Application Note

High Analog Integration Tuning-Fork Level-Switch Solution

Stanley Dai, Lars Lotzenburger

ABSTRACT

Level detection is often needed in industrial applications for process control or alarms. In these applications, using a tuning fork for level detection is popular, because tuning forks can work with most viscous liquids and withstand harsh conditions, such as highly corrosive liquids and high working temperatures. This application note introduces a digital solution based on the TI MSP430FR235x microcontroller to implement a tuning fork level switch sensor. This digital solution is low cost and easy to accomplish compared to traditional analog system implementations. Programming flow, source code, and on-bench test results are included.

The source code described in this application note can be downloaded from https://www.ti.com/lit/zip/slaae63.

Table of Contents

Trademarks

1 Introduction

Tuning fork level switch sensors are used for monitoring viscous liquid levels in industrial applications to detect levels of liquid, grain, or powder-like material in a storage tank as shown in Figure 1-1. The physics behind this method is that a piezoelectric crystal oscillates a tuning fork at its natural frequency, as shown in Figure 1-2. The frequency and amplitude of the sensor depends on the medium in which the tuning fork is immersed. If the fork is immersed in material (tank filled up to point where sensor is installed) the resonance frequency and amplitude change correspondingly.

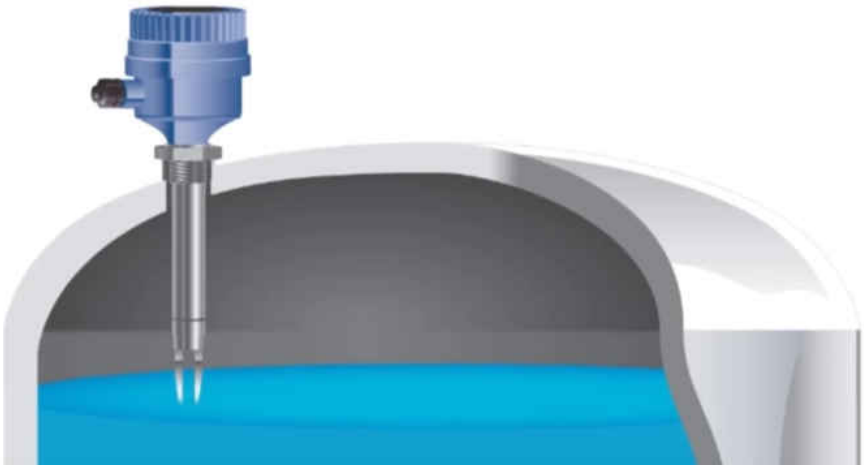

Figure 1-1. Tuning Fork Level Industry Application Diagram

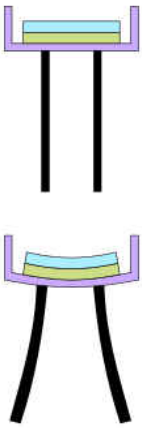

Figure 1-2. Physical Oscillation of Piezo-Electric Diagram

Members of the MSP430FR235x microcontroller (MCU) series are ultra-low power low-cost devices for sensing and measurement applications with analog integration. MSP430FR235x MCUs integrate four configurable smart analog combos (SACs) for analog signal conditioning. Each SAC features a 12-bit digital-to-analog converter (DAC) and a configurable programmable-gain amplifier (PGA) to meet the specific needs of a system while reducing the BOM and PCB size. The integrated 12-bit 200-ksps analog-to-digital converter (ADC) digitized the analog input value for further processing. Timers generate the driving signal and the ADC conversion timing. This application note implements a low cost and compact digital solution for tuning fork level switches based on the MSP430FR2355 MCU.

2 Hardware Implementation

The sensor uses two piezoelectric actuators, one to drive the tuning fork (TX) and the other to receive the response of the fork (RX). An MCU timer module is used to produce a 50:50 duty cycle frequency sweep driving the TX. To measure higher-viscousity liquids, the sweep frequency range is set from 800 Hz to 1498 Hz with

the increment of 2 Hz after each period, which results in (1498 Hz – 800 Hz) / 2 Hz = 350 periods per sweep (approximately 600-ms duration). Figure 2-1 shows the TX sweep signal and the expected RX response signal.

