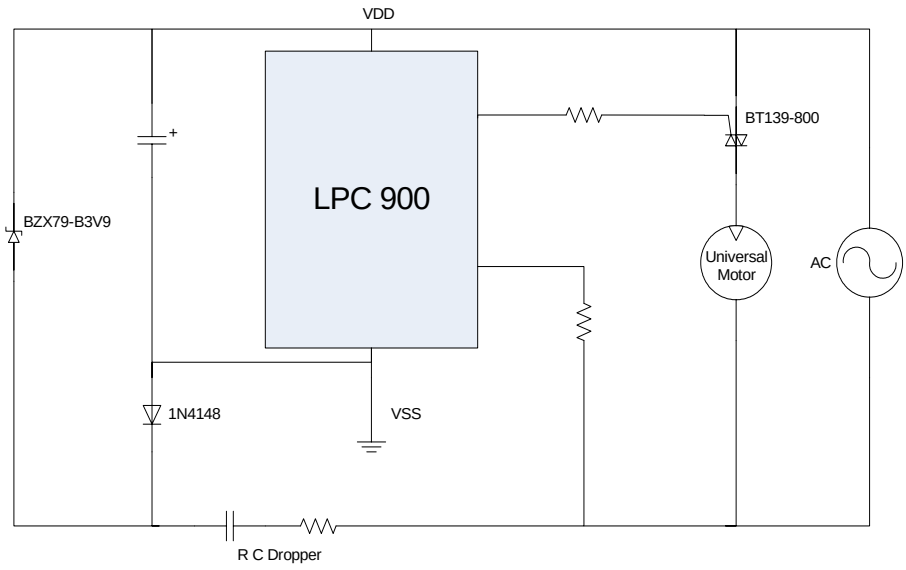

Fig 1. Vacuum cleaner reference design board

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.

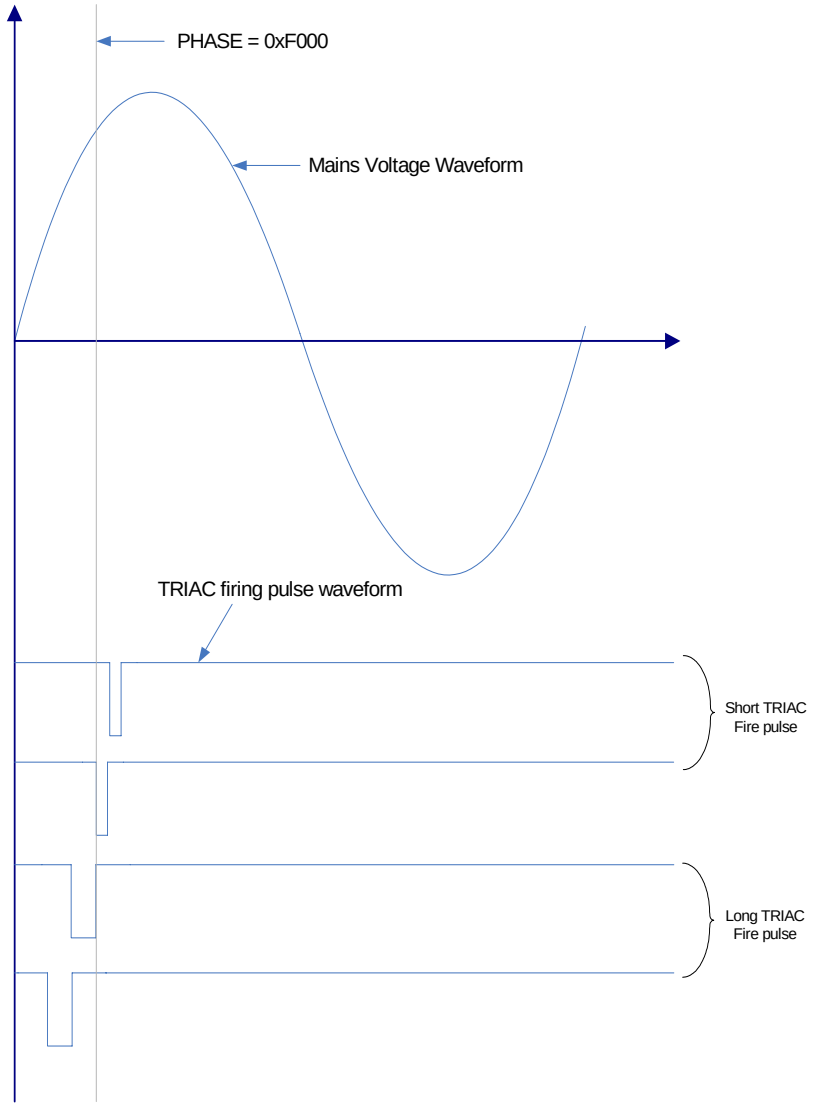

Fig 2. Long fire waveform

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.

