Application Note

Using a PWM Timer as a Digital to Analogue Converter

Document No. U19096EE1V0AN00

Date published December 2007

Printed in Germany

Legal Notes

- **The information in this document is current as of November, 2007. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**

- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.

- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

- The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime

systems, safety equipment and medical equipment (not specifically designed for life support).
"Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)
(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives anddistributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and otherlegal issues may also vary from country to country.

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668, Japan
Tel: 044 4355111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554,
U.S.A.
Tel: 408 5886000
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211 65030
http://www.eu.necel.com/

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908 691133

Succursale Française
9, rue Paul Dautier, B.P. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01 30675800

Sucursal en España
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091 5042787

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 6387200

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02 667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven,
The Netherlands
Tel: 040 2654010

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27
ZhiChunLu Haidian District,
Beijing 100083, P.R.China
Tel: 010 82351155
http://www.cn.necel.com/

NEC Electronics Shanghai Ltd.
Room 2511-2512, Bank of China
Tower,
200 Yincheng Road Central,
Pudong New Area,
Shanghai 200120, P.R. China
Tel: 021 58885400
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
12/F., Cityplaza 4,
12 Taikoo Wan Road, Hong Kong
Tel: 2886 9318
http://www.hk.necel.com/

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R.O.C.
Tel: 02 27192377

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253 8311
http://www.sg.necel.com/

NEC Electronics Korea Ltd.
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku, Seoul,
135-080, Korea Tel: 02-558-3737
http://www.kr.necel.com/

Table of Contents

Chapter 1 Introduction

Many embedded microcontroller applications need to generate DC and AC analogue signals. Some microcontrollers have integrated Digital to Analogue Converters (DAC), if not, then a discrete DAC device could be used, or a DAC could be constructed using PWM signals and a few low cost components.

This application note discusses using a PWM timer and a low pass filter to generate a DC level, a ramp and a sine wave.

NEC Electronics microcontrollers can generate PWM signals with a fixed frequency, varying duty cycle and amplitude of $2.2\ V \leq V_{PWM} \leq 5.5\ V$. An example of a PWM signal is shown below.

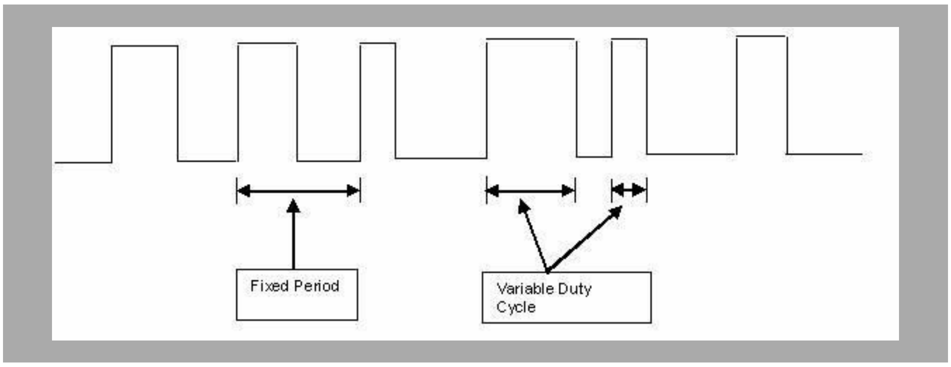

Figure 1-1 Example of a PWM signal

The PWM signals can be decomposed to a D.C. component plus a square wave with identical duty cycle but with time average amplitude of zero. The amplitude of the D.C. component is directly proportional to the PWM cycle. This is shown in the next figure.

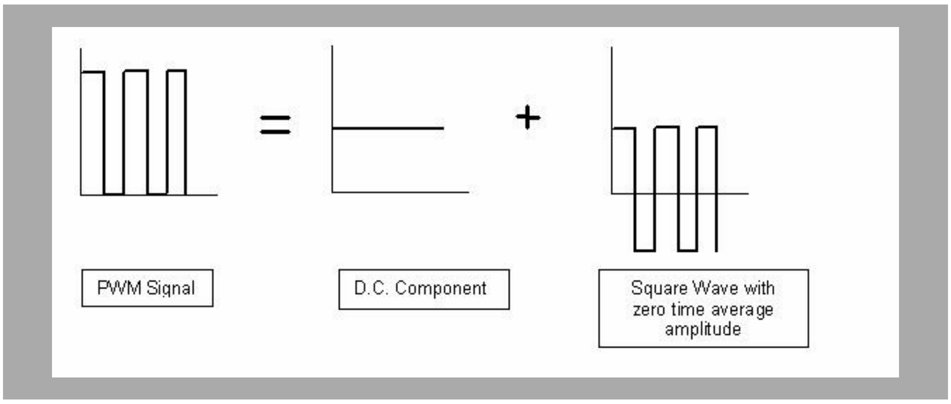

Figure 1-2 Amplitude of the D.C. component

The PWM method of constructing a DAC is not new and in general has been confined to low bandwidth, low resolution applications.

Given that the PWM signal is a sum of a D.C. component which is directly proportional to the duty cycle and a square wave with zero time average amplitude, a low pass filter can be used to filter the signal, remove most of the high frequency components and leave the D.C. component i.e. a DAC can be realised.

The performance of such a DAC is dependant on the PWM frequency and also on the ability of the low pass filter to remove unwanted high frequency components. If a filter with a low bandwidth is used then the bandwidth of the DAC will be limited, if the cut off frequency is too high or the roll off is too slow then the resolution of the DAC will be compromised. Both of these problems can be resolved by increasing the PWM frequency but then the reduced digital resolution reduces the resolution of the DAC.

Chapter 2 Theory

Fourier theory states that any periodic waveform can be decomposed into an infinite sum of harmonics at integer multiples of the periodic frequency. The Fourier series representation of the PWM signal can be simplified by placing the time origin so that the signal becomes an even mathematical function as shown in *Figure 2-1*.