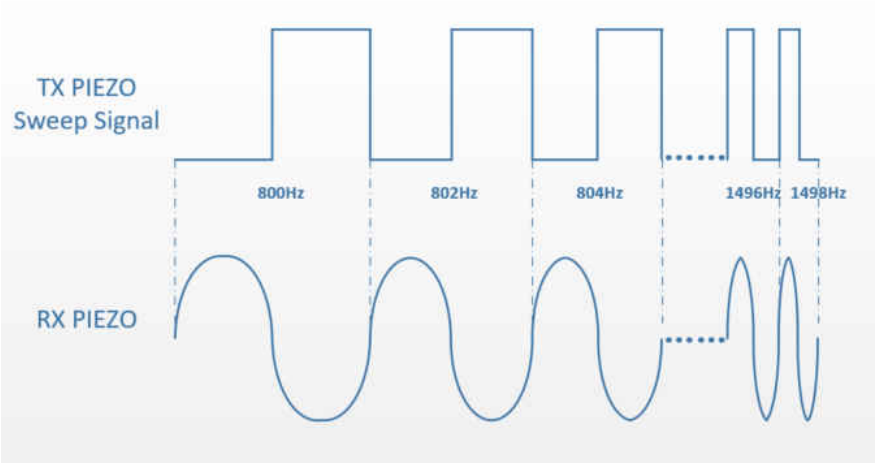

Figure 2-1. TX Piezo Sweep Signal and RX Piezo Response Signal Waveform

The resonant frequency of the tested tuning fork in air is approximately 1368 Hz, which is determined by type of fork level sensor. Each different type of tuning fork level sensor has a different resonant frequency, so the user must adjust the example sweep signal range accordingly. The MCU timer is set for running at 8-MHz frequency. The discrete steps of the timer for the sweep signal generation introduce a maximum error of ±0.13 Hz for a given frequency. Two MCU output pins are used for the PWM signal for better drive capability of the TX sweep signal. Optionally, the user can add external buffers or switches to add even more driving capability of the sweep signal.

Figure 2-2 shows the block diagram. In a tuning fork level sensor switch application, the power is typically from 4-20mA loop, so an additional LDO is needed. This example application uses a high-voltage input LDO (30 V), the TLV76033, to power the MCU. Because the MSP430FR2355 integrates an SVS function, no external voltage monitor IC is needed.

