1 CHANNEL DAC USING A PWM

INTRODUCTION

Many MCU applications require a digital-to-analog conversion (DAC). Though many of these applications require a complex DAC, some may only require a simple 1 channel DAC. Because most MCUs do not have a DAC built-in, and an external DAC would be too expensive, an alternative method for the 1 channel DAC is used. Such a method is the generation of a Pulse Width Modulator (PWM), which is afterwards integrated by a low pass filter. This opportunity provides the 1 channel DAC with a resistor and capacitor. The additional advantage is that the DAC resolution can be configured by software. However, the basic principle around such a DAC is the generation of the PWM.

PWM GENERATION

There are several ways of generating a PWM. The simplest form is generating the PWM straight from the dedicated PWM-timer on the MCU. Unfortunately, not all MCUs have a PWM-timer built in. As an alternative solution, one could also use 2 timers (one for the frequency and one for the duty cycle for this task) to achieve high frequency PWMs; however, a high frequency PWM is not always necessary allowing the user to save one of the two timers. This application note describes low frequency PWM generation with just one general purpose timer.

MCU INITIALIZATION

Shortly after the definition and interrupt table is initiated, the software initializes the MCU. It then enters the main routine, which is currently an endless loop which consists out of NOPs. The programmer can insert instructions here. The PWM generator itself is completely interrupt-driven and is found below the label *timer1:*. After saving the *Register Pointer* and swapping data from of the working register group, the PWM-generator determines if it is currently in the low or high cycle of the period. If it is in the low cycle, then it jumps to the label *turn_on:*, which initiates the high cycle. Conversely, if the PWM is in the high cycle, it jumps to the label *turn_off:*, which initiates the low cycle. Ultimately, every time the timer expires it initiates the next cycle.

At *turn_on:* (initiation of the high cycle), the port pin is pulled high and Timer1 is loaded with the value for the *duration_of_high_cycle*. At *turn_off:* (initiation of low cycle), the port pin is pulled low and Timer1 is loaded with the value for the *duration_of_low_cycle*.

Next, the interrupt routine determines if the *integration_timer* expired. The *integration_timer* is a software counter that performs the routine checks on the keys allowing the increasing or decreasing of the duty cycle(= the high cycle of the PWM). From that point, the routine computes the correct values for the low cycle (duration_of_low_cycle) and the high cycle (duration_of_high_cycle) to keep the right period time or PWM frequency.

At the end of the interrupt routine, the original value of the register pointer is restored and the program counter is diverted back to the main program.

Note: Timer1 runs continuously and loads on the fly.

CONSTANTS DEFINITION

There is a portion of the source code called the *Constants Definition*. Here the values of some constants can be changed and the software behavior can be altered. The number of PWM levels (constant = pwm_levels) provides the resolution of the PWM. The predefined value is 63, which provides a resolution of 6 bits. Thus, 2 by the power of the resolution in bits with 1 subtracted, provides the *pwm_levels*. The constant *max_high_cycle* limits the duration of the PWM duty cycle. The *Max_high_cycle* should be at least one less than the *pwm_levels*.

SOFT START OPTION

Additionally, there is a soft start option available. Conditional assembly can activate it. After the reset, this routine slowly raises the duty cycle. The constant *softstart_time* defines a time for the slow increase of the duty cycle. When the duty cycle becomes long enough, the *softstart* terminates. This situation occurs when the variable *duration_of_high_cycle* = *start_up_level*.

The following flow chart (Figure 1) indicates the program sequence. Figure 2 illustrates the Interrupt Routine.