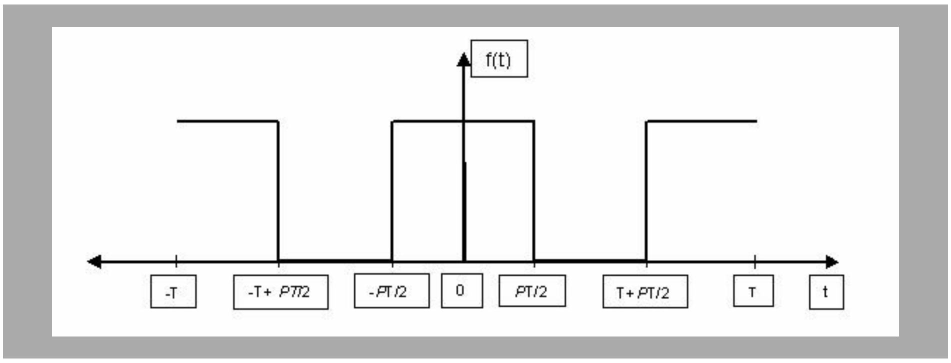

Figure 2-1 PWM signal time-shifted for even symmetry

P is the PWM duty cycle $0 \le P \le 1$ and T denotes the period of the carrier frequency.

The Fourier series representation of an even periodic function f(t) is:

$$f(t) = A_0 + {}^{\infty}\sum_{n=1}[A_n \cos(2n\pi t/T) + B_n \sin(2n\pi t/T)] \quad (1)$$

where:

$$A_0 = 1/2T\ {}^{T}\int_{-T} f(t)dt \quad (2)$$

$$A_n = 1/T\ {}^{T}\int_{-T} f(t) \cos(2n\pi t/T)dt \quad (3)$$

$$B_n = 1/T\ {}^{T}\int_{-T} f(t) \sin(2n\pi t/T)dt \quad (4)$$

Let K be the amplitude of the signal in *Figure 2-1*, then after performing the integrals (2) – (4) you get:

$$A_0 = K \cdot P \quad (5)$$

$$A_n = K \cdot 1/n\pi[\sin(n\pi P) - \sin(2n\pi(1 - P/2))]$$

$$B_n = 0$$

$B_n = 0$ for an even function.

A_0 is the D.C. component, and is the product of the signal amplitude and the duty cycle. By varying the duty cycle of the PWM signal, we will be able to achieve an output voltage where $0 \le A_0 \le K$.

The A_n term represents the amplitudes of the high frequency harmonic components of the PWM signal. The harmonics exist at integer multiples of the PWM carrier frequency 2π/T Hz. If an ideal low pass filter with a cut of frequency which is equal to the PWM carrier frequency is used, then all the high frequency components will be removed and only the D.C. component will be left. In reality, filters do not have ideal characteristics and they will always allow some of the harmonics to pass, which will then appear as ripple voltage in the output.

Chapter 3 DAC Resolution

In most microcontrollers, the PWM signal will be generated by a timer which has a compare register. The timer can be 8 or 16 bits long and will be driven by a clock of frequency f_{ck}. The duty cycle *P* is set by a value stored in the compare register and the PWM period T is set by the frequency of the clock and the length of the counter.

$T = 2^n / f_{ck}$ Where n is the length of the counter in number of bits. (6)

The output of the PWM is initially set to 1 or 0, the timer counts up (or down) and when the timer output is equal to the value stored in the compare register, the output of the PWM toggles, and when the timer overflows (or underflows), the PWM output toggles again. This generates a PWM signal of a frequency 1/T and duty cycle *P*.

The resolution of the DAC is specified by number of bits and this is directly equivalent to the length of the timer counter. The LSB of the PWM DAC is one count and the resolution is the total number of counts.

The desired D/A output is affected by two main sources of errors. The PWM cycle can only be specified with finite resolution is the first source, e.g. if an 8-bit timer is being used and the amplitude of the generated PWM signal is 3.3 V, then the smallest voltage that can be resolved is $3.3\ V/2^8 = 12.89$ mV i.e. the output voltage can only be varied in steps of 12.89 mV. The second source of error is the ripple voltage generated by harmonics that have not been filtered out. The sum of the two sources of error is the *Total Uncertainty*.

Total Uncertainty = duty cycle resolution + harmonic ripple voltage (7)

The Total Uncertainty can be reduced by increasing the resolution of the counter as well as by reducing the harmonics in the output voltage.

Chapter 4 Analogue Low Pass Filters

A complete analysis of low pass filters will not be carried out, as this can easily be obtained from other sources but some properties that are relevant to this application note will be highlighted.

The performance of the PWM based DAC is heavily dependant on the choice of the low pass filter. Passive filters are designed using components such as resistors, capacitors and inductors. Active filters are designed using operational amplifiers. The cost of implementing the filter and the number of components used need to be kept to a minimum otherwise it can soon become more cost effective to use an integrated DAC device.

Unlike passive filters, active filters are not affected by input and output impedance loading. However, the operational amplifiers used in active filters, need to have a gain bandwidth product of 5 to 10 times greater than that of the highest expected input frequency, otherwise the operational amplifier will not be able to filter out the higher frequencies. The cost of operational amplifiers with a high gain bandwidth product might be prohibitive.

There are two critical properties of low pass filters used in PWM DAC applications and these are the bandwidth and the stop-band roll off. The bandwidth of the filter defines the highest frequency the filter will allow to pass before the gain falls to -3dB (0.707). The stop-band roll off defines how quickly, after the -3dB point, the gain decreases with increasing frequency. The combination of the bandwidth and the stop-band roll off determines how much harmonic ripple voltage will be seen in the output.

A passive low pass filter can be constructed using a single capacitor and resistor as shown in *Figure 4-1*.

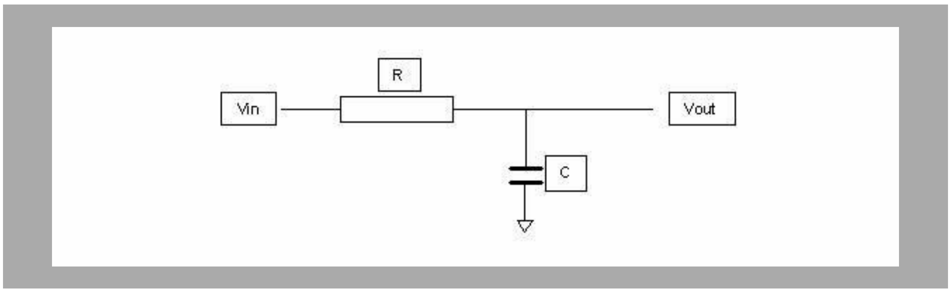

Figure 4-1 1st Order passive low pass filter

The transfer function for a 1st order filter is given by the equation

$$V_{out} / V_{in} = 1/(1 + j\omega RC) \quad (8)$$

The filter bandwidth is given by the equation

$$BW = 1/RC \quad (9)$$

The stop-band roll off of the filter is -20dB/decade, which is slow and would allow a lot off harmonics to be seen in the output voltage as ripple. However, a 2nd order passive filter (shown in *Figure 4-2*) giving a stop band roll off of -40dB/decade can be constructed.