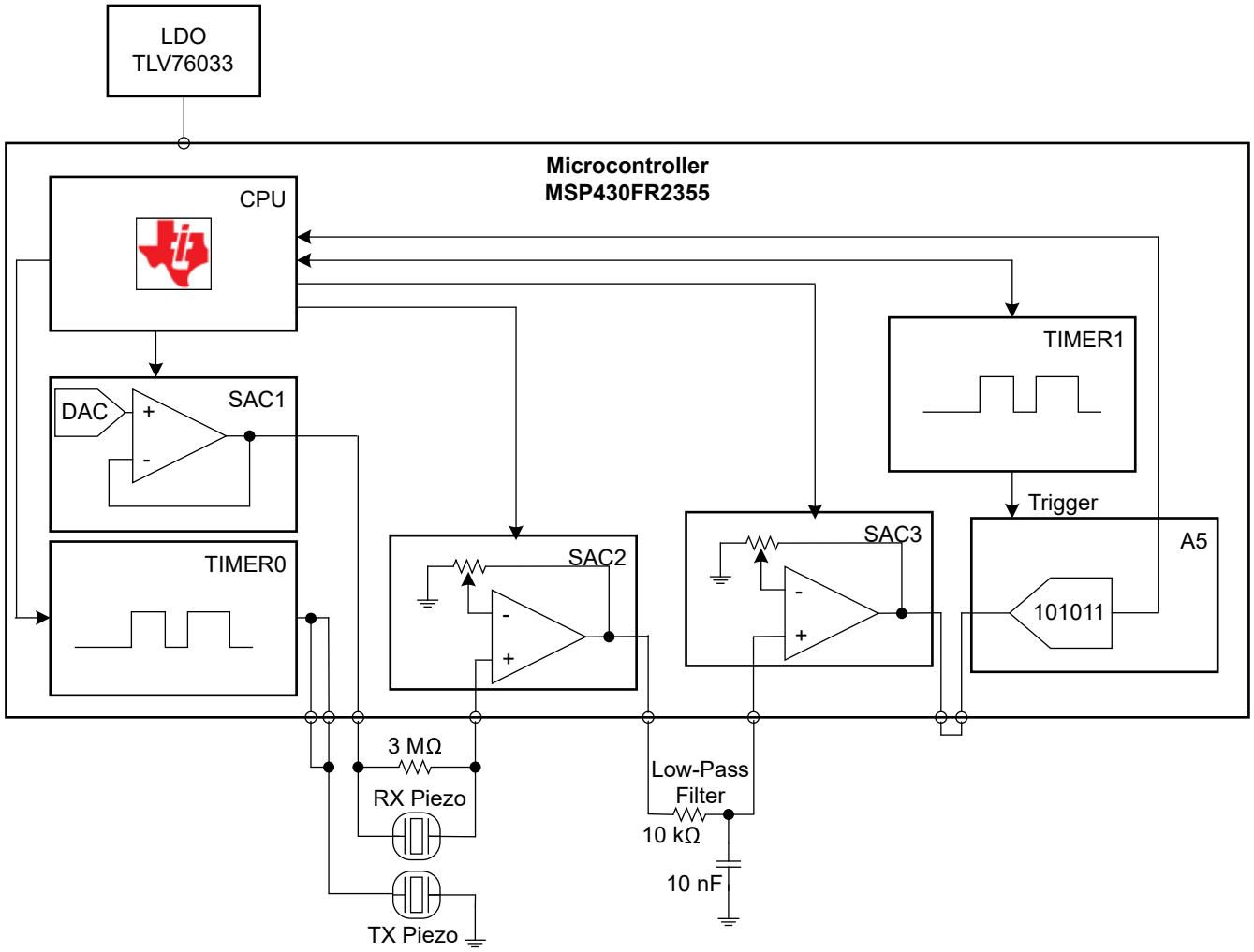

Figure 2-2. System Diagram Block Based on MSP430FR2355

For signal conditioning, three SAC modules are used to offset, filter, and amplify the response signal of the RX piezo. The RX piezo itself is referenced by a positive voltage to avoid negative input voltages at the MCU pins. The biasing voltage is generated by SAC1, which is configured in DAC buffer mode. The user can adjust the DC bias voltage through the MCU internal DAC function for different tuning fork level sensor types. In this example, , 500 mVDC is chosen.

SAC2 is configured as noninverting PGA with fixed gain to use the analog input range of the ADC of 0 to 2.5 V with the RX piezo in air, which typically has the highest amplitude. In this way, MCU internal ADC can maximize the sample accuracy. And another benefit of the noninverting amplifier configuration is the high input impedance required for the RX piezo. In this scenario, 2-V/V amplify gain is chosen. Additionally, an external low-pass filter (R-C) is implemented to limit the bandwidth of the input signal. The corner frequency is based on the natural resonant frequency in the air of the tuning fork level sensor. In this scenario, the natural resonant frequency is 1368 Hz, so a low-pass filter (R = 10 kΩ, C = 10 nF) is chosen based on Equation 1.

$$f_{corner} = \frac{1}{2\pi RC} \geq 1368Hz \quad (1)$$

SAC3 is also configured as noninverting PGA to add gain to the amplitude of the RX signal for higher viscosity liquids. Typically, higher-viscosity liquids have lower resonant amplitudes, which can be hard to measure by the MCU ADC. The user can adjust the gain of SAC3 corresponding. In this scenario, 1 V/V is chosen. In addition, a 3-MΩ resistor is recommended in parallel to the RX piezo to decrease the output impedance.

The ADC converter sample point is dependent on the TX driving signal. Tests have shown that the maximum RX signal amplitude happens at the resonance frequency, which has fixed phase shift of 80 degrees if the tuning

fork is in liquid instead of air. This shift is independent of the viscosity of the liquid. Therefore, the RX signal can be sampled once per TX period with an 80-degree shift to the TX signal to decrease the amount of data to be processed. Because the peak of the RX amplitude is at a different phase compared to out-of-resonance air frequencies, the selectivity of the resonance frequency is further enhanced. The software can detect if the tuning fork level sensor is immersed in the target viscous liquid, therefore outputting a switch signal indicating the liquid level is achieved.

3 Software Implementation

This section describes how to implement the MCU software based on the hardware system in Section 2. Figure 3-1 shows the details of programming flow. Only the highest amplitude value is needed to find the corresponding resonant frequency, so the ADC array needs to store only the highest value.

