Freescale Semiconductor
Application Note

Document Number: AN3471
Rev. 0, 07/2008

Ceiling Fan Speed Control

Single-Phase Motor Speed Control Using MC9RS08KA2

by: Cuauhtemoc Medina
RTAC Americas

Contents

1 Introduction

This application note introduces a method for controlling a single-phase AC induction motor. This motor is widely used in ceiling fans due to various advantages over other types of motors. It is low cost, low maintenance, and has direct connection to the AC power source.

Using the MC9RS08KA2 MCU series combined with the basic TRIAC topology is cost-performance solution. The traditional mechanical speed control of the ceiling fan can be replaced with this solution avoiding problems such as non-linearity on speed.

2 Solution

2.1 Single Phase Induction Motor Control Theory

Single-phase induction motors are the most used. These motors have only one stator winding, operate with a single-phase power supply, and are also squirrel cage. Because of the single phase, the motor is not self-started when connected to a power supply. The necessary torque is not generated therefore causing the motor to only vibrate and not rotate.

To provide the starting torque most single-phase motors have a main and auxiliary winding, both in quadrature to help generate the phase-shifted magnetic field.

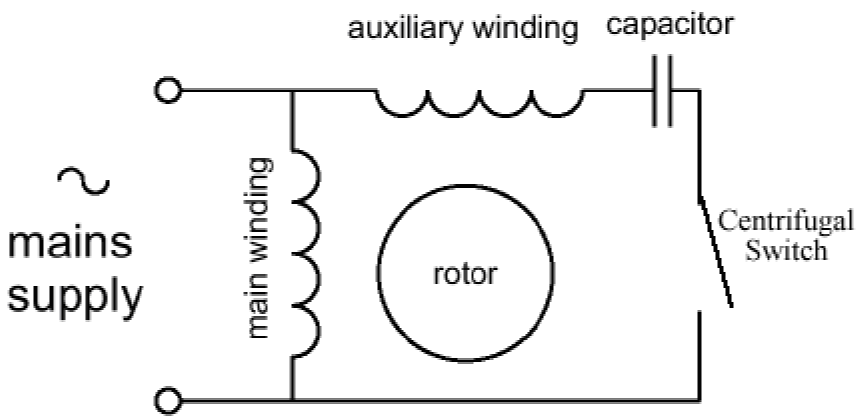

Figure 1. Capacitor Start AC Induction Motor

The auxiliary winding current from the main winding is phase-shifted. Connecting a capacitor in series with the auxiliary winding causes the motor to start rotating. Using a centrifugal switch disconnects the capacitor and the auxiliary winding at 75% of the motor nominal speed. This topology is used if high torque is required. In most fan motors, the capacitor and the auxiliary winding remain connected. This configuration is called permanent split capacitor (PSC) AC induction motor. No centrifugal switch is used and are considered to be the most reliable single-phase motors. At rated load, they can be designed for optimum efficiency and high power factor (PF).

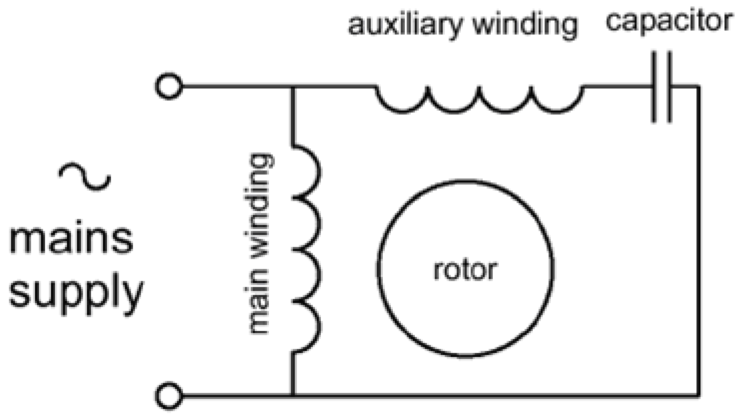

Figure 2. PSC Starting Mechanism

2.2 Typical Solutions

Motors commonly used in ceiling fans are single-phase induction motors with a PSC starting mechanism. Most of them have three different speeds that are mechanically selected by pulling a chain. Every time the chain is pulled, the motor circuit changes to a predefined coil winding that causes the speed to vary. It is recommended that the fan be set at maximum speed. Considering that the load of the motor is proportional to the consumed current it is not the same range of speed variation with the load then without it. The range of speed variation needs to be recalculated.

2.3 Proposed Solution and Phase Angle Control

When the TRIAC switch is connected between the AC power supply and the motor, the power flow can be controlled by varying the RMS of the AC voltage. This is called an AC voltage controller. There are two types of control normally used:

- On-off control — TRIAC switches connect the load to the AC source for a few cycles and then disconnect it for another few cycles of the source voltage
- In phase control — TRIAC switches connect the load to the AC source for a moment in each cycle Figure 3

A reliable speed control of a ceiling fan AC motor can be accomplished by combining the MC9RS08KA2 and the phase angle control using a TRIAC. A benefit of this approach is avoiding non-linearity that is present if using only the TRIAC. Another benefit is, it can replace the mechanical speed variation commonly used in ceiling fans.