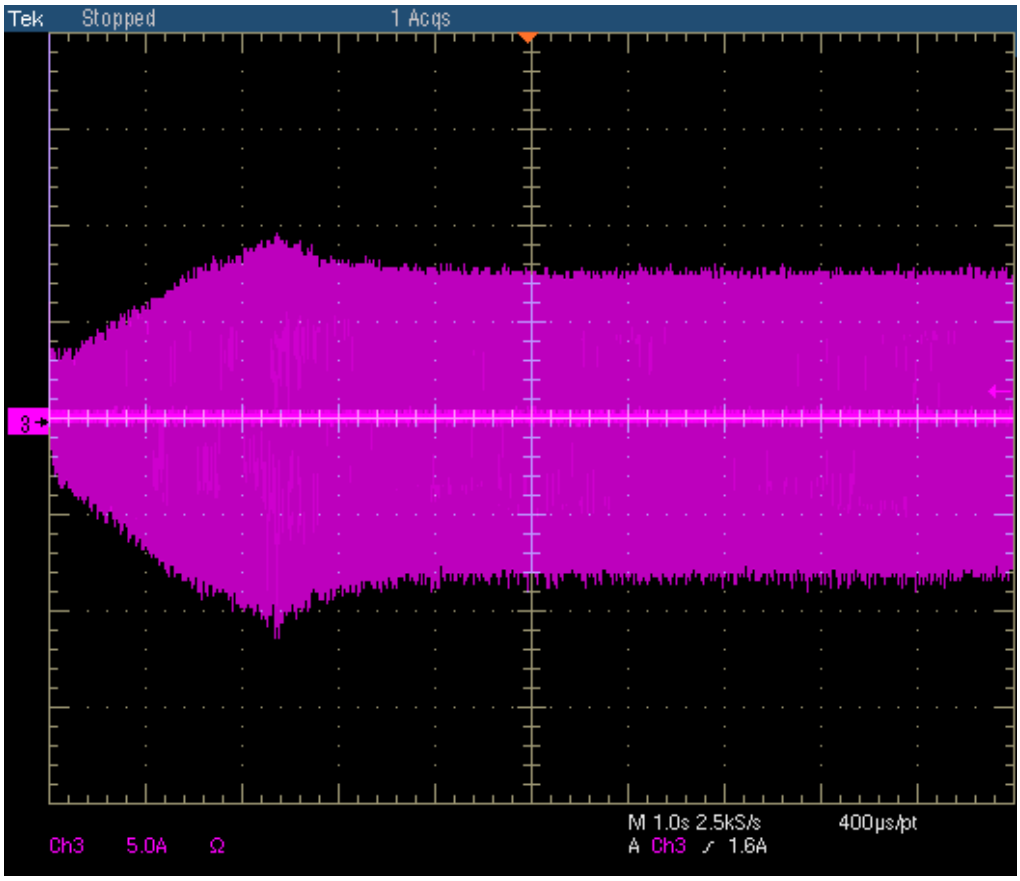

Fig 3. Start-up delay demo

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).