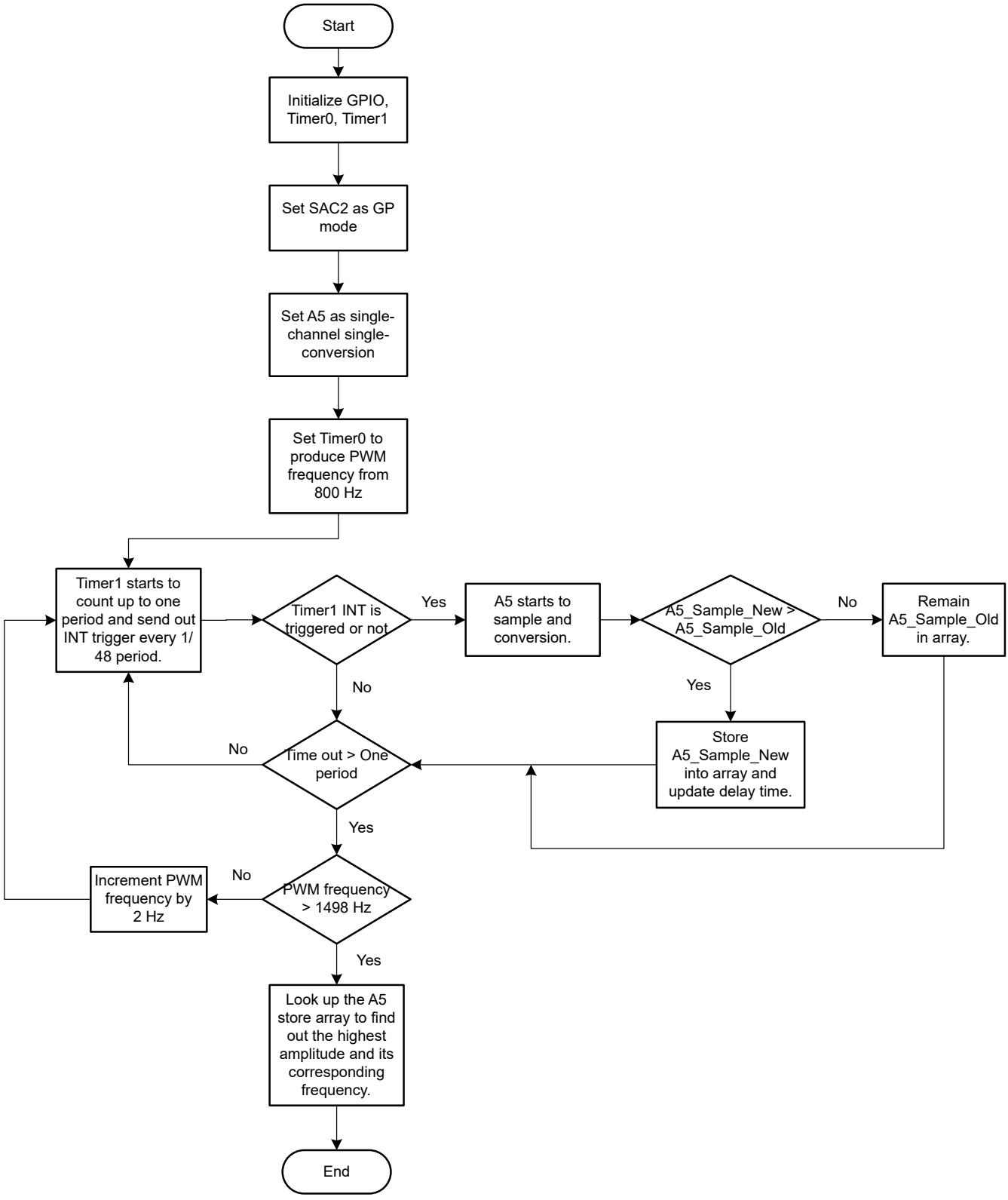

Figure 3-1. Software Programming Flow Based on MSP430FR2355

The MCU program is interrupt-driven to save power during inactive phases by entering the sleep mode, which improves low power consumption.

The software source code is available in Section 7.

4 Bench Test Performance

Lab tests were performed using the MSP430FR2355 LaunchPad™ development kit. The program was developed with Code Composer Studio™ IDE (Version 12.0.0), and the code size is approximately 22KB.

The system was tested in air, water and oil. Figure 4-1 shows the sweep signal of the TX piezo (turquoise signal) and the RX ADC input signals for air, water and oil (blue signals). The RX piezo response signal is biased with 500 mVDC produced by SAC1 in DAC mode. The gain of SAC2 is fixed to 2 V/V (noninverting amplifier configuration) and boosts the input signal to approximately 2.3 Vpp, which is close to the ADC input range of 0 to 2.5 V. The gain of SAC3 is the default of 1 V/V (noninverting amplifier configuration) and can be adjusted for higher viscosity liquids.

The purple waveform in Figure 4-1 represents the trigger signal. When the signal is logical high status, the sweep-up (800 to 1498 Hz) is performed. The logical low status of the trigger signal indicates a fast sweep down process. Back-to-back sweeps have shown unwanted oscillations as the piezo cannot keep up with the sudden frequency change from 1498 Hz to 800 Hz, so this is not recommended.