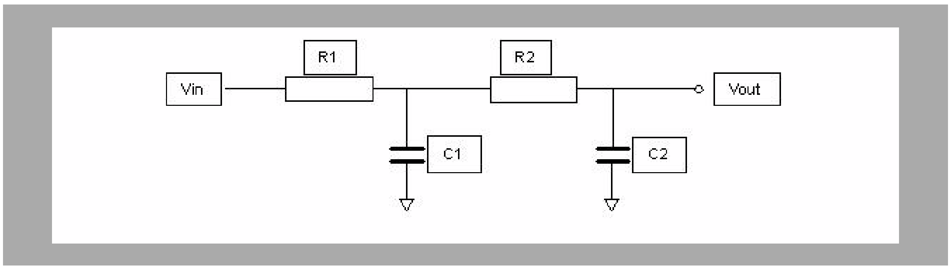

Figure 4-2 2nd Order passive low pass

It can be shown that for the 2nd order filter:

The Cut Off frequency is: $\omega = 1/\sqrt{(R_1 R_2 C_1 C_2)}$ (10)
The Damping factor is: $\zeta = (R_1 C_1 + R_1 C_2 + R_2 C_2)/2\sqrt{(R_1 R_2 C_1 C_2)}$ (11)

Where ω is the un-damped natural frequency in units of $(rad/s)^2$, and ζ is the damping factor. ζ is non dimensional.

The filter's damping ratio ζ is an important parameter as it determines the filters step response to changing the DACs output voltage from one level to another. In order to have fast rise times it is desirable to have as low a value for the damping ratio as possible. However, low damping ratios produce a large step response, overshoots and long settling times. The smallest damping factor with no overshoot in the step response is when $\zeta = 1$, and the smallest damping factor with no resonant peak in the response of the filter is when $\zeta = 0.707$ $(1/\sqrt{2})$. When $\zeta = 0.707$ the bandwidth BW = ω.

As can be seen from equation (11) it is not possible to have a damping ratio of less than 1, and so this filter is not suitable where a small settling time is required for large step changes in the output voltage. However, in this application where only small step changes are made, a Damping factor ≥ 1 is not a problem.

Chapter 5 Circuit Diagrams and Signals

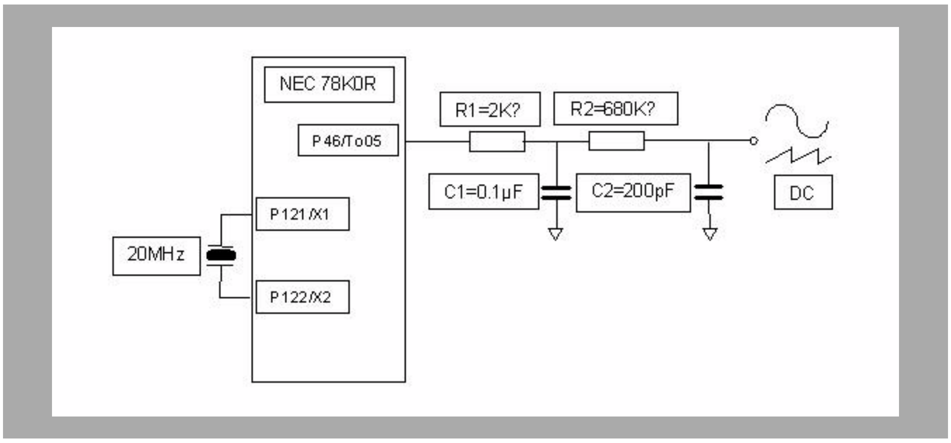

Figure 5-1 Circuit diagram

Figure 5-1 shows the diagram of the circuit used to generate the Sine Wave, Voltage Ramp and DC level.

The Cut Off frequency for the filter = 5363 Hz and the Damping factor = 1.

The same filter is used for all three examples and therefore is not optimised for any of them. In practice the filter would be designed for the particular waveform and frequency required.

The performance of the PWM DAC could be affected by loading, especially dynamic loads and so it would be advisable to buffer the output of the filter e.g. with an operational amplifier buffer.

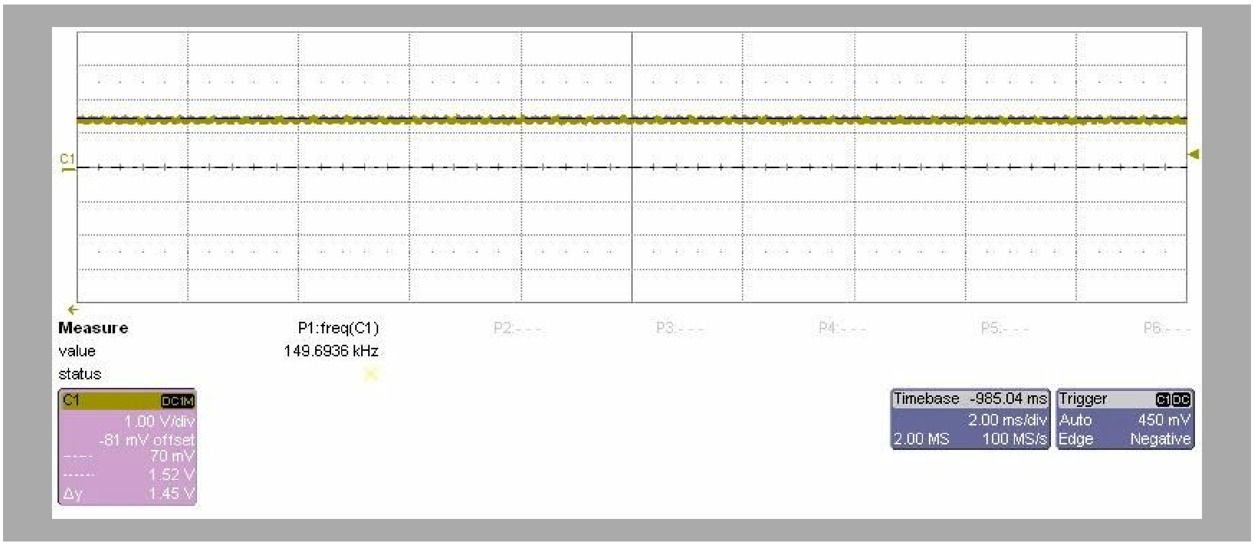

Figure 5-2 DC level

The DC level in *Figure 5-2* was produced by setting the PWM signal duty cycle and not altered until a different DC level is required.