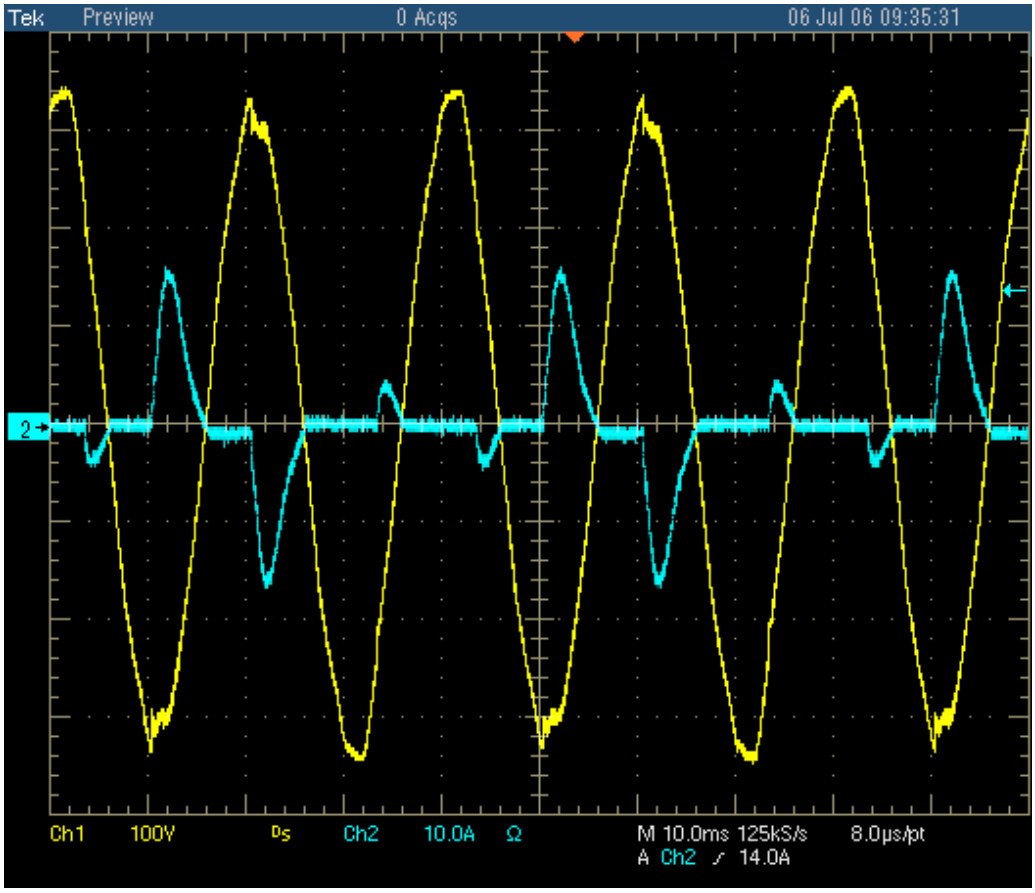

Fig 4. Harmonic reducing demo

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

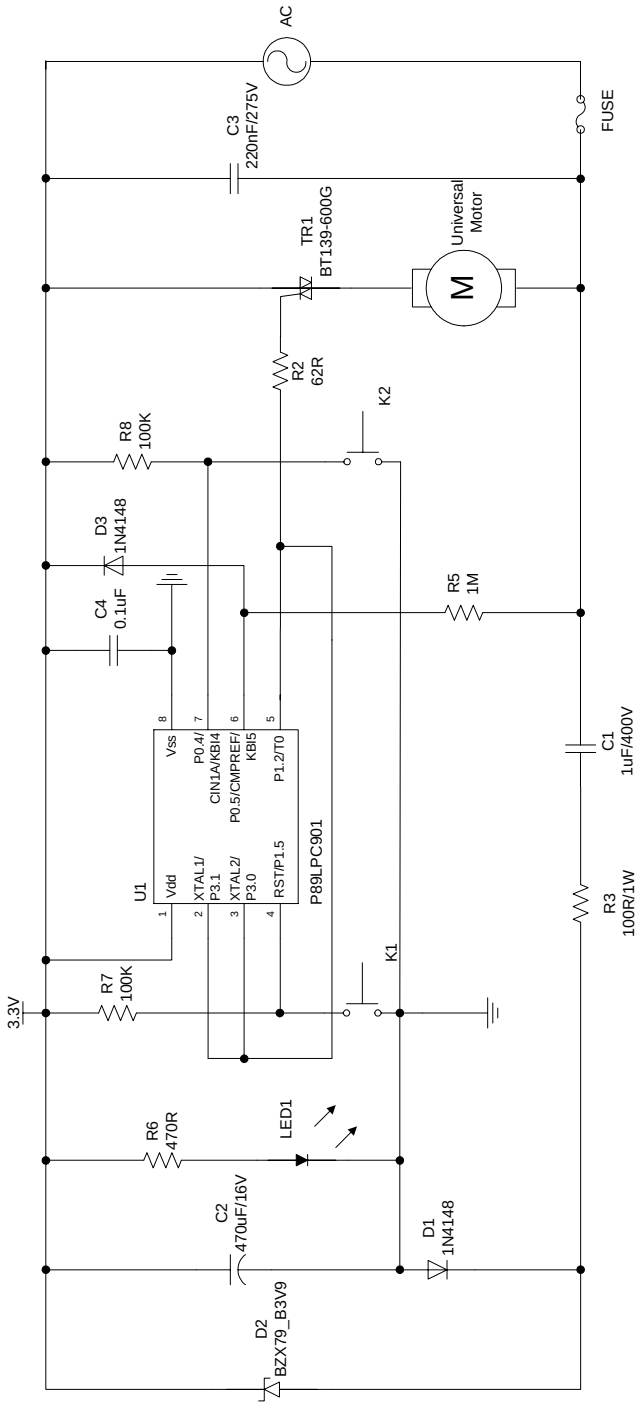

Fig 5. Detailed schematics of the vacuum cleaner circuit using the P89LPC901 microcontroller