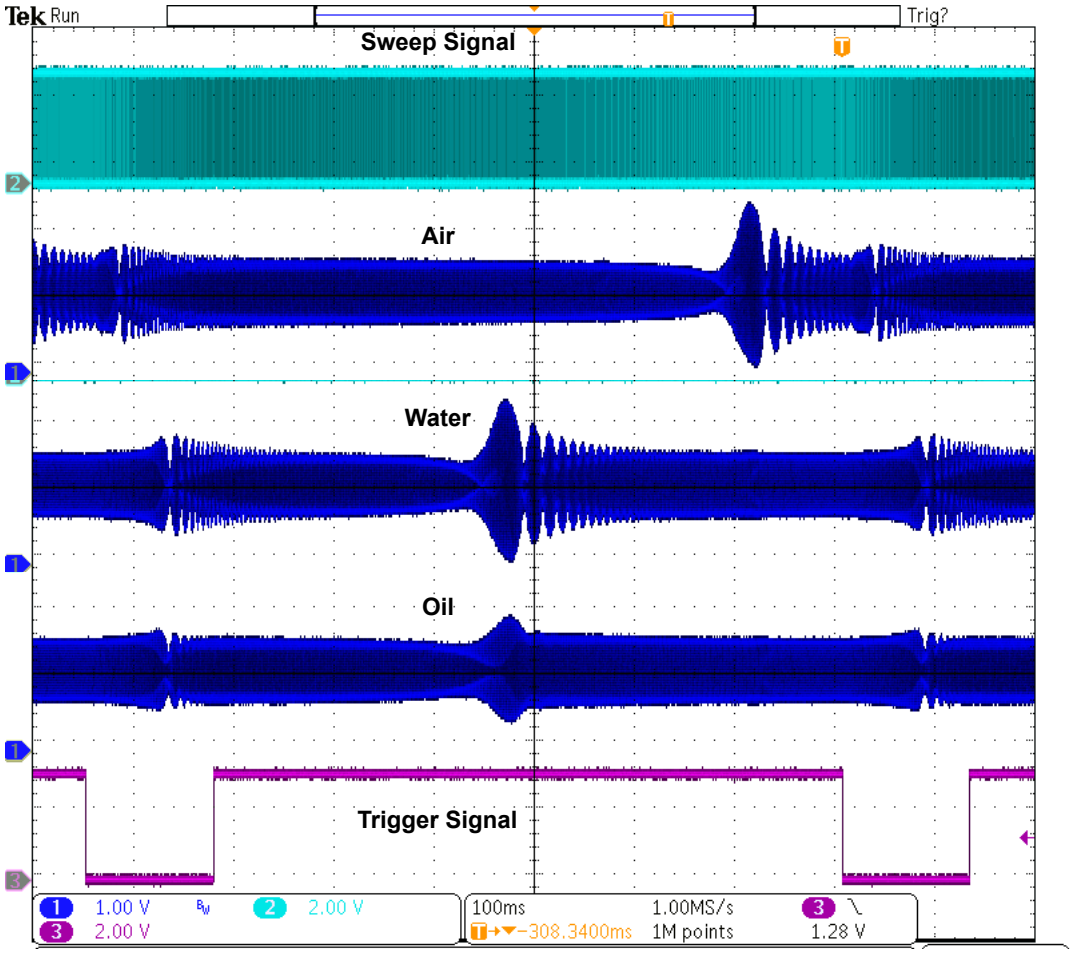

Figure 4-1. TX Piezo Sweep Signal, RX Piezo Response Signal, and Frequency Sweep Direction Trigger Signal

The resonance frequency is easily detectable by its peak amplitude. A separation of the resonance frequencies in air and the liquids is also clearly determinable. The resonance frequency of different liquids is approximately the same, near 1070 Hz. Therefore, the detection of liquid presence can be as simple as defining a threshold between the resonance frequency in air and in the liquids.

Figure 4-2 shows the ADC data sampled by the MCU. The X-axis represents the 350 periods per sweep, and the Y-axis is the digitized value (0 to 4095). The 80-degree shift of peak amplitude phase delay further increases the selectivity as the input signal decreases before ramping up to the resonance frequency. The three

measurements use the same SAC settings (SAC1 with 500-mVDC output, SAC2 gain of 2 V/V, and SAC3 gain of 1 V/V) to demonstrate the amplitude attenuation in the different mediums.