A PWM frequency of 5 KHz was chosen.

The value to be loaded into the timer controlling the PWM frequency will be:
20 MHz/5 KHz = 4000 = 0FA0H.

12. The duty cycle of the PWM waveform can vary from 0% to 100%.
    % Duty Cycle = {(Value of Slave Timer Data Register)/(Value of Master Timer Data Register + 1)} x 100.

Therefore, the register controlling the duty cycle can vary between 0 and 4001 (0FA1H).

The maximum DC voltage is 3 Volts i.e. each increment has a resolution of:
3 Volts/4000 = 750µV.

e.g. To generate a DC level of 0.75 V the value of the duty cycle register will be equal to:

(0.75/3 Volts) x 4000 = 1000 (03E8H).

The register controlling the PWM duty cycle is incremented or decremented to achieve the desired DC level.

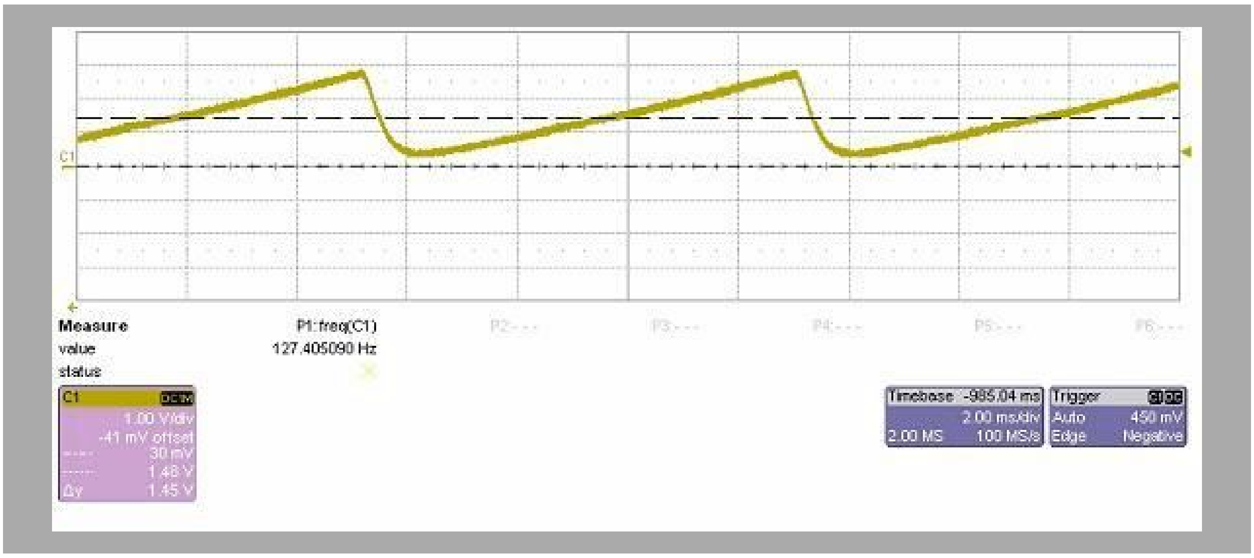

Figure 5-3 64 samples per cycle

The ramp in *Figure 5-3* was produced using 64 samples per cycle.

If a ramp frequency of 125 Hz is required with 32 times oversampling:
The PWM frequency will be:
125 Hz x 64 = 8 KHz

The value to be loaded into the timer controlling the PWM frequency will be:
20 MHz/8 KHz = 2500 = 09C4H

Since the ramp is increasing linearly, each increment up to the maximum value 09C4H will be:
2500/64 = 39.06 = 27H

The sample values are contained in an array at the beginning of the program. The data register of the timer used to set the PWM duty cycle is updated during the timer’s interrupt service routine.

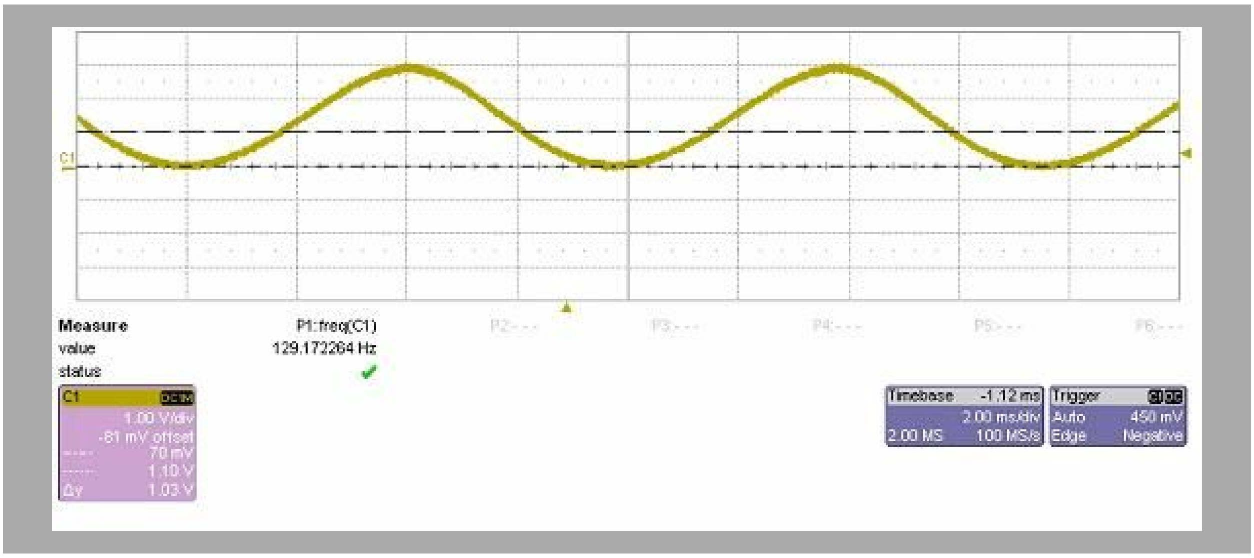

Figure 5-4 74 samples per cycle

The sine wave in *Figure 5-4* was produced using 74 samples per cycle.