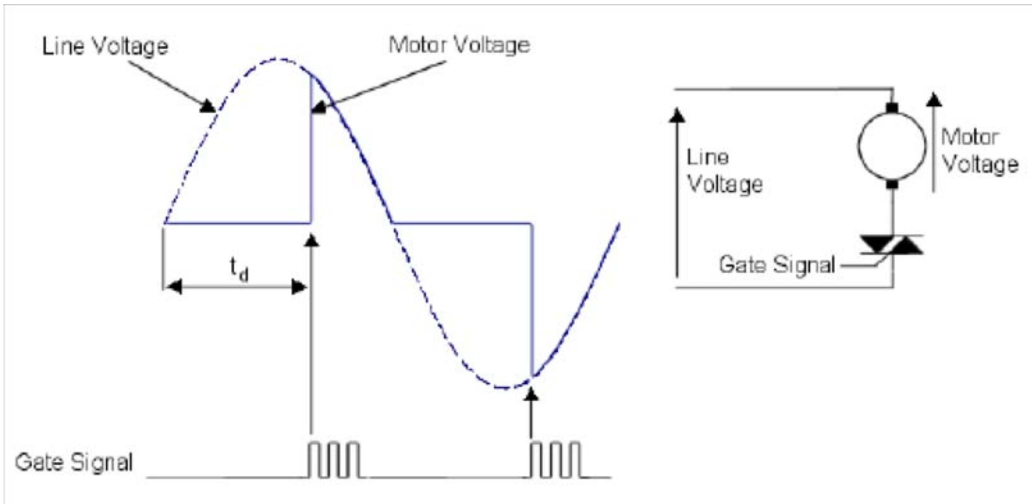

Figure 3. Line Voltage vs. Motor Voltage

3 Design Requirements

- DEMO9RS08KA2 board and a computer running with CodeWarrior
- A ceiling fan motor
- Components Section Appendix A, “Bill Of Materials (BOM)”

4 Instructions

4.1 Board Configuration

Steps for configuring the board:

1. Pull out every jumper in j101 except for RESET and LED 0
2. Connect the j101 SW0 pin on the push button side of PTA5 on j102. The push button SW0 connects to PTA5
3. Make sure the board is in host mode, j202 in the USB, and j203 V_{DD} enabled
4. Set jumpers on j101 to RESET and LED

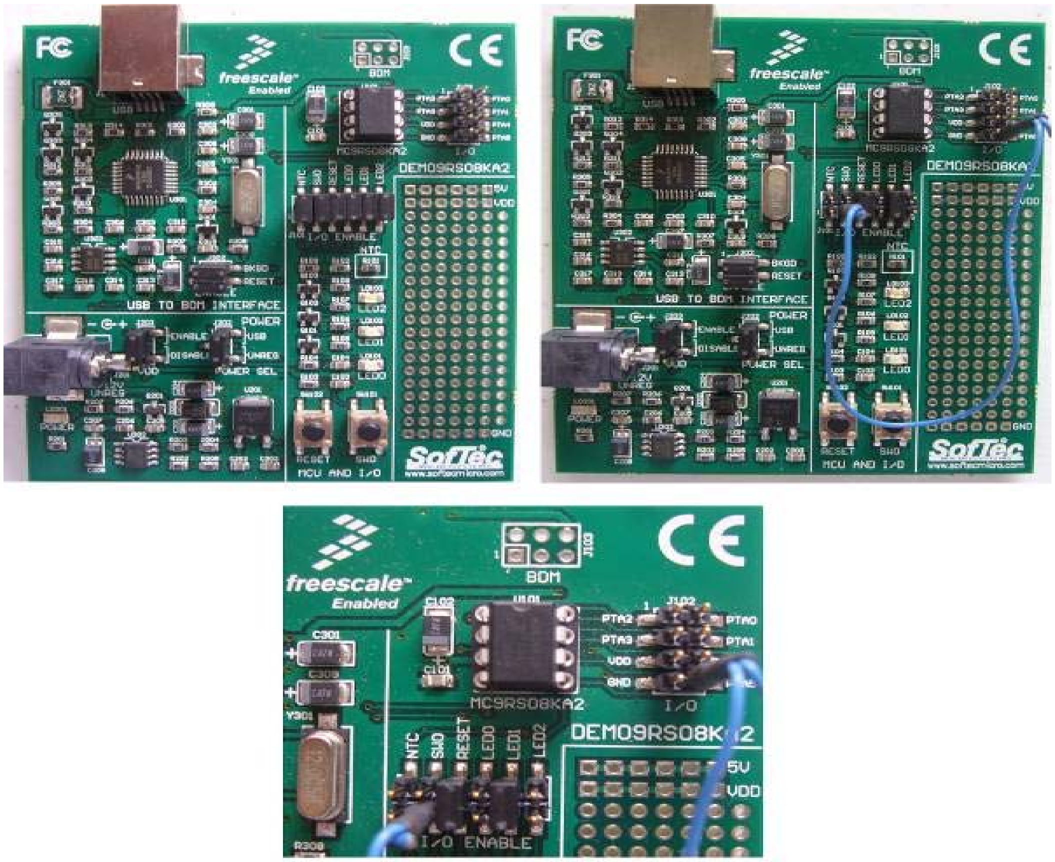

Figure 4. Board Configuration

In Figure 4, the upper left image shows the board. The following image to the right shows the board with the proper connections and the last image shows a close-up of the jumper configuration and connections.

4.2 Rectifier and Transformer

1. Identify the primary and secondary windings of the transformer (127 V – 60 Hz/ 6 V – 500 mA).
2. Connect common to GND. The GND must be common on the board
3. Connect the two cables of the secondary winding to the AC input on the bridge. Consult the transformer data sheet to identify the cables.
4. Connect the positive side of the bridge to GND with a 10 KΩ resistor.
5. Connect the negative side of the bridge to GND.
6. Check with an oscilloscope the voltage on the 10 KΩ resistor. It must show the full wave rectified.

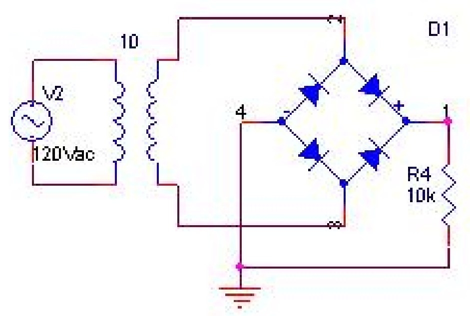

Figure 5. Rectifier and Transformer

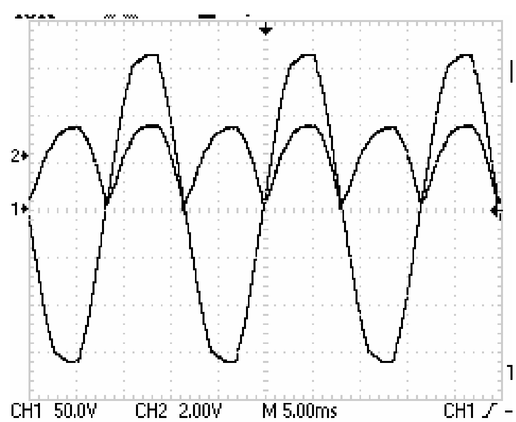

Figure 6. AC Wave vs. Rectified Wave

4.3 Connection from Rectifier to Board

1. Connect the positive bridge output to j102 PTA1. This the negative input of the controller comparator.
2. Make sure j102 GND is connected to the line GND.
3. Connect j102 PTA0 to the voltage on the output divider.