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

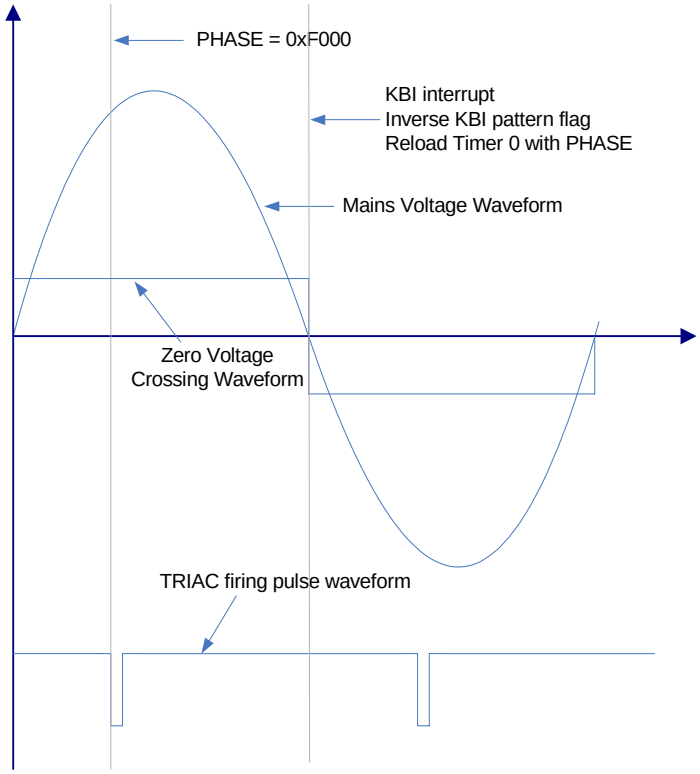

Fig 6. Detailed KBI controlling waveform

8. Appendix C

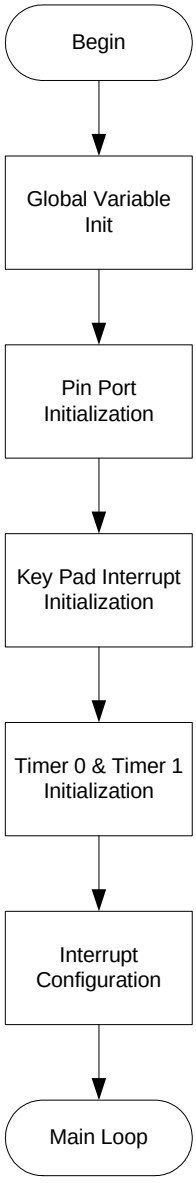

Fig 7. Init subroutine flowchart

9. Appendix D

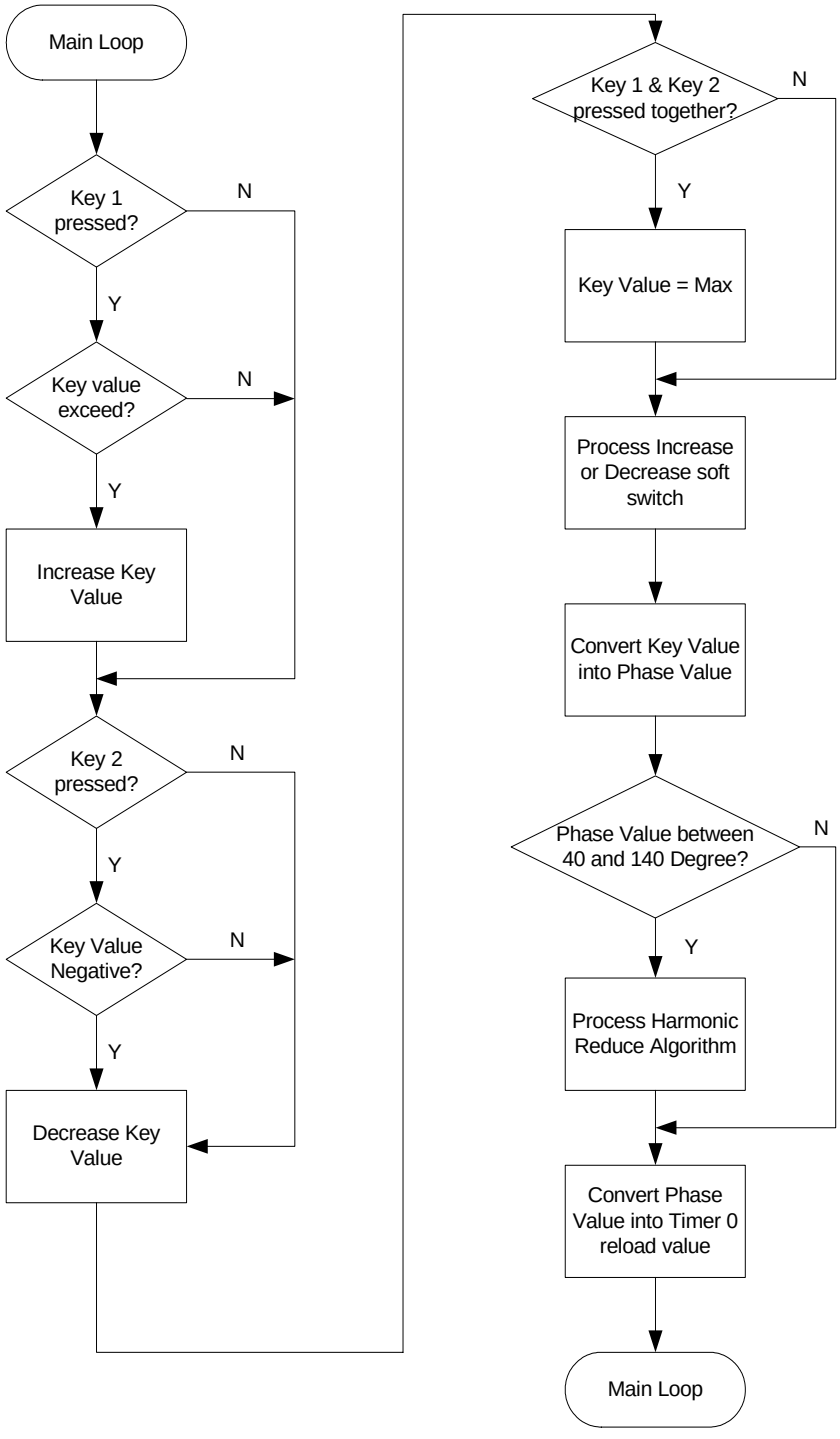

Fig 8. Main loop flowchart

10. Appendix E

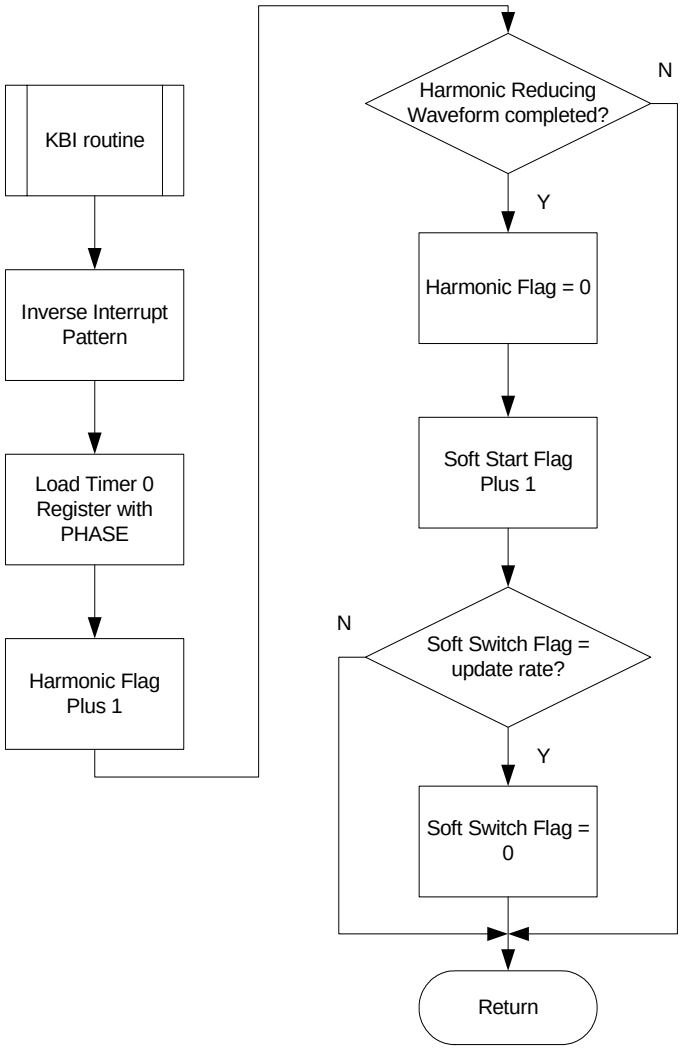

Fig 9. KBI subroutine flowchart

11. Appendix F

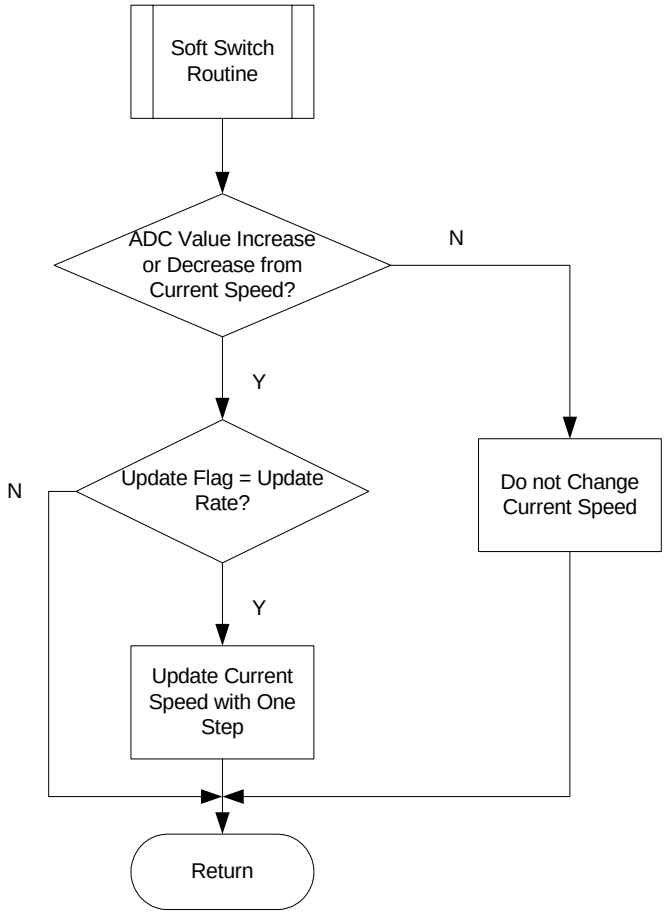

Fig 10. Softswitch subroutine flowchart

12. Appendix G

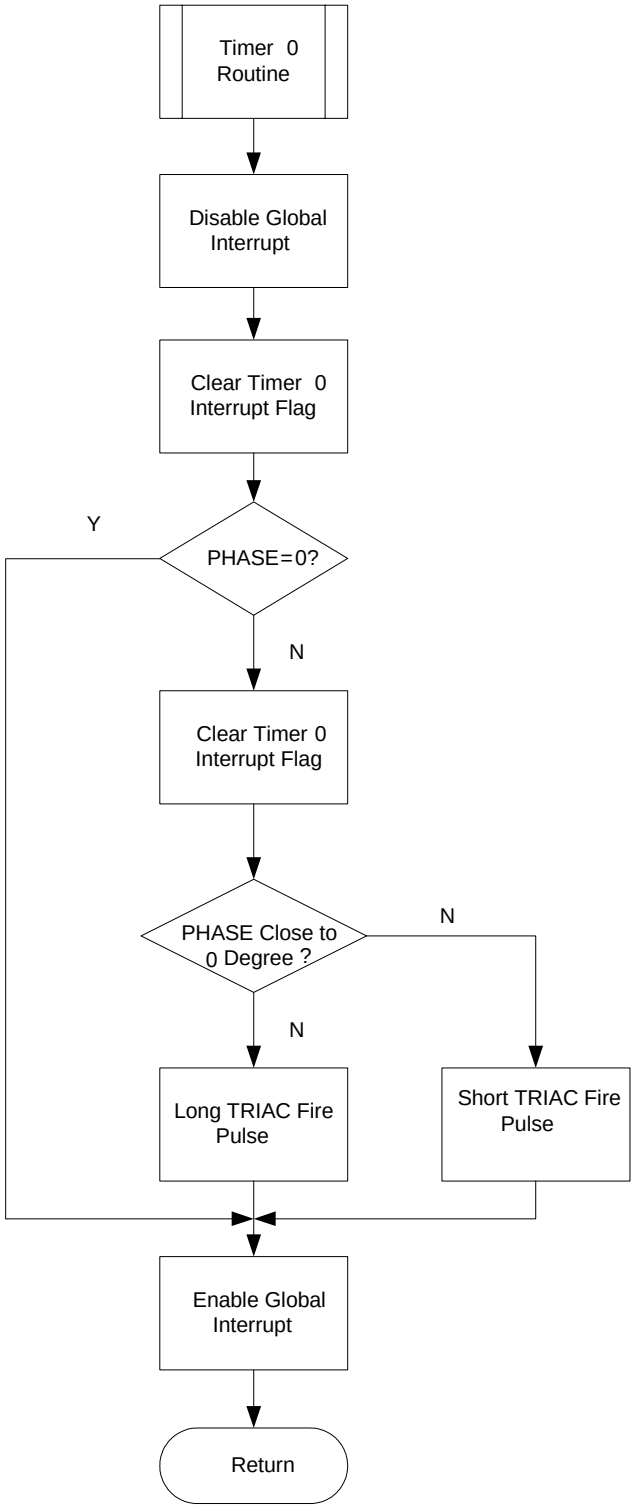

Fig 11. Timer0 subroutine flowchart

13. Appendix H

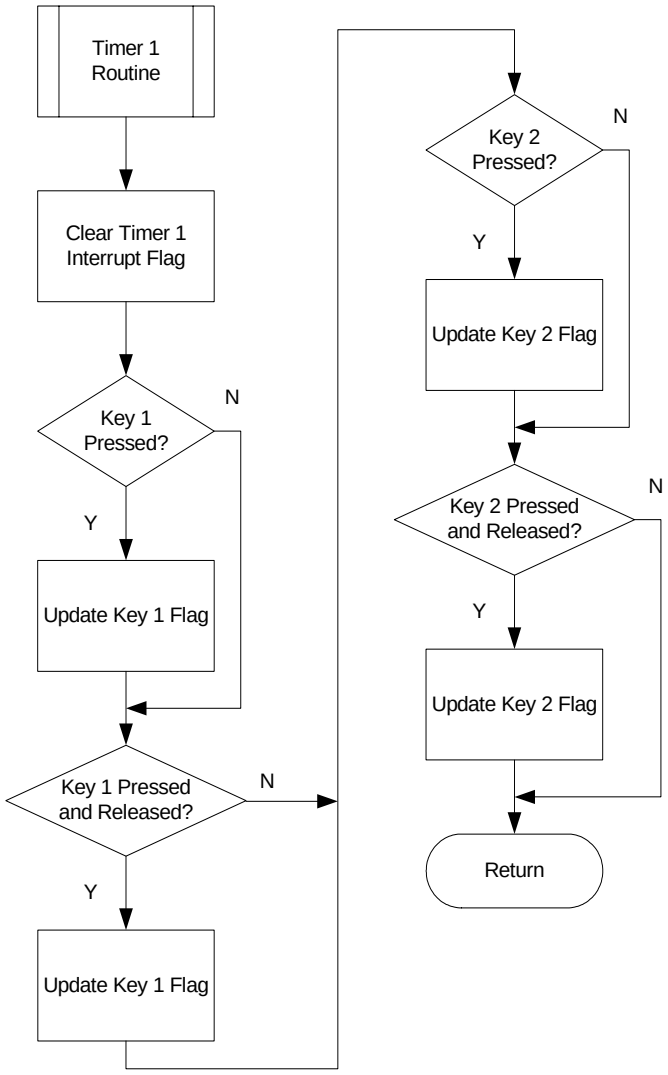

Fig 12. Timer1 subroutine flowchart

14. Appendix I

```
1   //*************************************************************************
2   //*
3   //* vacuum.c
4   //* Date : July 2005
5   //* Description : Vacuum Cleaner demo program with Philips P89LPC901
6   //*
7   //********************Revise History*************************************
8   //* Date : July 2005
9   //* Description : Creat
10  //*************************************************************************
11  #include <REG901.H>             // register definition
12  //-----------------------------------------------
13  //Port Pin Definitions
14  //-----------------------------------------------
15  sbit key1 = P0^4;
16  sbit key2 = P1^5;
17  sbit Port1 = P1^2;
18  sbit Port2 = P3^0;
19  sbit Port3 = P3^1;
20  //-----------------------------------------------
21  //Global Variable Definitions
22  //-----------------------------------------------
23  volatile unsigned char curr_flag;
24  volatile unsigned char syn;
25  volatile unsigned int PHASE;
26  volatile unsigned int adc_old;
27  volatile unsigned char one_step;
28  volatile unsigned char harm_flag;
29  volatile unsigned char update;
30  volatile unsigned char key1_flag;
31  volatile unsigned char key2_flag;
32  volatile unsigned char key_value;
33  volatile unsigned char speed_max;
34  //-----------------------------------------------
35  //Constant Definitions
36  //-----------------------------------------------
37  #define update_rate  2
38
39  //*************************************************************************
40  //* Functions
41  //*************************************************************************
42  void init(void);
43  void startupdelay (unsigned int degree, unsigned int cnt);
44  unsigned int phase2timer(unsigned int phase_value);
45  void KBI_ISR(void);
46  void T0_ISR(void);
47  void T1_ISR(void);
48  void fire_triac(void);
49  unsigned int get_speed(void);
50  unsigned int harm_reduce(unsigned int phase_value);
```

```
51   unsigned int get_ADC(void);
52   unsigned int softswitch(unsigned int adc_value);
53   void delay (unsigned int cnt);
54   //*************************************************************************
55   //* Name: main()
56   //* Input(s) : none.
57   //* Returns : none.
58   //* Description : main loop
59   //*************************************************************************
60   void main()
61   {
62        init();
63        startupdelay(140,8);                  //wait for startup stable
64        while(1)
65        {
66             PHASE = get_speed();
67        }
68   }
69   //*************************************************************************
70   //* Name: init()
71   //* Input(s) : none.
72   //* Returns : none.
73   //* Description : initialization of P89LPC901
74   //*************************************************************************
75   void init(void)
76   {
77   curr_flag = 0;
78   syn = 0;
79   PHASE = 0;
80   adc_old = 0x0;
81   one_step = 1;
82   harm_flag = 0;
83   update = 0;
84   key1_flag = 0;
85   key2_flag = 0;
86   key_value = 0x0;
87   speed_max = 46;  //from 0~140 degree, set as 46 levels, so 3 degree a level