The PWM frequency required for a sine wave of 125 Hz is:
125 Hz x 74 = 9250 Hz
The value to be loaded into the timer controlling the PWM frequency will be:
20 MHz/9250 Hz = 2162.161 = 0872 Hz

A sine wave of 3.0 V pk-pk is required i.e. the voltage is resolved to:
3.0V/2162 = 1.39 mV

13. The formula used to generate the values for a sine wave of 125 Hz and 3.0 V pk-pk is:
    {[sinθ + 1] x 1.5}/1.39 mV.

The sample values are contained in an array at the beginning of the program. The data register of the timer used to adjust the PWM duty cycle is updated during the timer’s interrupt service routine.

Chapter 6 Appendix A - Software Listings

6.1 DC Level

```
/=============================================================================
// PROJECT      = 78K0R - PWM DAC
// MODULE       = pwm_init_DC.c
// DEVICE       = 78K0R/KG3 (uPD78F1166)
// VERSION      = 1.0
// DATE         = 04.10.2007
// LAST CHANGE  = -
// ============================================================================
// Description:  Initialization of peripherals and variables
// ============================================================================
// By:          NEC Electronics (Europe) GmbH
// ============================================================================
//-----------------------------------------------------------------------------
// Include files
//-----------------------------------------------------------------------------

#include "defines.h"
#include "lcd.h"

#include <intrinsics.h>

#include "io78f1166_a0.h"
#include "io78f1166_a0_ext.h"

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------
extern __saddr unsigned char menu;
extern __saddr volatile unsigned char sw1_in;

//-----------------------------------------------------------------------------
// Module: HardwareInit
// Description: This module initialize peripheral hardware.
//-----------------------------------------------------------------------------
void HardwareInit(void)               // hardware inizialization
{
//-----------------------------------
// clock generator setting
//-----------------------------------

  OSMC = 0x01;                  // Operation speed mode control register
                                // frequency higher than 10MHz

  OSTS = 0x07;                  // Set osc. stabilization time selection
                                // to 2^18/fx

  CMC = 0x51;                   // Clock operation mode register
                                // X1 osc. mode, XT1 osc. mode, fx > 10MHz

  CKC = 0x08;                   // System clock control register
                                // fclk = fih

  CSC   = 0x00;                 // enable X1 , XT1 operation

  while(OSTC < 0xFF)            // Wait until fX1 clock stabilization
  {                             // time has been elapsed
    __no_operation();
  }

  CKC = 0x18;                   // System clock control register
                                // fclk = fmx = 20MHz
```

```
  CSC = 0x01;                   // stop internal high speed oscillator

  PER0_bit.no0 = 1;             // supply input clock to timer array

//-----------------------------------
// PORT setting
//-----------------------------------

  PM8=0x00;                     // Port initalization for LC display
  PM5_bit.no0=0;
  PM6_bit.no6=0;
  PM6_bit.no5=0;


  PM7 |=0x1f;                   // Port initalization for SW2
  PU7 =0x1f;
  KRM =0x1f;
//  KRMK = 0;                   // enable key interrupt
  KRMK = 1;                     // disble key interrupt


  PM14_bit.no5 = 0;             // Port initalization for LED1
  PM4_bit.no6  = 0;             // Port initalization for LED2

  LED1 = 0;
  LED2 = 0;

//-----------------------------------
// Timer4,5 and Timer6,7 initialization
//-----------------------------------
  TPS0  = 0x0070;               // Timer clock selection register
                                // CKS01 = 156.2kHz; CKS00 = 20MHz

  TMR04 = 0x0800;               // Timer4 mode register, set master mode
  TMR05 = 0x0409;               // Timer5 mode register, set slave mode

  TDR04 = 0x0fa0;               // Timer4 data register, set PWM period (16-bit) 5KHz
  TDR05 = 0x0;                  // Timer5 data register, set initial duty cycle = 0

  TMR06 = 0x0800;               // Timer6 mode register, set master mode
  TMR07 = 0x0409;               // Timer7 mode register, set slave mode

  TDR06 = 0x00ff;               // Timer6 data register, set PWM period (8-bit)
  TDR07 = 0x00ff;               // Timer7 data register, set duty cycle

  TOE0  = 0x00A0;               // Timer output enable register Timer 5 output enabled
  TOM0  = 0x00A0;             // Timer output mode register Timer 5 set to use interrupt from Timer 4
}

//---------------------------------------------------------------------------
// Module: vSoftwareInit
// Description: This module initialize variables.
//---------------------------------------------------------------------------
void SoftwareInit(void)
{
  sw1_in = 0;
  menu   = 0;
}
```