If j102 PTA0 is connected directly to 0 volts, the 0 voltage is not always reached. To ensure this detection the voltage on j102 PTA0 is near 0 (0.1 volts must work).

4.4 Pulse Width Modulation (PWM)

A pulse width modulation (PWM) signal is generated by using the timer and the comparator. The controller performs the following tasks:

- The controller is constantly checking for zero crossings through the comparator
- If this condition is true the output pin is cleared and the timer is started
- The timer counts up to a certain value. If it reaches a predefined value (MODULO), it is stopped and reset for the next cycle. The output pin is also set.
- The cycle is repeated by waiting for the next zero crossing.

The timer must not count more than the time it takes to detect a new zero cross. Calculate the correct value for the MODULO. For example, the line is 120 V, 60 Hz (1 cycle: 1/60 =.016 s). The rectified wave is 120 Hz MODULO and has an 8-bit value. Each half-wave is .008 s (.008 s/ 255 bit = 31.3 µs/bit). The duty cycle = t/T. The calculated values are:

— MODULO = 128 (half). Timer counts 128*31.3 µs = 4.006 ms. The output is 0 when it detects the zero crossing. The timer counts up to 128 and the output value is 1 until the next zero crossing occurs. The duty cycle is 50%.

— MODULO = 64 (quarter). Timer count 64*31.3 µs = 2.003 ms. The output is 0 when it detects the zero crossing. The timer counts to 64 and the output value is 1 until the next zero crossing. The duty cycle of 75%.

4.5 Checking the PWM

1. Open the project DEMO.mcp in the demo folder. If CodeWarrior is not installed refer to the starter kit user´s manual.
2. Connect the board to the computer. Use the USB cable.
3. Click on the green arrow with the bug, it enters debug mode. This downloads the program. See Figure 7.
4. The true time simulator real time debugger is opened and requests the MCU configuration. Choose DEMO9RS08KA2. See Figure 8.
5. Click on the green arrow to run the program. See Figure 9.
6. Connect the oscilloscope to see the signal in j102 PTA4. This is the PWM output.
7. On the windows for data variables find the duty cycle. See Figure 10.
8. Double-click on the number to select it and change it. This is an 8-bit number and only varies from 0 to 255.
9. Watch the signals on the oscilloscope by varying the duty_cycle variable.