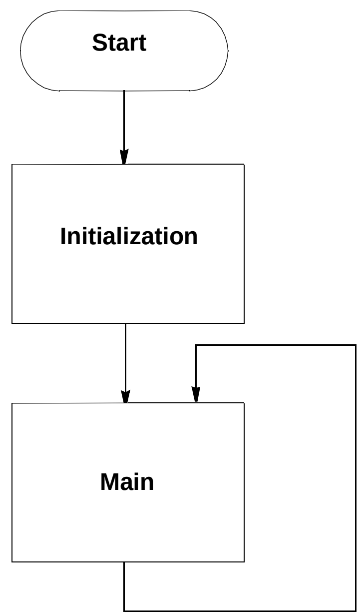

Figure 1. Program Sequence

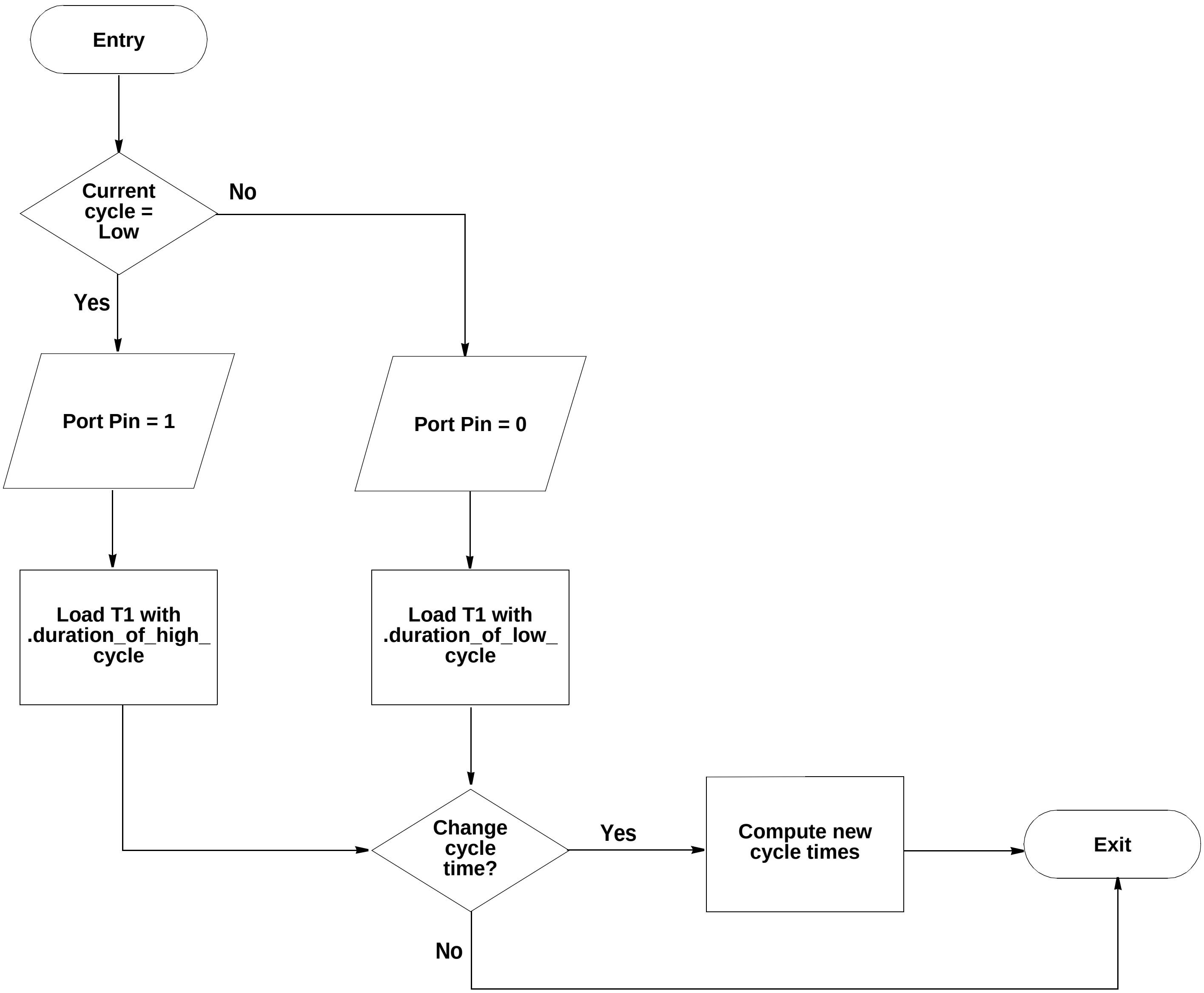

Figure 2. Interrupt Routine

The assembler source code itself is stored in the file *pwm_gen1.s*. With the software now in place, the next step is to examine the external hardware.The user must integrate the PWM to obtain a straight voltage with the required level to make it *flat*. The easiest way to perform this action is by adding a low-pass RC filter (see Figure 3).