6.2 Voltage Ramp

```
//============================================================================
// PROJECT      = 78K0R - PWM DAC
// MODULE       = pwm_init_ramp.c
// DEVICE       = 78K0R/KG3 (uPD78F1166)
// VERSION      = 1.0
// DATE         = 04.10.2007
```

```
// LAST CHANGE  = -
// ===========================================================================
// Description:  Initialization of peripherals and variables
// ===========================================================================
// By:          NEC Electronics (Europe) GmbH
// ===========================================================================

//--------------------------------------------------------------------------
// Include files
//--------------------------------------------------------------------------

#include "defines.h"
#include "lcd.h"

#include <intrinsics.h>

#include "io78f1166_a0.h"
#include "io78f1166_a0_ext.h"

//--------------------------------------------------------------------------
// Global variables
//--------------------------------------------------------------------------
extern __saddr unsigned char menu;
extern __saddr volatile unsigned char sw1_in;

//--------------------------------------------------------------------------
// Module: HardwareInit
// Description: This module initialize peripheral hardware.
//--------------------------------------------------------------------------
void HardwareInit(void)               // hardware inizialization
{
//-----------------------------------
// clock generator setting
//-----------------------------------

  OSMC = 0x01;                  // Operation speed mode control register
                                // frequency higher than 10MHz

  OSTS = 0x07;                  // Set osc. stabilization time selection
                                // to 2^18/fx

  CMC = 0x51;                   // Clock operation mode register
                                // X1 osc. mode, XT1 osc. mode, fx > 10MHz

  CKC = 0x08;                   // System clock control register
                                // fclk = fih

  CSC   = 0x00;                 // enable X1 , XT1 operation

  while(OSTC < 0xFF)            // Wait until fX1 clock stabilization
  {                             // time has been elapsed
    __no_operation();
  }

  CKC = 0x18;                   // System clock control register
                                // fclk = fmx = 20MHz

  CSC = 0x01;                   // stop internal high speed oscillator

  PER0_bit.no0 = 1;             // supply input clock to timer array

//-----------------------------------
// PORT setting
//-----------------------------------

  PM8=0x00;                     // Port initalization for LC display
  PM5_bit.no0=0;
  PM6_bit.no6=0;
  PM6_bit.no5=0;


  PM7 |=0x1f;                   // Port initalization for SW2
  PU7 =0x1f;
```

```
  KRM =0x1f;
  KRMK = 0;                     // enable key interrupt


  PM14_bit.no5 = 0;             // Port initalization for LED1
  PM4_bit.no6  = 0;             // Port initalization for LED2

  LED1 = 0;
  LED2 = 0;

//-------------------------------------
// Timer4,5 and Timer6,7 initialization
//-------------------------------------
  TPS0  = 0x0000;               // Timer clock selection register
                                // CKS01 = 20MHz; CKS00 = 20MHz

  TMR04 = 0x0800;               // Timer4 mode register, set master mode
  TMR05 = 0x0409;               // Timer5 mode register, set slave mode

  TMMK04 = 0;                   // Enable timer 4 interrupt
  MK2H = 0xff;

//  TDR04 = 0x1389;               // Timer4 data register, set PWM period (16-bit) 4KHz
//  TDR05 = 0x0000;               // Timer5 data register, set initial duty cycle 0%
  TDR04 = 0x9c4;                // Timer4 data register, set PWM period (16-bit) 8KHz
  TDR05 = 0x0000;               // Timer5 data register, set initial duty cycle 0%

  TMR06 = 0x0800;               // Timer6 mode register, set master mode
  TMR07 = 0x0409;               // Timer7 mode register, set slave mode

  TDR06 = 0x00ff;               // Timer6 data register, set PWM period (8-bit)
  TDR07 = 0x00ff;               // Timer7 data register, set duty cycle

  TOE0  = 0x00A0;               // Timer  5 & 7 outputs enabled
  TOM0  = 0x00A0;               // Timer output mode 0 = toggle, 1= combination mode
                                // set by Master int, reset by Slave int
}

//-----------------------------------------------------------------------------
// Module: vSoftwareInit
// Description: This module initialize variables.
//-----------------------------------------------------------------------------
void SoftwareInit(void)
{
  sw1_in = 0;
  menu   = 0;
}




//=============================================================================
// PROJECT      = 78K0R - PWM DAC
// MODULE       = pwm_main_ramp.c
// DEVICE       = 78K0R/KG3 (uPD78F1166)
// VERSION      = 1.0
// DATE         = 04.10.2007
// LAST CHANGE  = -
// ============================================================================
// Description: This sample program demonstrates the generation
//              16-bit PWM and it's use to construct a DAC.
// ============================================================================
// By:          NEC Electronics (Europe) GmbH
// ============================================================================

//-----------------------------------------------------------------------------
// Include files
//-----------------------------------------------------------------------------
#include "defines.h"
#include "lcd.h"

#include <intrinsics.h>
#include <stdlib.h>
```

```
#include "io78f1166_a0.h"
#include "io78f1166_a0_ext.h"


//------------------------------------------------------------------------------
// Option Bytes
//------------------------------------------------------------------------------
#pragma location = "OPTBYTE"
__root const unsigned char opbytes[3]={0x00,0xFF,0x85};

//----------------------------------------------------------------------------------
// Security ID CODE: for OCD on-chip debugging (TK-78K0R + QB-MINI2)
//----------------------------------------------------------------------------------
#pragma location = "SECUID"
__root const unsigned char secuid[10]={0xff,0xff,0xff,0xff,0xff,
                                       0xff,0xff,0xff,0xff,0xff};

//------------------------------------------------------------------------------
// Function prototyps
//------------------------------------------------------------------------------
void HardwareInit(void);
void SoftwareInit(void);

//------------------------------------------------------------------------------
// Global variables
//------------------------------------------------------------------------------
__saddr unsigned char menu;
__saddr volatile unsigned char sw1_in;

__saddr unsigned char  direction;
__saddr unsigned char  Wavetable_index;

//------------------------------------------------------------------------------
// Constants
//------------------------------------------------------------------------------
const char *stext00 ="**  PWM DAC   **";
const char *stext01 ="** Press SW1  **";
const char *stext02 ="** 78K0R/KG3  **";
const char *stext03 ="** 125Hz RAMP **";

unsigned char Wavetable_end;

__near int Wavetable[] = {0x27,0x4e,0x75,0x09c,0xc3,0xea,0x111,0x138,0x15f,
                          0x186,0x1a0,0x1d4,0x1fb,0x222,0x249,0x270,0x297,0x2be,0x2e5,
                          0x30c,0x333,0x35a,0x381,0x3a8,0x3cf,0x3f6,0x41d,0x444,0x46b,
                          0x492,0x4b9,0x4e0,0x507,0x52e,0x555,0x57c,0x5a3,0x5ca,0x5f1,
                          0x618,0x63f,0x666,0x68d,0x6b4,0x6db,0x702,0x729,0x750,0x777,
                          0x79e,0x7c5,0x7ec,0x813,0x83a,0x861,0x888,0x8af,0x8d6,0x8fd,
                          0x924,0x94b,0x972,0x999};

//------------------------------------------------------------------------------
// Module:   wait_n_1ms
// Function: waits for a time 1ms * n
//------------------------------------------------------------------------------
void wait_n_1ms(unsigned char n)
{
  unsigned char i;

  TDR01 = 0x009C;              // Timer1 data register
  TMR01 = 0x8000;              // Timer1 mode register, 1ms interval time
  TS0L_bit.no1 = 1;            // start Timer1
  for(i=0; i < n; i++)
  {
    while(!TMIF01);            // wait for TM01 Interrupt
    TMIF01 = 0;                // clear interrupt flag
  }
  TT0L_bit.no1 = 1;            // stop Timer1
}

//------------------------------------------------------------------------------
// Module:   main
// Function: main function
```

```
//------------------------------------------------------------------------------
void main(void)
{

  HardwareInit();
  SoftwareInit();

  LCD_init();                    // LCD Initialization
  LCD_inst(dclear);              // clear display

  LCD_cursor(0x0);
  LCD_string(&stext00[0]);
  LCD_cursor(0x40);
  LCD_string(&stext01[0]);

  direction = 0;
  Wavetable_index = 0;

  __enable_interrupt();          // enable all interrupts
  while(!sw1_in);                // wait for key press on SW1
  sw1_in = 0;

  LCD_cursor(0x0);
  LCD_string(&stext02[0]);
  LCD_cursor(0x40);
  LCD_string(&stext03[0]);

  Wavetable_end = (sizeof(Wavetable)/sizeof(int));
  TDR05 = Wavetable[Wavetable_index];
  TS0   |= 0x00F0;               // start Timer4, 5 and Timer6, 7
  while(1)
  {
    sw1_in=0;
  }
}

//------------------------------------------------------------------------------
// ISR:      isr_INTTM04
// Function: Timer interrupt service routine - Timer underflow - countdown
//------------------------------------------------------------------------------
#pragma vector = INTTM04_vect
__interrupt void isr_INTTM04(void)
{
  ++Wavetable_index;
  if(Wavetable_index < Wavetable_end)
    TDR05 = Wavetable[Wavetable_index];
  else
  {
    TDR05 = 0;
    Wavetable_index = 0;
  }
}
//------------------------------------------------------------------------------
// ISR:      isr_INTKR
// Function: Key interrupt service routine / navigator switch / key debouncing
//------------------------------------------------------------------------------
#pragma vector = INTKR_vect
__interrupt void isr_INTKR(void)
{
  unsigned char sw1_first,sw1_second;

  sw1_first= (~P7) & 0x1f;      // read SW1 first time
  TDR00 = 0x061A;               // Timer data register
  TMR00 = 0x8000;               // Timer mode register
  TS0L_bit.no0 = 1;             // start Timer
  while(!TMIF00);               // wait for TM00 Interrupt, interval time = 10ms
  TMIF00 = 0;
  TT0L_bit.no0 = 1;             // stop timer
  sw1_second= (~P7) & 0x1f;     // read SW1 second time
  if(sw1_first==sw1_second) sw1_in=sw1_first; // debounce SW1
  else sw1_in=0;
}
```