88   /* Pin configuration */
89   P0M1 = 0x20;
90   P0M2 = 0x0;      //pin 7 (P0.4) as Quasi-bidir. & pin 6 (P0.5) as input only
91   P1M1 = 0x0;
92   P1M2 = 0x04;              //pin 5 (P1.2) as Push-Pull
93   P3M1 = 0x0;
94   P3M2 = 0x03;              //pin 2, 3 (P3.1, P3.0) as Push-Pull
95   /* KBI configuration */
96   KBMASK = 0x20;        //P0.5 as keypad interrupt
97   KBPATN = 0x20;        //pattern is high-level
98   KBCON = 0x0; //when signal in P0.5 is not equal to high-level, generate interrupt
99   /* Timer configuration */
100       TMOD = 0x11;          //Timer0 as Mode 1 and Timer1 as Mode 1
101       TAMOD = 0x0;          //16 bit mode
```

```
102       TH0 = 0x0;            //init Timer0 value to be maximum(about 18ms)
103       TL0 = 0x0;            //note: The timer counts up
104       TR0 = 0x01;           //run Timer0
105       TH1 = 0x0;            //init Timer1 value to be maximum(about 18ms)
106       TL1 = 0x0;
107       TR1 = 0x01;           //run Timer1
108       /* Interrupt configuration */
109       EA = 0x01;            //enable global interrupt
110       ET0 = 0x1;            //enable Timer0 interrupt
111       EKBI = 0x1;           //enable KBI interrupt
112       ET1 = 0x01;           //enable Timer1 interrupt as key input
113       IP1H = 0x02;          //set KBI interrupt priority as level 3 (highest)
114       IP1 = 0x02;
115       IP0H = 0x02;          //set Timer0 interrupt priority as level 3 (highest)
116       IP0 = 0x02;           //and Timer1 interrupt priority as level 0 (lowest)
117       /* close all other interrupt */
118       EBO = 0x0;
119       EWDRT = 0x0;
120       EC = 0x0;
121       /* RC Oscillator */
122       DIVM = 0x00;               //Fcpu = Fosc / (2 * DIVM)
123  }
124  //*************************************************************************
125  //* Name: startupdelay()
126  //* Input(s) : unsigned int degree, unsigned int cnt.
127  //* Returns : none.
128  //* Description : provide softstart function
129  //*************************************************************************
130  void startupdelay (unsigned int degree, unsigned int cnt)
131  {
132       unsigned int i,j;
133       for(i=170;i > degree;i--)
134       {
135            for(j=0;j<cnt;j++)
136            {
137                 while(!syn);
138                 syn = 0;
139            }
140            PHASE = phase2timer(i);
141       }
142  }
143  //*************************************************************************
144  //* Name: T0_ISR()
145  //* Input(s) : none.
146  //* Returns : none.
147  //* Description : Interrupt from Timer 0
148  //*************************************************************************
149  void T0_ISR(void) interrupt 1    //T0 interrupt vector address is 000Bh
150  {
151       EA = 0x0;                  //disable global interrupt
152       while(TF0 != 0x0)
```

```
153       {
154            TF0 = 0x0;            //clear TF0 bit
155       }
156       //fire TRIAC
157       if(PHASE != 0x0000)
158       {
159            fire_triac();
160       }
161       ET0 = 0x0;                 //disable Timer0 interrupt
162       EKBI = 0x1;                //enable KBI interrupt
163       EA = 0x01;                 //enable global interrupt
164  }
165  //*************************************************************************
166  //* Name: T1_ISR()
167  //* Input(s) : none.
168  //* Returns : none.
169  //* Description : Interrupt from Timer 1
170  //*************************************************************************
171  void T1_ISR(void) interrupt 3    //T1 interrupt vector address is 001Bh
172  {
173       TF1 = 0x0;            //clear TF1 bit
174       /* key1 sampling */
175       if ((key1 == 0) && (key1_flag == 0))
176       {
177            key1_flag = 1;
178       }