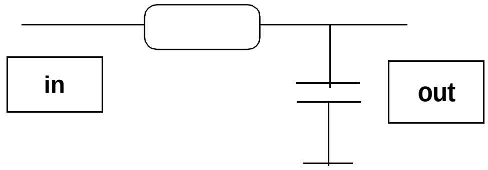

Figure 3. RC Filter Configuration

SOFT START OPTION (Continued)

The values for R and C can be computed with the following formula

$$f_g = \frac{1}{2\pi R_{filter} C_{filter}}$$

A standard design rule is to make

$$f_g = \frac{1}{10} * f_{pwm}$$
$$\tau_{filter} = \frac{10}{2\pi * f_{pwm}}$$

This way the user can get the RC time constant:

A standard design rule is to make

$$R_{filter} = \frac{R_{load}}{10}$$

By incorporating this guideline, the user will not lose too much power in the filter resistor. Not following the formula could also distort our calculation, causing the load resistance to be too low in comparison to the filter resistor. The load resistor also influences the filter frequency. A factor of 1/10 makes the error negligible.

$$C_{filter} = \frac{\tau_{filter}}{R_{filter}}$$

If the remaining voltage ripple is still too high, the next choice is to put 2 RC filters in series (see Figure 4).

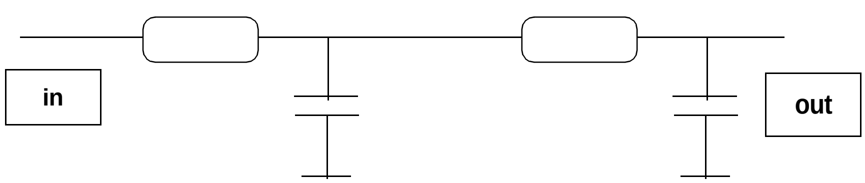

Figure 4. RC Configuration Using 2 Filters

The filter frequency (assuming both filters have the same R and C) would now be represented as:

$$C_{filter} = \frac{\tau_{filter}}{R_{filter}}$$

$$f_g = \frac{1}{10} * f_{PWM}$$

The resulting calculation would yield the following:

$$\tau_{filter} = \frac{10}{\sqrt{2} * \pi * f_{PWM}}$$

$$R_{filter} = \frac{R_{load}}{20} \qquad C_{filter} = \frac{\tau_{filter}}{R_{filter}}$$

In many cases, it is also preferable to drive some kind of induction, such as:

- motor
- magnet
- valve
- relay

Since an induction always consists out of an L and an R, a low pass filter to straighten the PWM is not always desired. This combination of L and R straightens the current (instead of the low pass filter straightening the voltage). An example of this induction using a series resistor it illustrated in Figure 5.

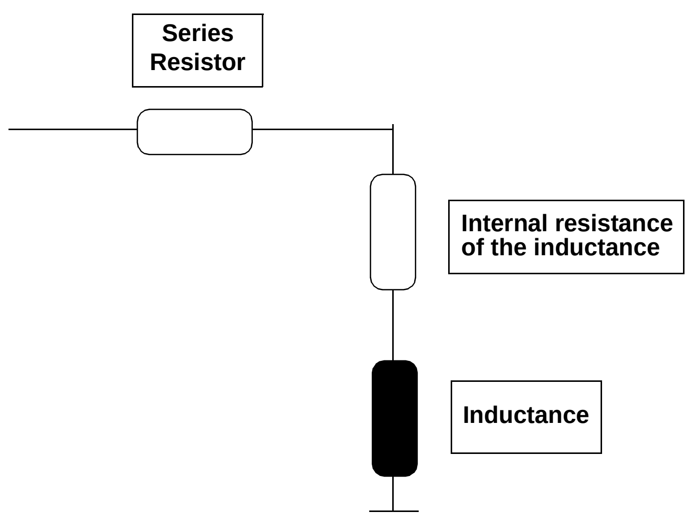

Figure 5. Induction Example

The design rule here is that it may be necessary to obtain the value for the imaginary resistance of the inductance. The easiest way is to measure the resistance of the inductance at the PWM frequency (voltage and current) is by incorporating the following formula:

$$X_{inductance} = 2\pi * f_{PWM} * L_{inductance}$$

SAMPLE CODE (Continued)

If L is specified, then the user can calculate X; however, be careful the value for L at the PWM frequency is used. An alternate way to get X could be calculated as follows:

$$R = \frac{1}{10} * X_{inductance}$$

The ohmic resistance of the inductance, on the other hand, is mostly specified. If not, the user can simply measure it with a ohm-meter. A series resistor is required, however, before the inductance to straighten the current. The formula reads as follows

$$R_{series} = R - R_{inductance}$$

A feedback diode is necessary to obtain a straight current and to protect the port output transistors. When the software and hardware are determined the target board can be built.