6.3 Sine Wave

```
//===========================================================================
// PROJECT      = 78K0R - PWM DAC
// MODULE       = pwm_init_sine.c
// DEVICE       = 78K0R/KG3 (uPD78F1166)
// VERSION      = 1.0
// DATE         = 04.10.2007
// LAST CHANGE  = -
// ==========================================================================
// Description:  Initialization of peripherals and variables
// ==========================================================================
// By:          NEC Electronics (Europe) GmbH
// ==========================================================================

//--------------------------------------------------------------------------
// Include files
//--------------------------------------------------------------------------

#include "defines.h"
#include "lcd.h"

#include <intrinsics.h>

#include "io78f1166_a0.h"
#include "io78f1166_a0_ext.h"

//--------------------------------------------------------------------------
// Global variables
//--------------------------------------------------------------------------
extern __saddr unsigned char menu;
extern __saddr volatile unsigned char sw1_in;

//--------------------------------------------------------------------------
// Module: HardwareInit
// Description: This module initialize peripheral hardware.
//--------------------------------------------------------------------------
void HardwareInit(void)                // hardware inizialization
{
//------------------------------------
// clock generator setting
//------------------------------------

  OSMC = 0x01;                   // Operation speed mode control register
                                 // frequency higher than 10MHz

  OSTS = 0x07;                   // Set osc. stabilization time selection
                                 // to 2^18/fx

  CMC = 0x51;                    // Clock operation mode register
                                 // X1 osc. mode, XT1 osc. mode, fx > 10MHz

  CKC = 0x08;                    // System clock control register
                                 // fclk = fih

  CSC   = 0x00;                  // enable X1 , XT1 operation

  while(OSTC < 0xFF)             // Wait until fX1 clock stabilization
  {                              // time has been elapsed
    __no_operation();
  }

  CKC = 0x18;                    // System clock control register
                                 // fclk = fmx = 20MHz

  CSC = 0x01;                    // stop internal high speed oscillator

  PER0_bit.no0 = 1;              // supply input clock to timer array
```

```
//-----------------------------------
// PORT setting
//-----------------------------------

  PM8=0x00;                     // Port initalization for LC display
  PM5_bit.no0=0;
  PM6_bit.no6=0;
  PM6_bit.no5=0;


  PM7 |=0x1f;                   // Port initalization for SW2
  PU7 =0x1f;
  KRM =0x1f;
  KRMK = 0;                     // enable key interrupt


  PM14_bit.no5 = 0;             // Port initalization for LED1
  PM4_bit.no6  = 0;             // Port initalization for LED2

  LED1 = 0;
  LED2 = 0;

//-----------------------------------
// Timer4,5 and Timer6,7 initialization
//-----------------------------------
  TPS0  = 0x0000;               // Timer clock selection register
                                // CKS01 = 20MHz; CKS00 = 20MHz

  TMR04 = 0x0800;               // Timer4 mode register, set master mode
  TMR05 = 0x0409;               // Timer5 mode register, set slave mode

  TMMK04 = 0;                   // Enable timer 4 interrupt
  MK2H = 0xff;

  TDR04 = 0x872;               // Timer4 data register, set PWM period (16-bit) 9.25KHz
//  TDR04 = 0x1389;               // Timer4 data register, set PWM period (16-bit) 4KHz
//  TDR04 = 0x9c4;               // Timer4 data register, set PWM period (16-bit) 8KHz
  TDR05 = 0x0000;               // Timer5 data register, set initial duty cycle 0%

  TMR06 = 0x0800;               // Timer6 mode register, set master mode
  TMR07 = 0x0409;               // Timer7 mode register, set slave mode

  TDR06 = 0x00ff;               // Timer6 data register, set PWM period (8-bit)
  TDR07 = 0x00ff;               // Timer7 data register, set duty cycle

  TOE0  = 0x00A0;               // Timer  5 & 7 outputs enabled
  TOM0  = 0x00A0;               // Timer output mode 0 = toggle, 1= combination mode
                                // set by Master int, reset by Slave int
}

//---------------------------------------------------------------------------
// Module: vSoftwareInit
// Description: This module initialize variables.
//---------------------------------------------------------------------------
void SoftwareInit(void)
{
  sw1_in = 0;
  menu   = 0;
}

//===========================================================================
// PROJECT      = 78K0R - PWM DAC
// MODULE       = pwm_sine_main.c
// DEVICE       = 78K0R/KG3 (uPD78F1166)
// VERSION      = 1.0
// DATE         = 04.10.2007
// LAST CHANGE  = -
// ==========================================================================
// Description: This sample program demonstrates the generation
//              16-bit PWM and it's use to construct a DAC.
// ==========================================================================
// By:          NEC Electronics (Europe) GmbH
// ==========================================================================
```

```
//------------------------------------------------------------------------------
// Include files
//------------------------------------------------------------------------------