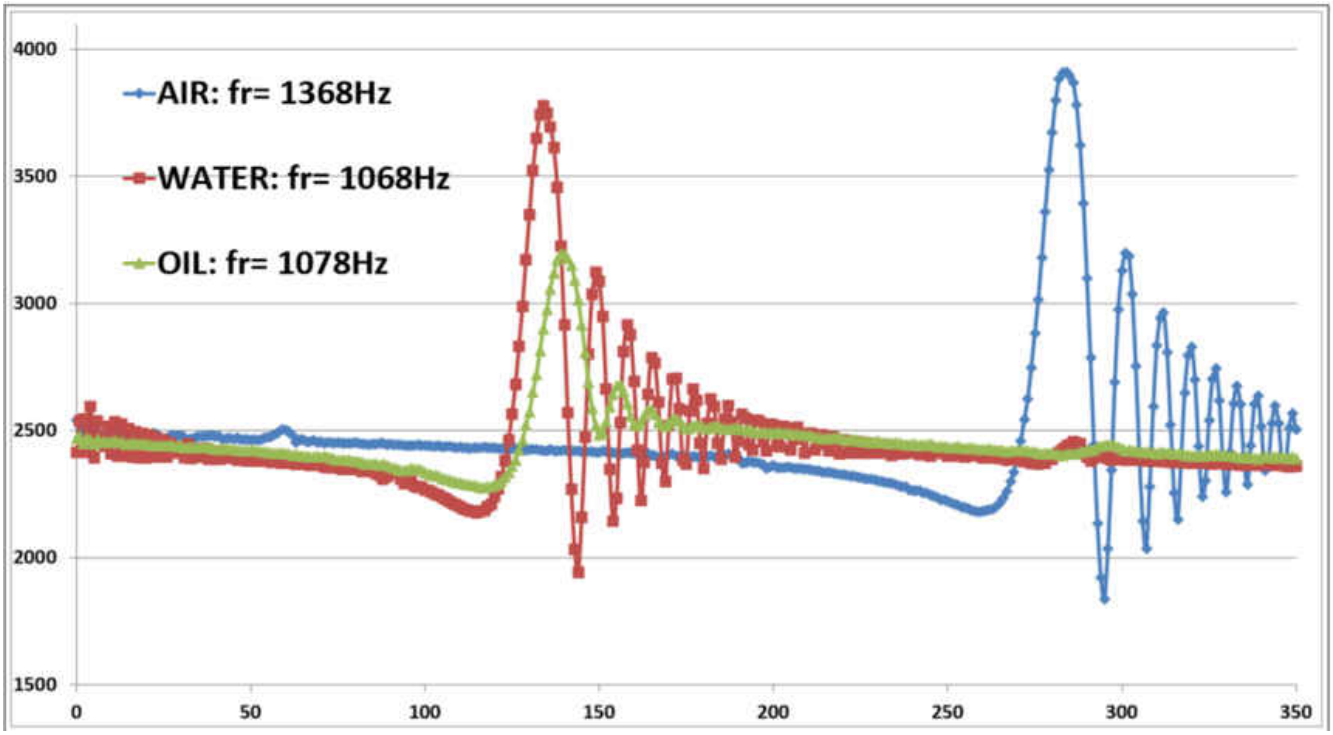

Figure 4-2. ADC Sample Data of RX Piezo Signal

Note that measurements are taken from a specific tuning fork level sensor. Results and optimum sampling times vary with different types of tuning forks.

5 Device Recommendation

The device used in this example is part of MSP430 microcontrollers portfolio of low-cost MCUs, designed for sensing and measurement applications. This example can use the devices shown in Table 5-1. For more information on the entire portfolio, visit the MSP430 microcontrollers overview.

Table 5-1. Device Recommendations

Part Number	Key Features
MSP430FR2353	16 KB FRAM, DAC, OpAmp, 12-bit ADC
MSP430FR2355	32 KB FRAM, DAC, OpAmp, 12-bit ADC

6 References

1. MSP430FR2355 Product Page
2. TLV76033 Product Page
3. Smart Analog Combo Enabling Tomorrow's Sensing and Measurement Applications
4. How to Use the Smart Analog Combo in MSP430™ MCUs