Figure 6 illustrates the schematic for the 1 Channel PWM.

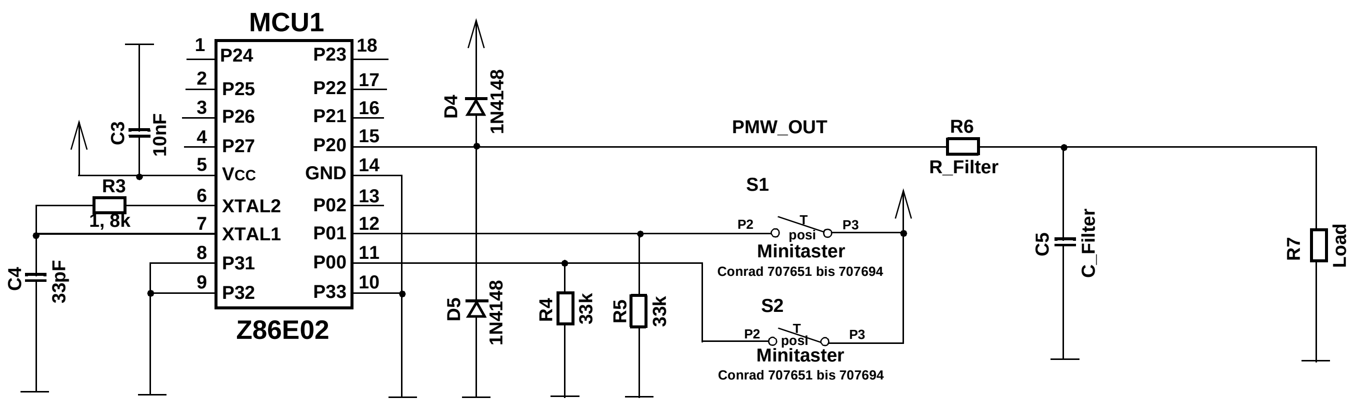

Figure 6. 1 Channel PWM Schematic

SAMPLE CODE

Pages 7–13 illustrate the step-by-step process of allowing a PWM to generate a DAC.