#include "defines.h"
#include "lcd.h"

#include <intrinsics.h>
#include <stdlib.h>

#include "io78f1166_a0.h"
#include "io78f1166_a0_ext.h"


//------------------------------------------------------------------------------
// Option Bytes
//------------------------------------------------------------------------------
#pragma location = "OPTBYTE"
__root const unsigned char opbytes[3]={0x00,0xFF,0x85};

//----------------------------------------------------------------------------------
// Security ID CODE: for OCD on-chip debugging (TK-78K0R + QB-MINI2)
//----------------------------------------------------------------------------------
#pragma location = "SECUID"
__root const unsigned char secuid[10]={0xff,0xff,0xff,0xff,0xff,
                                       0xff,0xff,0xff,0xff,0xff};

//------------------------------------------------------------------------------
// Function prototyps
//------------------------------------------------------------------------------
void HardwareInit(void);
void SoftwareInit(void);

//------------------------------------------------------------------------------
// Global variables
//------------------------------------------------------------------------------
__saddr unsigned char menu;
__saddr volatile unsigned char sw1_in;

__saddr unsigned char  direction;
__saddr unsigned char  Wavetable_index;

//------------------------------------------------------------------------------
// Constants
//------------------------------------------------------------------------------
const char *stext00 ="**  PWM DAC   **";
const char *stext01 ="** Press SW1  **";
const char *stext02 ="** 78K0R/KG3  **";
const char *stext03 ="125Hz Sine Wave ";

unsigned char Wavetable_end;

__near int Wavetable[ ] = {0x00,0x04,0x10,0x24,0x41,0x65,0x90,0xc3,0xfc,0x13c,0x181,0x1cc,
                          0x21b,0x26f,0x2c6,0x31f,0x37b,0x3d9,0x437,0x495,0x4f2,0x54e,
                          0x5a8,0x5ff,0x652,0x6a2,0x6ec,0x732,0x771,0x7ab,0x7dd,0x809,
                          0x82d,0x849,0x85d,0x86a,0x86e};
//------------------------------------------------------------------------------
// Module:   wait_n_1ms
// Function: waits for a time 1ms * n
//------------------------------------------------------------------------------
void wait_n_1ms(unsigned char n)
{
  unsigned char i;

  TDR01 = 0x009C;              // Timer1 data register
  TMR01 = 0x8000;              // Timer1 mode register, 1ms interval time
  TS0L_bit.no1 = 1;            // start Timer1
  for(i=0; i < n; i++)
  {
    while(!TMIF01);            // wait for TM01 Interrupt
    TMIF01 = 0;                // clear interrupt flag
  }
  TT0L_bit.no1 = 1;            // stop Timer1
```

```
}

//--------------------------------------------------------------------------
// Module:   main
// Function: main function
//--------------------------------------------------------------------------
void main(void)
{

  HardwareInit();
  SoftwareInit();

  LCD_init();                   // LCD Initialization
  LCD_inst(dclear);             // clear display

  LCD_cursor(0x0);
  LCD_string(&stext00[0]);
  LCD_cursor(0x40);
  LCD_string(&stext01[0]);

  direction = 0;
  Wavetable_index = 0;

  __enable_interrupt();         // enable all interrupts

  while(!sw1_in);               // wait for key press on SW1
  sw1_in = 0;

  LCD_cursor(0x0);
  LCD_string(&stext02[0]);
  LCD_cursor(0x40);
  LCD_string(&stext03[0]);

  Wavetable_end = (sizeof(Wavetable)/sizeof(int));
  Wavetable_end = Wavetable_end - 1;
  TDR05 = Wavetable[Wavetable_index];
  TS0   |= 0x00F0;              // start Timer4, 5 and Timer6, 7

  while(1)
  {
    sw1_in=0;
    if(direction ==0 && Wavetable_index >= Wavetable_end)
      direction = 1;
    if(direction ==1 && Wavetable_index == 0)
      direction = 0;
  }
}

//--------------------------------------------------------------------------
// ISR:      isr_INTTM04
// Function: Timer interrupt service routine - Timer underflow - countdown
//--------------------------------------------------------------------------
#pragma vector = INTTM04_vect
__interrupt void isr_INTTM04(void)
{
  switch(direction)
  {
    case 0: // direction = down the Wavetable array
    if(Wavetable_index != Wavetable_end)
    {
      ++Wavetable_index;
      TDR05 = Wavetable[Wavetable_index];
      break;
    }
    else
    {
      TDR05 = Wavetable[Wavetable_index];
      ++Wavetable_index;
      break;
    }
    case 1: // direction  = up the Wavetable array
      if(Wavetable_index != 0)
      {
```

```
        -- Wavetable_index;
        TDR05 = Wavetable[Wavetable_index];
      }
      else
        TDR05 = Wavetable[Wavetable_index];
      break;

    default: break;
  }
}
//---------------------------------------------------------------------------
// ISR:      isr_INTKR
// Function: Key interrupt service routine / navigator switch / key debouncing
//---------------------------------------------------------------------------
#pragma vector = INTKR_vect
__interrupt void isr_INTKR(void)
{
  unsigned char sw1_first,sw1_second;

  sw1_first= (~P7) & 0x1f;      // read SW1 first time
  TDR00 = 0x061A;               // Timer data register
  TMR00 = 0x8000;               // Timer mode register
  TS0L_bit.no0 = 1;             // start Timer
  while(!TMIF00);               // wait for TM00 Interrupt, interval time = 10ms
  TMIF00 = 0;
  TT0L_bit.no0 = 1;             // stop timer
  sw1_second= (~P7) & 0x1f;     // read SW1 second time
  if(sw1_first==sw1_second) sw1_in=sw1_first; // debounce SW1
  else sw1_in=0;
}
```