7 Source Code

The source code shown below can also be downloaded from https://www.ti.com/lit/zip/slaae63.

```
/* --COPYRIGHT--,BSD
 * Copyright (c) 2017, Texas Instruments Incorporated
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 * --/COPYRIGHT--*/
//!*****************************************************************************
//! SAC - SAC-L3, General-Purpose Mode
//!
//! Description: Configure SAC-L3 for general purpose mode.
//! The "+" terminal is connected to OA0+ and the "-" terminal is connected to OA0-.
//! OA0_ and OA0O are connected as below to implement x3 amplifier.
//! SAC OA is selected to low speed and low power mode.
//! ACLK = n/a, MCLK = SMCLK = default DCODIV ~1MHz.
//!
//!           MSP430FR2xx_4xx Board
//!             -------------------
//!         /|\|                   OA0-|----||---------R2=100Kohm---GND
//!          | |                       |  R1=200Kohm
//!          | |                   OA0O|----||
//!          --|RST                    |
//!            |                   OA0+|<----
//!            |                       |
//!            |                       |
//!
//! Wallace Tran
//! Texas Instruments Inc.
//! May 2018
//*****************************************************************************
#include "Board.h"
#include "driverlib.h"
#include <string.h>
//***Lookup Table Setup***//
#define PRDS_PER_SWEEP 350
uint16_t peak[PRDS_PER_SWEEP];
uint16_t phase[PRDS_PER_SWEEP];
const uint16_t TMRCCR[PRDS_PER_SWEEP] =
{
 10000    ,
9975    ,
9950    ,
9926    ,
9901    ,
9877    ,
9852    ,
9828    ,
9804    ,
9780    ,
9756    ,
```

```
9732    ,
9709    ,
9685    ,
9662    ,
9639    ,
9615    ,
9592    ,
9569    ,
9547    ,
9524    ,
9501    ,
9479    ,
9456    ,
9434    ,
9412    ,
9390    ,
9368    ,
9346    ,
9324    ,
9302    ,
9281    ,
9259    ,
9238    ,
9217    ,
9195    ,
9174    ,
9153    ,
9132    ,
9112    ,
9091    ,
9070    ,
9050    ,
9029    ,
9009    ,
8989    ,
8969    ,
8949    ,
8929    ,
8909    ,
8889    ,
8869    ,
8850    ,
8830    ,
8811    ,
8791    ,
8772    ,
8753    ,
8734    ,
8715    ,
8696    ,
8677    ,
8658    ,
8639    ,
8621    ,
8602    ,
8584    ,
8565    ,
8547    ,
8529    ,
8511    ,
8493    ,
8475    ,
8457    ,
8439    ,
8421    ,
8403    ,
8386    ,
8368    ,
8351    ,
8333    ,
8316    ,
8299    ,
8282    ,
8264    ,
8247    ,
8230    ,
8214    ,
8197    ,
8180    ,
```

```
8163    ,
8147    ,
8130    ,
8114    ,
8097    ,
8081    ,
8065    ,
8048    ,
8032    ,
8016    ,
8000    ,
7984    ,
7968    ,
7952    ,
7937    ,
7921    ,
7905    ,
7890    ,
7874    ,
7859    ,
7843    ,
7828    ,
7813    ,
7797    ,
7782    ,
7767    ,
7752    ,
7737    ,
7722    ,
7707    ,
7692    ,
7678    ,
7663    ,
7648    ,
7634    ,
7619    ,
7605    ,
7590    ,
7576    ,
7561    ,
7547    ,
7533    ,
7519    ,
7505    ,
7491    ,
7477    ,
7463    ,
7449    ,
7435    ,
7421    ,
7407    ,
7394    ,
7380    ,
7366    ,
7353    ,
7339    ,
7326    ,
7313    ,
7299    ,
7286    ,
7273    ,
7260    ,
7246    ,
7233    ,
7220    ,
7207    ,
7194    ,
7181    ,
7168    ,
7156    ,
7143    ,
7130    ,
7117    ,
7105    ,
7092    ,
7080    ,
7067    ,
7055    ,
7042    ,
```

```
7030    ,
7018    ,
7005    ,
6993    ,
6981    ,
6969    ,
6957    ,
6944    ,
6932    ,
6920    ,
6908    ,
6897    ,
6885    ,
6873    ,
6861    ,
6849    ,
6838    ,
6826    ,
6814    ,
6803    ,
6791    ,
6780    ,
6768    ,
6757    ,
6745    ,
6734    ,
6723    ,
6711    ,
6700    ,
6689    ,
6678    ,
6667    ,
6656    ,
6645    ,
6633    ,
6623    ,
6612    ,
6601    ,
6590    ,
6579    ,
6568    ,
6557    ,
6547    ,
6536    ,
6525    ,
6515    ,
6504    ,
6494    ,
6483    ,
6472    ,
6462    ,
6452    ,
6441    ,
6431    ,
6421    ,
6410    ,
6400    ,
6390    ,
6380    ,
6369    ,
6359    ,
6349    ,
6339    ,
6329    ,
6319    ,
6309    ,
6299    ,
6289    ,
6279    ,
6270    ,
6260    ,
6250    ,
6240    ,
6231    ,
6221    ,
6211    ,
6202    ,
6192    ,
6182    ,
```

```
6173    ,
6163    ,
6154    ,
6144    ,
6135    ,
6126    ,
6116    ,
6107    ,
6098    ,
6088    ,
6079    ,
6070    ,
6061    ,
6051    ,
6042    ,
6033    ,
6024    ,
6015    ,
6006    ,
5997    ,
5988    ,
5979    ,
5970    ,
5961    ,
5952    ,
5944    ,
5935    ,
5926    ,
5917    ,
5908    ,
5900    ,
5891    ,
5882    ,
5874    ,
5865    ,
5857    ,
5848    ,
5839    ,
5831    ,
5822    ,
5814    ,
5806    ,
5797    ,
5789    ,
5780    ,
5772    ,
5764    ,
5755    ,
5747    ,
5739    ,
5731    ,
5722    ,
5714    ,
5706    ,
5698    ,
5690    ,
5682    ,
5674    ,
5666    ,
5658    ,
5650    ,
5642    ,
5634    ,
5626    ,
5618    ,
5610    ,
5602    ,
5594    ,
5587    ,
5579    ,
5571    ,
5563    ,
5556    ,
5548    ,
5540    ,
5533    ,
5525    ,
5517    ,
5510    ,
```

```
5502    ,
5495    ,
5487    ,
5479    ,
5472    ,
5464    ,
5457    ,
5450    ,
5442    ,
5435    ,
5427    ,
5420    ,
5413    ,
5405    ,
5398    ,
5391    ,