179       if ((key1 == 0) && (key1_flag == 1))
180       {
181            key1_flag = 2;
182       }
183       if ((key1 == 1) && (key1_flag == 2))
184       {
185            key1_flag = 3;
186       }
187       /* key2 sampling */
188       if ((key2 == 0) && (key2_flag == 0))
189       {
190            key2_flag = 1;
191       }
192       if ((key2 == 0) && (key2_flag == 1))
193       {
194            key2_flag = 2;
195       }
196       if ((key2 == 1) && (key2_flag == 2))
197       {
198            key2_flag = 3;
199       }
200       TH1 = 0x0;                 //reload Timer1 value to be maximum(about 18ms)
201       TL1 = 0x0;
202  }
203  //*************************************************************************
```

```
204  //* Name: KBI_ISR()
205  //* Input(s) : none.
206  //* Returns : none.
207  //* Description : Interrupt from key pad pins
208  //*****************************************************************************
209  void KBI_ISR(void) interrupt 7   //KBI interrupt vector address is 003Bh
210  {
211       EA = 0x0;                          //disable global interrupt
212       while(KBCON & 0x01 != 0x0)
213       {
214            KBCON = 0x0;               //clear KBIF bit
215       }
216       //Inverse the interrupt edge
217       TL0 = PHASE & 0xff;
218       TH0 = (PHASE >> 8) & 0xff; //set Timer0 value
219       if (curr_flag == 0)                //falling edge caused interrupt
220       {
221            curr_flag = 1;
222            while(KBPATN != 0x0)
223            {
224                 KBPATN = 0x0;
225            }
226       }
227       else if (curr_flag != 0)        //rising edge caused interrupt
228       {
229            curr_flag = 0;
230            while(KBPATN != 0x20)
231            {
232                 KBPATN = 0x20;
233            }
234       }
235       //harmonic reducing flag
236       harm_flag++;
237       if (harm_flag > 3)
238       {
239            harm_flag = 0;
240       }
241       //soft start control
242       if (harm_flag == 3)
243       {
244            update++;
245            if (update > update_rate)
246            {
247                 update = 0;
248            }
249       }
250       syn = 1;
251       ET0 = 0x1;                    //enable Timer0 interrupt
252       EKBI = 0x0;                   //disable KBI interrupt
253       EA = 0x01;                    //enable global interrupt
254  }
```

```
255  //*****************************************************************************
256  //* Name: fire_triac()
257  //* Input(s) : none.
258  //* Returns : none.
259  //* Description : fire TRIAC with P1.2, P3.0 and P3.1 together
260  //*****************************************************************************
261  void fire_triac(void)
262  {
263       Port1 = 0;
264       Port2 = 0;
265       Port3 = 0;
266       if (PHASE > 0xF000)
267       {
268            delay(400);           //long fire pulse delay
269       }
270       else
271       {
272            delay(200);           //short fire pulse delay
273       }
274       Port1 = 1;
275       Port2 = 1;
276       Port3 = 1;
277       return;
278  }
279  //*****************************************************************************
280  //* Name: get_speed()
281  //* Input(s) : none.
282  //* Returns : unsigned int phase_value.
283  //* Description : control routine in geting adc value, process softstart and look
284  //* up phase value according to the adc value.
285  //*****************************************************************************
286  unsigned int get_speed(void)
287  {
288       unsigned int adc_value, timer_value, phase_value;
289       adc_value = get_ADC();
290       adc_value = softswitch(adc_value);
291       //Converse adc value into timer reload value
292       phase_value = 140 - (adc_value * 3); //adc value to fire phase