Figure 7. CodeWarrior Menu

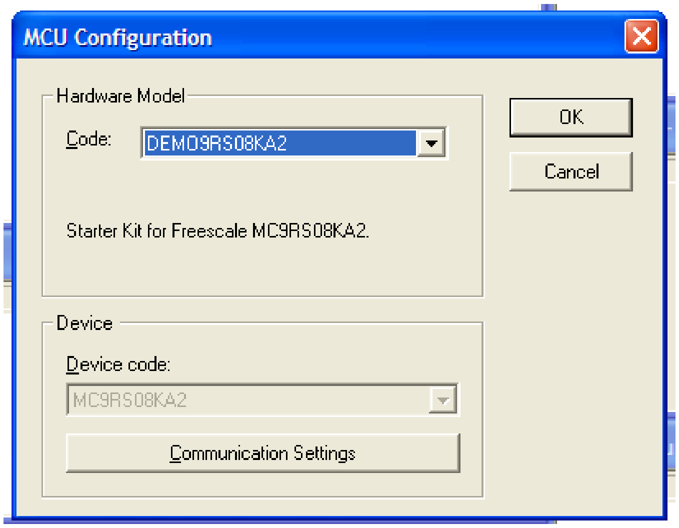

Figure 8. True Time Simulator – Real Time Debugger Configuration Window

Figure 9. True Time Simulator Menu

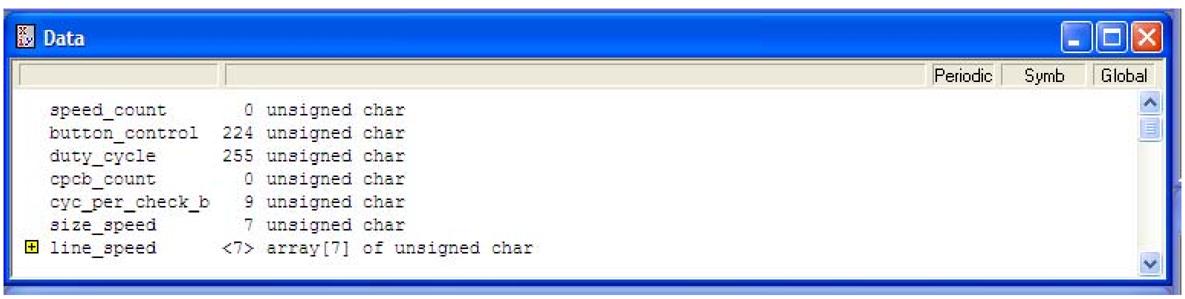

Figure 10. True Time Simulator Data Window

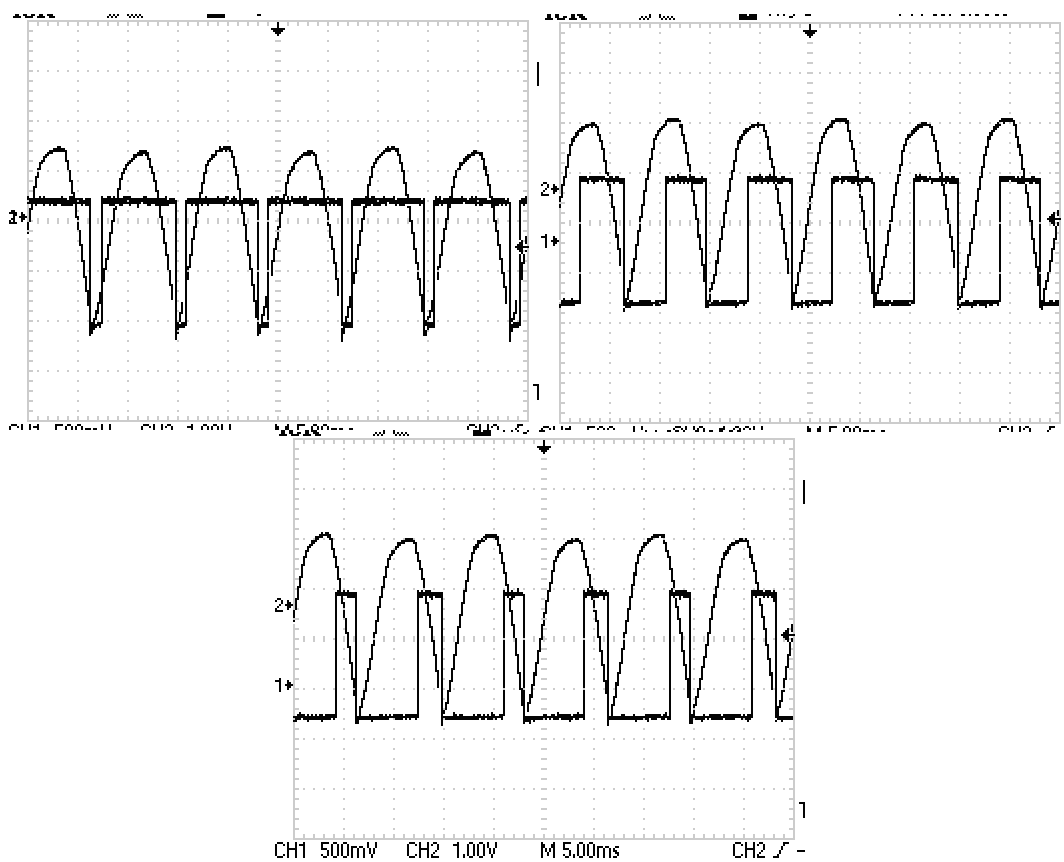

Figure 11. Rectified Wave and PWM at 87%, 50%, and 25%

4.6 Optocouplers (MOC)

Optocouplers (MOCs) are used to transmit signals between circuits that do not share a power source. MOCs have a LED and a sensor inside. If the led is turned on, it activates the sensor and lets the current flow.

This circuit is used to isolate signal circuitry from transients generated or transmitted by power supply and high-current control circuits.

1. Connect j102 PTA4 (PWM output) to pin1 of the MOC
2. Connect pin 2 to GND
3. Connect pin 4 to the TRIAC gate

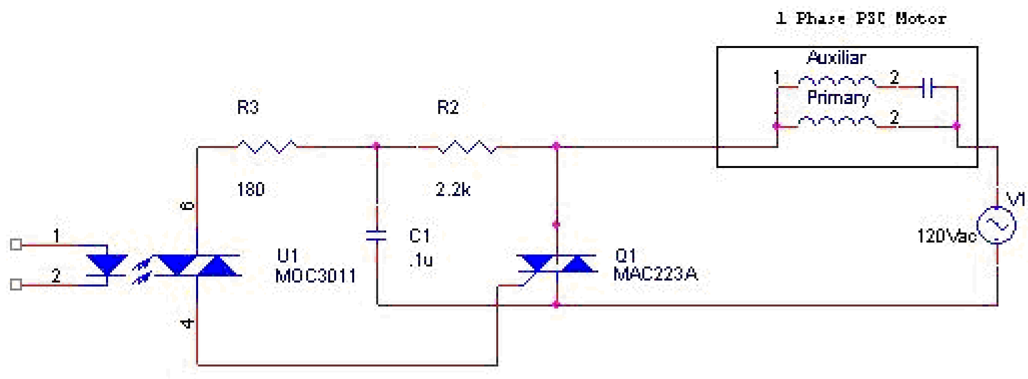

Figure 12. Schematic of the Power Circuit

4.7 TRIAC and SNUBBER

The TRIAC is an electronic bi-directional switch. If there is voltage on the gate, it transmits over its terminals until the current through it drops below a certain threshold value. A snubber network is used to assist the turn off and prevent premature triggering. In this circuit the combination of resistors and capacitors are used to suppress the rapid rise and fall of the voltage.

1. Connect to pin 6 of the MOC a resistance of 180 Ω in series with 2 KΩ
2. Between the two resistances connect a capacitor of .2 uF to the reference.
3. Connect any of the terminals of the TRIAC in series with the 2 KΩ
4. Connect the other TRIAC terminal to the reference.
5. Connect pin 4 of the MOC to the gate of the TRIAC.
6. Connect the motor one cable where the 2 KΩ and the TRIAC are connected and the other to the reference Figure 13.
7. Check the output of the TRIAC and compare it with the AC voltage. See Figure 14.