CONCLUSION

By following these suggestions, a user can easily implement DAC functionality through minor adjustments in the PWM. Though this application note does not cover complex digital-to-analog conversions, it allows a user to supplement a 1 channel DAC for use in simpler applications.

```
;********************************************************************
;       This application note is for
;       PWM generation.
;
;       FILE:           pwm_gen1.S
;       DATE:           26.02.97
;       MCU:            Z86E02
;       PROJECT:        PWM generation techniques
;       AUTHOR:         Klaus Buchenberg
;       SOFTWARE:       REVISION 1.0
;
;       Oscillator = 8MHz
;       This program is assembled by ZiLOG ZMASM assembler
;********************************************************************

      GLOBALS ON

;********************************************************************
;       When the softstart option is wished, then set
;       SOFTSTART_WISHED to 1 = Yes
;********************************************************************
SOFTSTART_WISHED .equ    1       ; 1 = Yes     0 = No

;********************************************************************
;       Bitnumber definitions
;********************************************************************
bitno0          .equ    %01
bitno1          .equ    %02
bitno2          .equ    %04
bitno3          .equ    %08
bitno4          .equ    %10
bitno5          .equ    %20
bitno6          .equ    %40
bitno7          .equ    %80

;********************************************************************
;               PORTS DEFINITION
;********************************************************************

;       Port 0 pin
;       P00 : power. UP key
upkey          .equ    bitno0
;       P01 : power. DOWN key
downkey        .equ    bitno1
;       P02 : idle input
```

```
;       Port 2 pin
;       P20 : output for pwm
pwm_output     .equ    bitno0
;       P21-P27 : output idle N.C.


;       Port 3 pin
;       P31 : output voltage sensing, pull to ground when not used
;Usense         .equ    bitno1
;       P32 : idle input, pull to ground when not used
;       P33 : voltage reference, pull to ground when not used
;Uref           .equ    bitno3

********************************************************************
;               REGISTERS DEFINITION
;*******************************************************************
integration_timer       .equ    r4      ; counts the number of periods
; softstart_counter     .set    r5      ; counter for softstart routine
duration_of_low_cycle   .equ    r6      ; length of low time of period
duration_of_high_cycle  .equ    r7      ; length of high time of period
; delay_counter         .set    %FE     ; counter for delay routine

;*******************************************************************
;               BITS DEFINITION
;*******************************************************************



;*******************************************************************
;               CONSTANTS DEFINITION
;*******************************************************************
pre1_min                .equ    01111011b       ; PRE1=30, continuous mode, int. clock
                                                ; pwm freq. =
                                                ; osc. freq./(PRE1 *
8*no_voltage_levels)
                                                ; in this case pwm freq. = 521Hz
                                                ; pwm freq. = 8E6Hz/(30*8*64)
pwm_levels              .equ    63              ; number of pwm levels
                                                ; pwm resolution = log(base2)x
                                                ; pwm resolution = log(base2)64 = 6
start_up_level          .equ    20              ; start level <= pwm_levels
max_high_cycle          .equ    62              ; max duration of high cycle
softstart_time          .equ    4               ; extends start up time by
                                                ; x * 10msec.

;*******************************************************************
;               MACROS
;       Refer to Z8 technical manual for macro definition
;*******************************************************************
bset            MACRO   register,bitnumber              ; set the appropriate bit in
            or              \register,#\bitnumber           ; the specified register
```

```
            MACEND

bclr            MACRO   register,bitnumber              ; clear the appropriate bit in
            and             \register,# ~(\bitnumber)       ; the specified register
            MACEND

brset           MACRO   register,bitnumber,label                ; IF  the appropriate bit
in
            tcm             \register,#\bitnumber           ; the specified register is
set
            jr              z,\label                                ; THEN jump to label
            MACEND                                  ; ELSE go on

brclr           MACRO   register,bitnumber,label                ; IF the appropriate bit
in the
            tm              \register,#\bitnumber           ; specified register is reset
            jr              z,\label                                ; THEN jump to label
            MACEND                                  ; ELSE go on

pwm_high_cycle MACRO
            bset    r2,pwm_output
            MACEND

pwm_low_cycle   MACRO
            bclr    r2,pwm_output
            MACEND

;*********************************************************************
;               INTERRUPTS VECTOR
;*********************************************************************

            .MLIST
            .LIST

;       Interrupt vector address  %00 to %0C

            .ORG    %0000
            .word   irq0
            .word   irq1
            .word   irq2
            .word   irq3
            .word   irq4
            .word   timer1

;************************************************
;               PROGRAM STARTS HERE             *
;************************************************

BEGINNING:
            .ORG    %0C
irq0:
```

```
irq1:
irq2:
irq3:
irq4:
            di
            ld      P01M,#00000101b        ;P0, P1 input, internal stack
            ld      P2M,#00000000b         ;P20-P27 output
            ld      P3M,#00000011b         ;P3 analog + P2 push pull
            and     P2,#11111110b          ;Switch off transistor