293       if ((phase_value > 40)&&(phase_value < 140))
294       {
295            phase_value = harm_reduce(phase_value);
296       }
297       timer_value = phase2timer(phase_value);
298       return timer_value;
299  }
300  //*****************************************************************************
301  //* Name: get_ADC()
302  //* Input(s) : none.
303  //* Returns : unsigned int adc_value.
304  //* Description : get adc value from the I/O port P0.4 and P1.5,
305  //*             if press Key1 only, the adc_value will increas
```

```
306  //*             if press Key2 only, the adc_value will decrease
307  //*             if press Key1 and Key2 together, the adc_value will be given
308  //*             with the maxiam value. That will preform softstart demo.
309  //*************************************************************************
310  unsigned int get_ADC(void)
311  {
312       unsigned int adc_value;
313       if ((key1_flag == 3) && (key2_flag != 3) && (key2_flag != 2))
314       {
315            if (key_value < speed_max)
316            {
317                 key_value++;     //if key1 is pressed, speed increase.
318            }
319            key1_flag = 0;
320       }
321       if ((key2_flag == 3) && (key1_flag != 3) && (key1_flag != 2))
322       {
323            if (key_value > 0x0)
324            {
325                 key_value--;     //if key2 is pressed, speed decrease.
326            }
327            key2_flag = 0;
328       }
329       if ((key1_flag == 2) && (key2_flag == 2))
330       {
331  key_value = speed_max;      //if press two keys together, go directly to maximum
332  //speed.
333            key1_flag = 0;
334            key2_flag = 0;
335       }
336       adc_value = key_value;
337       return adc_value;
338  }
339  //*************************************************************************
340  //* Name: phase_lookup()
341  //* Input(s) : unsigned int adc_value.
342  //* Returns : unsigned int phase_value.
343  //* Description : convert adc_value into Timer 0 reload phase value
344  //*************************************************************************
345  unsigned int harm_reduce(unsigned int phase_value)
346  {
347       unsigned int new_phase_value;
348       switch (harm_flag)
349       {
350            case 0:
351                 new_phase_value = phase_value - 30;
352
353                 break;
354            case 1:
355                 new_phase_value = phase_value - 30;
356
```

```
357                 break;
358           case 2:
359                 new_phase_value = phase_value + 30;
360                 break;
361           case 3:
362                 new_phase_value = phase_value + 30;
363                 break;
364           default:
365                 harm_flag = 0;
366                 break;
367      }
368      return new_phase_value;
369 }
370 //*****************************************************************************
371 //* Name: softswitch()
372 //* Input(s) : unsigned int adc_value.
373 //* Returns : unsigned int adc_new.
374 //* Description : This method process soft-switch algorithm of the vacuum
375 //*****************************************************************************
376 unsigned int softswitch(unsigned int adc_value)
377 {
378      unsigned int adc_new;
379      if ((adc_old > adc_value) || (adc_value > adc_old) && (update ==
380       update_rate))
381      {
382           if (adc_old > adc_value)
383           {
384                adc_new = adc_old - one_step;
385           }
386           else
387           {
388                adc_new = adc_old + one_step;
389           }
390           update = 0;
391      }
392      else
393      {
394           adc_new = adc_old;
395      }
396      adc_old = adc_new;
397      return adc_new;
398 }
399 //*****************************************************************************