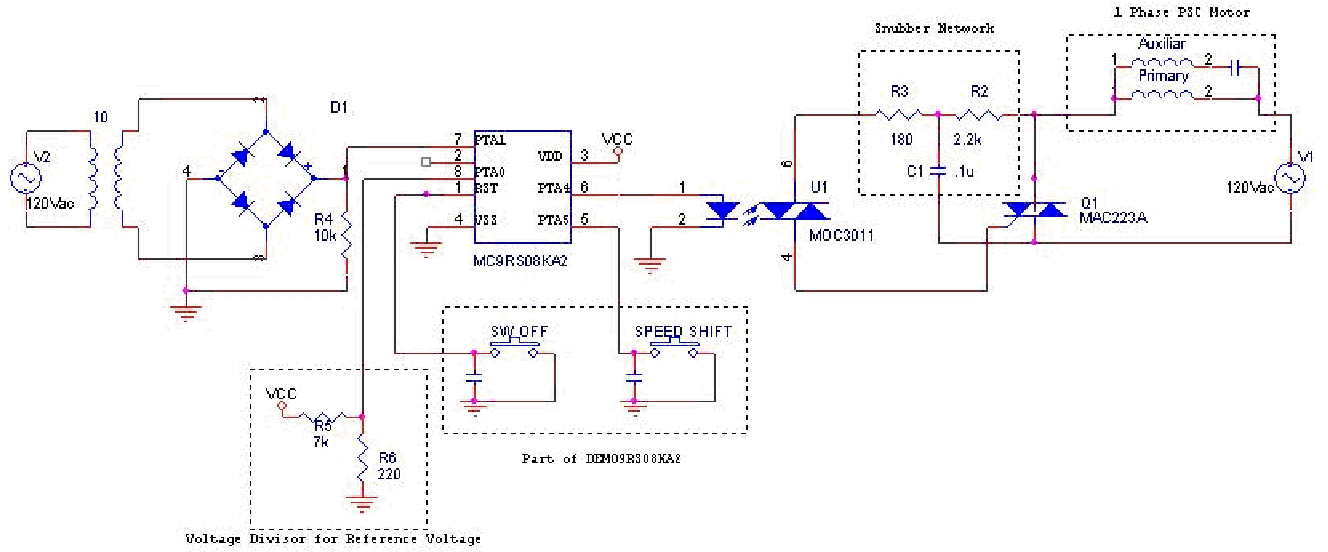

Figure 13. Complete Circuit

The PWM output signal starts with 0, after a certain time it triggers the TRIAC and conducts until AC reaches 0 again. Starting with one on the cross detection, the motor always runs at a certain speed. To see what the gate and terminals are consult the motor documentation.

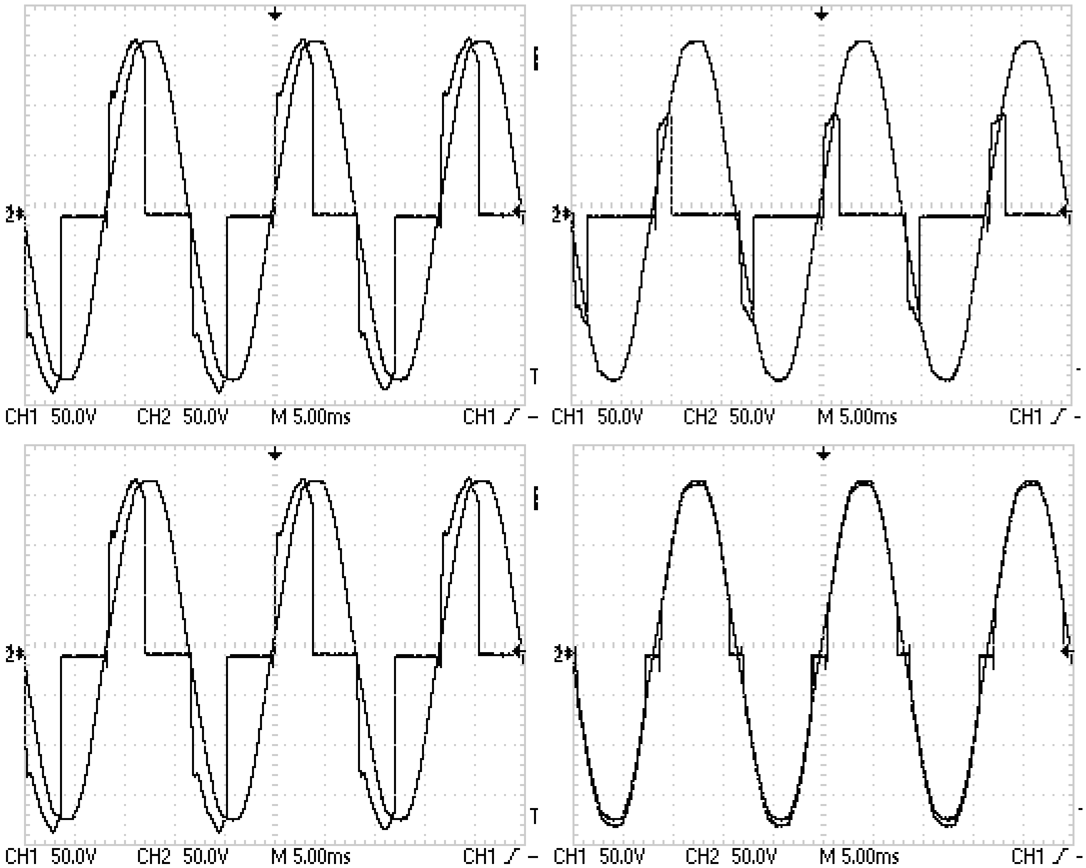

Figure 14. AC Voltage vs. TRIAC Output at Different PWM Values

5 Code

5.1 Description

The code can be divided into three modules:

- Init — This section of code is in charge of configuring the controller for this task. It disables the COP, configures FLL as clock source, configures the timer (8 Mhz input, and prescaler 256), configures the comparator (external reference, falling edge), and the GPIO's (PTA4 as output, PTA5 as input).

  The timer is configured to fit the frequency of 120 Hz of the rectified wave. Equation 1 and Equation 2.

$$(8Mhz)/(prescaler(256)) = 31.250Khz \qquad \textbf{Eqn. 1}$$

$$(31.250Khz)/(256counts) = 122Hz \qquad \textbf{Eqn. 2}$$

- Main Flow — This the infinite loop of the program. It waits for the zero crossing detected with the comparator. The timer then starts. When the timer overflows it activates the output PTA4 and the TRIAC gate. On the next zero crossing detection, it turns off the TRIAC gate and the cycle is repeated. It is also calls the check button function.
- Check button — This in charge of validating the button pushes. When a push is validated it changes the duty cycle of the PWM. This is the value to load in MODULO. Special care must be taken of the button. The code checks if the pin is low every cycle. If it is, the button is then pressed. In the next cycle the controller checks again if the pin remains low. The pin level is checked each cycle, 120 times each second.