            clr     SPH
            ld      SPL,#%40                ; INIT STACK POINTER
            ld      IPR,#00001010b          ; IRQ5 has highest priority
            ld      IMR,#00100000b          ; enable T1 interrupt

; INITIALIZE RAM TO "0"
;           srp     #%30
;           ld      R14,#%3d
;zram:      clr     @R14
;           djnz    R14,zram



;       Initialize all registers

            srp     #%00                                   ; set working register to %00
            clr     integration_timer
            ld      pre1,#pre1_min                        ; preset T1
            ei
            ld      TMR,#00011100b
            IF      SOFTSTART_WISHED
            ld      duration_of_high_cycle,#1
            call    softstart                                    ; increase level slowly
            ELSE
            ld      duration_of_high_cycle,#start_up_level  ; preset pwm level
            ENDIF

;*******************************************************************
;///////////////////////////////////////////////////////////////////
;       MAIN USER PROGRAM
;       The pwm generator runs as a batch task via T1 interrupt.
;       This is the user program that runs in front.
;///////////////////////////////////////////////////////////////////
Main:
            NOP                             ; insert your instructions
                                            ; here
            jp Main

;*******************************************************************
;-----------------------------------------------------
;               SUBROUTINES
```

```
;-----------------------------------------------------

            IF      SOFTSTART_WISHED
;/////////////////////////////////////////////////////
;       1.5uS x  (30x222)  =  10 mS
;/////////////////////////////////////////////////////

delay_counter   .set    %FE
delay10msec:
            ld      delay_counter,#222      ; 6 cycles
loop1:
            nop                                             ; 6 cycles
            nop                                             ; 6 cycles
            dec     delay_counter           ; 6 cycles
            jr      nz,loop1                ; 12 cycles
            ret

            ENDIF

;/////////////////////////////////////////////////////////////////////////////////////////
///////////////////
;               Increases the pwm level slowly after the start.
;               (extends light bulb life or speeds up
;           the engine slow = less start momentum)
;/////////////////////////////////////////////////////////////////////////////////////////
///////////////////
            IF      SOFTSTART_WISHED
softstart_counter       set     r5

softstart:

            cp      duration_of_high_cycle,#start_up_level  ; Is the start up level
            jr      GE,end_softstart                        ; reached, already?
            ld      softstart_counter,#softstart_time
softstart_delay:
            call    delay10msec
            djnz    softstart_counter,softstart_delay
slow_down:
            inc     duration_of_high_cycle                  ; No, then increase
            ;ld     duration_of_low_cycle,#pwm_levels       ; the level slowly.
            ;sub    duration_of_low_cycle,duration_of_high_cycle
            jr      softstart                               ; Yes, then go to main.

end_softstart:
            ret
            ENDIF



;////////////////////////////////////////////////////////////////////////////
;       Timer1 interrupt occurs on each edge of the period.
```

```
;       The timing depends on the power level = high/low ratio.
;////////////////////////////////////////////////////////////////////
timer1:
            push    rp                                      ; working register
            srp     #%00                                            ; group 0 reserved
                                                ; for timer1
                                                ; interrupt

            brclr   r2,pwm_output,turn_on                   ; If last pwm-cycle was
                                                ; low then next pwm-
                                                ; cycle is high
turn_off:
            pwm_low_cycle
            ld      T1,duration_of_high_cycle                       ; On next T1
end_of_count,
                                                      ; low cycle is finished, and
                                                ; duration_of_high_cycle
                                                ; is loaded from T1 into
                                                ; 8bit-down-counter
            jr      end_of_interrupt
turn_on:
            pwm_high_cycle
            ld      T1,duration_of_low_cycle                        ; On next T1
end_of_count,
                                                    ; high cycle is finished,
                                                ; and duration_of_low_cycle
                                                ; is loaded from T1 into 8bit-
; down-counter

;/////////////////////////////////////////////////////////////////////////////////////////
///////////////////
;       If upkey is pressed = P00 low then increase
;       the duration of the pwm-high-cycle.
;       If downkey is pressed = P01 low then decrease
;       the duration of the pwm-high-cycle
;/////////////////////////////////////////////////////////////////////////////////////////
///////////////////
            djnz    integration_timer,end_of_interrupt
            brclr   r0,upkey,increase_level
            brclr   r0,downkey,decrease_level
            jr      end_of_interrupt
decrease_level:
            djnz    duration_of_high_cycle,cycle_adjust
increase_level:
            inc     duration_of_high_cycle
            cp      duration_of_high_cycle,#max_high_cycle
            jr      GT,decrease_level
cycle_adjust:
            ld      duration_of_low_cycle,#pwm_levels
            sub     duration_of_low_cycle,duration_of_high_cycle
```

```
end_of_interrupt:
            pop     rp
            iret

            END
```

Information Integrity:

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact your local ZiLOG Sales Office to obtain necessary license agreements.

ZiLOG, Inc.
910 East Hamilton Avenue, Suite 110
Campbell, CA 95008
Telephone (408) 558-8500
FAX 408 558-8300
Internet: http://www.zilog.com