400 //* Name: delay()
401 //* Input(s) : unsigned int cnt.
402 //* Returns : none.
403 //* Description : process delay function
404 //*****************************************************************************
405 void delay (unsigned int cnt)
406 {
407   while (--cnt);
```

```
408  }
409  //*************************************************************************
410  //* Name: phase2timer()
411  //* Input(s) : unsigned int phase_value.
412  //* Returns : unsigned int timer_value.
413  //* Description : This method process conversion from phase value into timer0
414  //* reload value
415  //*************************************************************************
416  unsigned int phase2timer(unsigned int phase_value)
417  {
418       unsigned int timer_value;
419       timer_value = 0xffff - (phase_value * 0xba);
420       return timer_value;
421  }
422
423
424  /*-------------------------------------------------------------------------
425  REG901.H
426
427  Header file for Philips 89LPC901
428  -------------------------------------------------------------------------*/
429
430  #ifndef __REG901_H__
431  #define __REG901_H__
432
433  /*  BYTE Registers  */
434  sfr P0     = 0x80;
435  sfr P0M1   = 0x84;
436  sfr P0M2   = 0x85;
437
438  sfr P1     = 0x90;
439  sfr P1M1   = 0x91;
440  sfr P1M2   = 0x92;
441
442  sfr P3     = 0xB0;
443  sfr P3M1   = 0xB1;
444  sfr P3M2   = 0xB2;
445  //------------------
446  sfr PSW    = 0xD0;
447  sfr ACC    = 0xE0;
448  sfr B      = 0xF0;
449  sfr SP     = 0x81;
450  sfr DPL    = 0x82;
451  sfr DPH    = 0x83;
452  //------------------
453  sfr AUXR1  = 0xA2;
454  sfr CMP1   = 0xAC;
455  sfr DIVM   = 0x95;
456
457  sfr FMADRH = 0xE7;
458  sfr FMADRL = 0xE6;
```

```
459  sfr FMCON  = 0xE4;
460  sfr FMDATA = 0xE5;
461
462  sfr IEN0   = 0xA8;
463  sfr IEN1   = 0xE8;
464
465  sfr IP0    = 0xB8;
466  sfr IP0H   = 0xB7;
467  sfr IP1    = 0xF8;
468  sfr IP1H   = 0xF7;
469
470  sfr KBCON  = 0x94;
471  sfr KBMASK = 0x86;
472  sfr KBPATN = 0x93;
473
474  sfr PCON   = 0x87;
475  sfr PCONA  = 0xB5;
476  sfr PCONB  = 0xB6;
477
478  sfr PT0AD  = 0xF6;
479  sfr RSTSRC = 0xDF;
480
481  sfr RTCCON = 0xD1;
482  sfr RTCH   = 0xD2;
483  sfr RTCL   = 0xD3;
484
485  sfr TAMOD  = 0x8F;
486  sfr TCON   = 0x88;
487  sfr TL0    = 0x8A;
488  sfr TL1    = 0x8B;
489  sfr TH0    = 0x8C;
490  sfr TH1    = 0x8D;
491  sfr TMOD   = 0x89;
492  sfr TRIM   = 0x96;
493
494  sfr WDCON  = 0xA7;
495  sfr WDL    = 0xC1;
496  sfr WFEED1 = 0xC2;
497  sfr WFEED2 = 0xC3;
498
499  /*  BIT Registers  */
500  /*  PSW   */
501  sbit CY   = PSW^7;
502  sbit AC   = PSW^6;
503  sbit F0   = PSW^5;
504  sbit RS1  = PSW^4;
505  sbit RS0  = PSW^3;
506  sbit OV   = PSW^2;
507  sbit F1   = PSW^1;
508  sbit P    = PSW^0;
509
```

```
510  /*  TCON  */
511  sbit TF1    = TCON^7;
512  sbit TR1    = TCON^6;
513  sbit TF0    = TCON^5;
514  sbit TR0    = TCON^4;
515
516  /*  IEN0   */
517  sbit EA     = IEN0^7;
518  sbit EWDRT  = IEN0^6;
519  sbit EBO    = IEN0^5;
520  sbit ET1    = IEN0^3;
521  sbit ET0    = IEN0^1;
522
523  /*  IEN1   */
524  sbit EC     = IEN1^2;
525  sbit EKBI   = IEN1^1;
526
527  /*  IP0   */
528  sbit PWDRT  = IP0^6;
529  sbit PB0    = IP0^5;
530  sbit PT1    = IP0^3;
531  sbit PT0    = IP0^1;
532
533  /*  P0  */
534  sbit KB5    = P0^5;
535  sbit CMPREF = P0^5;
536  sbit KB4    = P0^4;
537  sbit CIN1A  = P0^4;
538
539  /*  P1  */
540  sbit RST     = P1^5;
541  sbit T0      = P1^2;
542
543  /*  P3  */
544  sbit XTAL1= P3^1;
545  sbit XTAL2= P3^0;
546
547  #endif
548
```

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

For more information, please visit: http://www.semiconductors.philips.com
For sales office addresses, email to: sales.addresses@www.semiconductors.philips.com

Date of release: 10 August 2006
Document identifier: AN10496_1