There are two options for the button:

- When pressed, `cpcb_count` counts how many cycles the button remains pressed. If `cpcb_count` equals `cyc_per_check_b` the speed is changed.
- When pressed, each change of speed checks the low pin, considers the button pressed, and waits until the pin is high to check it again.

5.2 Flow Diagram

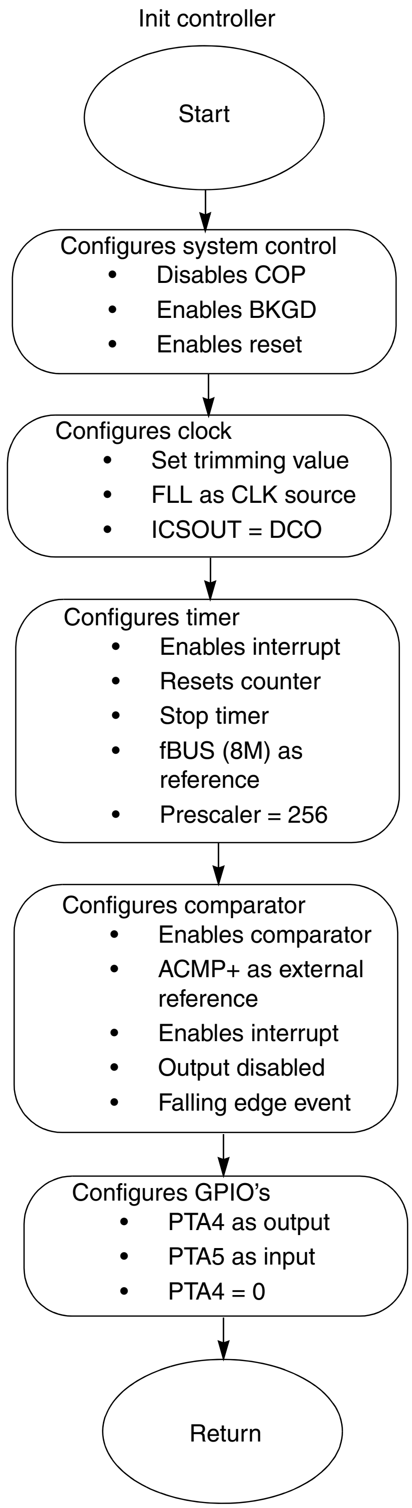

Figure 15. Init Flow Chart

Main flow:

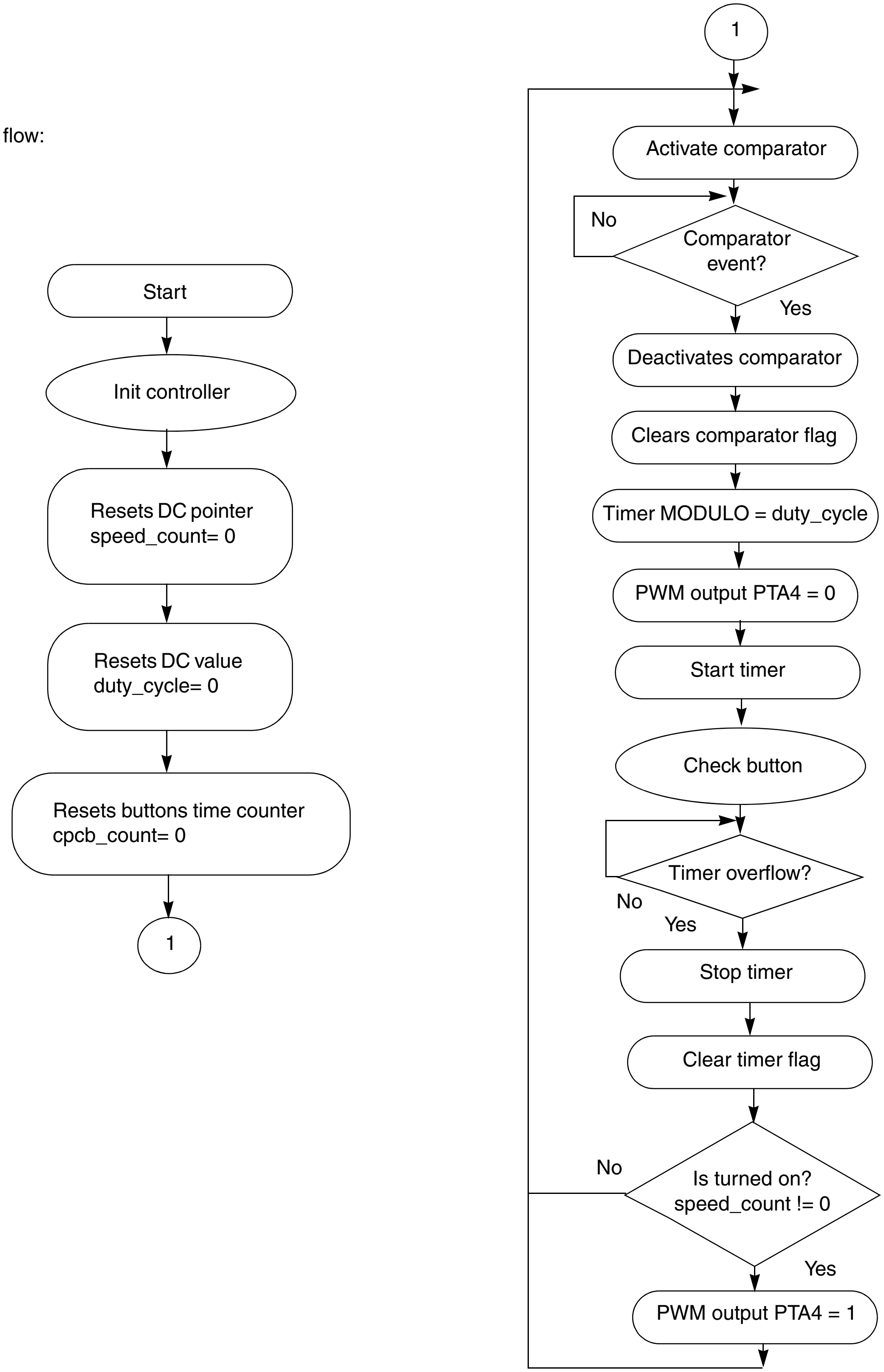

Figure 16. Main Program

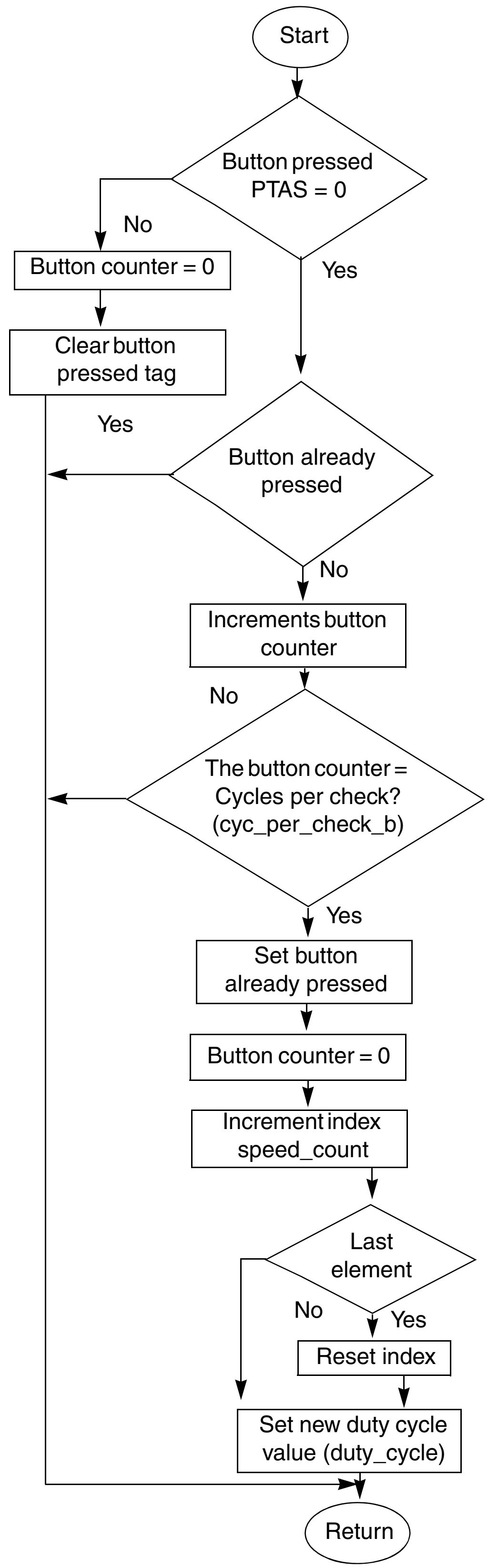

Figure 17. Push Button Flow Chart

6 Testing and Validation

The entire circuit can be divided in three modules and can be checked independently.

- The first module is the transformer and rectifier. At the end of this stage there is a rectified AC wave at 120 Hz at half of the original voltage peak-peak.
- The next module is the zero cross detection circuit. To test this module the previous is needed and the MCU with the program running. The designated output pin of the MCU is expected to have a pulse at 120 Hz. If this signal and the previous stage signal outputs are viewed at the same time, it generates a pulse each time the rectified sine wave reaches zero.
- The last module has the MOC and TRIAC. To test if this part of the circuit is working properly, disconnect the side of the MOC connected to the MCU and replace it with 3.3 VDC. The motor has to start working when on and if off the motor has to turn off. When the motor switches on/off it makes a sound, to be sure wait a lapse of time while the motor is off. It must come to a complete stop.

7 Conclusion

Using an MC9RS08KA2 microcontroller combined with the TRIAC topology, a reliable solution is reached for varying the speed of a ceiling fan. This solution is viable for replacing the existing commonly used mechanism.

8 References

See the Freescale web page www.freescale.com.

- *DRM039 — Single Phase AC Induction Motor Control Designer Reference Manual*
- *MC9RS08KA2 MC9RS08KA1 Data Sheet*
- *RS08 Core Reference Manual*

Appendix A Bill Of Materials (BOM)

- Transformer, Input: 127 Vac, Output: 6 V 500 mA
- Rectifier bridge 50 V, 2 A
- 10 K resistor
- 7 KΩ resistor
- 220 Ω resistor
- MOC 3011
- 0.1 uF capacitor
- 180 Ω resistor
- 2.2 KΩ resistor
- MAC223A TRIAC

Appendix B Source Code

```
;*********************************************************************************
;       MAIN.ASM
;*********************************************************************************;
; Single Phase Demo for the DEMO9RS08KA2
; -------------------------------------------------------------------------------
; This example controls a single phase motor through a PWM.
; The PWM is generated with the comparator (to detect zero crossing) and the timer (to; generate
duty cycle
;
; ACMP - is fed with a value approximate to 0
; ACMP + rectified wave
; PTA4 PWM output to feed the MOC, also connected to a LED to show the speed
; through LED's;intensity
; PTA5 push button to vary the speed
; RESET/PTA2 turns off the motor
;
;
; ********* If you want the button to run free the comment values, button to comment
; ********* set cyc_per_check_b to the value wanted
; ********* notice that cyc_per_check_b also avoids bouncing.
; ********* understand RUN FREE as the button is checked every
; ********* cyc_per_check_b * 8.4 ms (120 ;hz) that is, if kept pressed it
; ********* changes duty or press and release the button in
; ********* order to change duty values
; ***************************************************************************

; export symbols
             XDEF _Startup, main
            ; export both '_Startup' and 'main' as symbols. Either can
            ; be referenced in the linker .prm file or from C/C++ later on

; Include derivative-specific definitions
            INCLUDE 'derivative.inc'

D_X             equ   $0000000E
X_              equ   $0000000F
CLKST           EQU   2
PTA4            EQU   4
PTA5            EQU   5
ACF             EQU   5
ACME            EQU   7
TOF             EQU   7
TSTP            EQU   4
BUTTON_PRESSED  EQU   0                ; line for button pressed
                                       ; Variable/Data Section
MY_ZEROPAGE: SECTION  SHORT
speed_count:    DS.B  1                ; this is a pointer to select
                                       ; which value is used in line_speed array
button_control  DS.B  1                ; flag for button
duty_cycle:     DS.B  1                ; this value is how long (timer) it
                                       ; stays up/down
cpcb_count    DS.B 1          ; cycles to check button counter
                                       ; number of cycles it waits to check the
                                  ; button again
; Const Section
```

```
ConstSection:  SECTION
cyc_per_check_b DC.B  $09              ; number of counts (wave cycles) it takes to
                                       ; read button
size_speed      DC.B  $07              ; number of elements in line speed array
line_speed      DC.B  $01,$20,$40,$80,$C0,$E0,$F0    ; if you change the size of the
                                       ; linespeed do not forget to modify Project.map
;line_speed     DC.B  $F0,$E0,$C0,$80,$40,$20,$01    ; if you change the size of the
                                       ;linespeed do not forget to modify Project.map
; Code Section
MyCode:     SECTION
;**************************************************************************************
; Peripheral Initialization
;**************************************************************************************
init:

;CONFIGURES SYSTEM CONTROL
  MODE:         EQU 0                  ; MODE=0 Background Mode, MODE=1 Run Mode
  IFNE  MODE
    mov #HIGH_6_13(SOPT), PAGESEL
    mov #$01, MAP_ADDR_6(SOPT)         ; Disables COP and enables RESET (PTA2) pin
  ELSE
    mov #HIGH_6_13(SOPT), PAGESEL
    mov #$03, MAP_ADDR_6(SOPT)         ; Disables COP and enables BKGD (PTA3) and
                                       ; RESET (PTA2) pins
  ENDIF

;CONFIGURES CLOCK (FEI Operation Mode)
    mov #HIGH_6_13(NV_ICSTRM),PAGESEL
    lda MAP_ADDR_6(NV_ICSTRM
       sta ICSTRM                      ; Sets trimming value
          clr ICSC1                    ; Selects FLL as clock source and disables it
                                       ; in stop mode
          clr ICSC2                    ; ICSOUT = DCO output frequency
wait_clock:
    brset CLKST,ICSSC,wait_clock       ; Waits until FLL is engaged

;CONFIGURES TIMER
    mov #$70, MTIMSC                   ; Enables interrupt, stops and resets timer counter
    mov #$08, MTIMCLK                  ; Selects fBUS as reference clock (8 MHz)
                                       ; prescaler = 256 (increments timer counter every
                                       ; 32 us)

;CONFIGURES ACMP
    mov #$F0, ACMPSC                   ; Comparator enabled, ACMP+ as external reference
                                       ; clear flag and enable interrupt, output disabled
                                       ; Comparator falling edge
;CONFIGURES PORT
    bset PTA4, PTADD                   ; set port A4  as output
    bclr PTA5,PTADD                    ; sets port A5 as input
    bclr PTA4, PTAD                    ; clears port PTAD4

    rts

;**************************************************************************************
; Entry Point
;**************************************************************************************
```

```
_Startup:

main:
    bsr init
    mov #0, speed_count               ; this is a pointer for the main speed wanted, first
                                      ; set it to 0
    mov #0, duty_cycle                ; this is a parameter for the timer, indicating when
                                      ; it stops counting
    mov #0, cpcb_count                ; reset cycles count for button check
tag1:
    bset ACME,ACMPSC                  ; activates analog comparator
loop1:
    ;wait                             ; sends controller to wait state
    brclr ACF,ACMPSC,loop1 ; waits until there is a comparator event
    bclr ACME,ACMPSC                  ; deactivates comparator
    bset ACF,ACMPSC                   ; clears comparator flag
    mov duty_cycle,MTIMMOD            ; sets Modulo to maximum count, also resets
                                      ; counter and clears timer flag
    bclr PTA4, PTAD
    bclr TSTP,MTIMSC                  ; starts timer
    bsr check_button
check_timer:
    ;wait                             ; sends controller to wait state
    brclr 7,MTIMSC,check_timer        ; waits until there is a timer overflow
    bset TSTP,MTIMSC                  ; stops timer
    bclr TOF,MTIMSC                   ; clears timer overflow flag
    lda #$00                          ; turns off completely when it points to
                                      ; the minimum speed
    cmp speed_count
    beq  end_main
    bset  PTA4, PTAD
end_main:
    jmp tag1

check_button:
    brclr PTA5,PTAD,chb_cont          ; if the button is pressed, it goes to check_button
    mov #$00,cpcb_count               ; if button is released, it resets the counter to
    bclr BUTTON_PRESSED,button_control ; bit 0 as flag for button already pressed
    jmp button_return                 ;
chb_cont:
 ; check counter to wait to check button
    brset BUTTON_PRESSED,button_control,button_return ; BUTTON TO COMMENT
                                      ;if the button has not been released, it returns
                                      ; from subroutine
    inc cpcb_count                    ; increments counter for push button
    lda cpcb_count                    ; compares cpcb_count value with defined
                                      ; constant value
    mov #HIGH_6_13(cyc_per_check_b), PAGESEL
    cmp MAP_ADDR_6(cyc_per_check_b)
    bhs chb_cont2                     ; if the value is reached it jumps chb_count2, or
                                      ; returns and continues counting
    jmp button_return
chb_cont2:
    bset BUTTON_PRESSED,button_control ; sets it to mark the button as already pressed
    mov #$00,cpcb_count               ; resets counter because the button has been
                                      ; pressed long enough
    inc speed_count                   ; increments pointer, (this value is used
```

```
                                       ; with X)
    lda speed_count                    ; compares pointer value with array size
    mov #HIGH_6_13(size_speed), PAGESEL
    cmp MAP_ADDR_6(size_speed)
    blo set_duty
    mov #0,speed_count                 ; if pointer > array size then resets 0
set_duty:
    mov #HIGH_6_13(line_speed), PAGESEL;load duty value into duty_cycle variable
    lda #MAP_ADDR_6(line_speed)
    add speed_count
    sta X_
    mov D_X,duty_cycle
button_return:
    rts
```

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor

Document Number: AN3471
Rev. 0
07/2008