5384    ,
5376    ,
5369    ,
5362    ,
5355    ,
5348    ,
5340
};

//***function definition****//
void enable_SAC02(void);
void enable_ADC01(void);
void Software_Trim(void);

//*** Variable definition ***//
/* FRAM Variable that stores ADC results*/
#define MCLK_FREQ_MHZ 8
#pragma PERSISTENT(ADC_Conversion_Result)
int ADC_Conversion_Result = 0;          // ADC conversion result
int ADC_Fork_Result[10];
int ADC_Sample_Count = 0;
unsigned int DAC_data=819;   // Vout = Vref*DAC_data/4096, 1.5V --> 2457, 1.8V --> 2949, 1V -->
1638, 0.6V --> 983, 0.5V --> 819(Clik)

int main(void)
{
    //Stop WDT
    WDT_A_hold(WDT_A_BASE);

    // SYS CLK to 8MHz
    __bis_SR_register(SCG0);                // disable FLL
    CSCTL3 |= SELREF__REFOCLK;              // Set REFO as FLL reference source
    CSCTL1 = DCOFTRIMEN_1 | DCOFTRIM0 | DCOFTRIM1 | DCORSEL_3;// DCOFTRIM=3, DCO Range = 8MHz
    CSCTL2 = FLLD_0 + 243;                  // DCODIV = 8MHz
    __delay_cycles(3);
    __bic_SR_register(SCG0);                // enable FLL
    Software_Trim();                        // Software Trim to get the best DCOFTRIM value


    CSCTL4 = SELMS__DCOCLKDIV | SELA__REFOCLK; // set default REFO(~32768Hz) as ACLK source, ACLK =
32768Hz
                                            // default DCODIV as MCLK and SMCLK source

    memset(peak, 0, sizeof(peak));
    memset(phase, 0, sizeof(phase));
    // Configure GPIO
    P1DIR |= BIT0;                                  // P1.0/ output
    P1OUT |= BIT0;                                  // P1.0 high
    P1DIR |= BIT6|BIT7;                     // P1.6 and P1.7 output
    P1SEL1 |= BIT6|BIT7;                    // P1.6 and P1.7 options select

    PM5CTL0 &= ~LOCKLPM5;                   // Disable the GPIO power-on default high-impedance mode
                                            // to activate previously configured port settings
    TB0CCTL0 |= CCIE;                               // TBCCR0 interrupt enabled
    TB0CCTL1 |= OUTMOD_7;                               // TBCCR0 interrupt enabled
//    TB0CCTL2 |= OUTMOD_3;                               // TBCCR0 interrupt enabled
    TB0CCTL2 |= OUTMOD_7;
    TB0CCR0 = 16000-1;
    TB0CCR1 = 8000-1;
    TB0CCR2 = 8000-1;
    TB0CTL = TBSSEL__SMCLK | MC__UP | TBCLR;            // SMCLK, UP mode

    //Configure OA2 functionality
```

```
    enable_SAC02();

    // Configure OA30 as DAC Buffer Mode
      P3SEL0 |= BIT5;                           // Select P3.5 as OA0O function
      P3SEL1 |= BIT5;                           // OA is used as buffer for DAC

      PM5CTL0 &= ~LOCKLPM5;                     // Disable the GPIO power-on default high-impedance
mode
                                                // to activate previously configured port settings

      // Configure reference module
      PMMCTL0_H = PMMPW_H;                      // Unlock the PMM registers
      PMMCTL2 = INTREFEN | REFVSEL_2;           // Enable internal 2.5V reference
      while(!(PMMCTL2 & REFGENRDY));            // Poll till internal reference settles

      SAC3DAC = DACSREF_1 + DACLSEL_0 + DACIE;  // Select int Vref as DAC reference, DAC latch
loads when DACDAT written
      SAC3DAT = DAC_data;                       // Initial DAC data
      SAC3DAC |= DACEN;                         // Enable DAC

      SAC3OA = NMUXEN + PMUXEN + PSEL_1 + NSEL_1;//Select positive and negative pin input
      SAC3OA |= OAPM;                           // Select low speed and low power mode
      SAC3PGA = MSEL_1;                         // Set OA as buffer mode
      SAC3OA |= SACEN + OAEN;                   // Enable SAC and OA

    // Configure OA0 as non-inverting  PGA mode
        P1SEL0 |= BIT1 + BIT2 + BIT3;            // Select P1.1 P1.2 P1.3 OA function
        P1SEL1 |= BIT1 + BIT2 + BIT3;            // Select P1.1 P1.2 P1.3 OA function

        SAC0OA = NMUXEN + PMUXEN + PSEL_0 + NSEL_1;//Select positive and negative pin input
        SAC0OA |= OAPM;                          // Select low speed and low power mode
        SAC0PGA = GAIN0 + MSEL_2;                // Set Non-inverting PGA mode with Gain=1
        SAC0OA |= SACEN + OAEN;                  // Enable SAC and OA

    // Enable ADC01 functionality
    enable_ADC01();

//      __bis_SR_register(LPM0_bits | GIE);          // Enter LPM0 w/ interrupt
    //LPM3, SleepTimer will force exit
    __bis_SR_register(LPM3_bits + GIE);
    __no_operation();                           // For debug

    //Enter LPM3
//      __bis_SR_register(LPM3_bits);
    //For debug
//      __no_operation();
/*
    while(1){

            //Initialize sleep timer
        //  initSleepTimer(1638);// 1638/32768 = ~50ms and (65535/32768) = ~2sec

            //LPM3, SleepTimer will force exit
        __bis_SR_register(LPM3_bits + GIE);

        }
*/
}

void Software_Trim()
{
    unsigned int oldDcoTap = 0xffff;
    unsigned int newDcoTap = 0xffff;
    unsigned int newDcoDelta = 0xffff;
    unsigned int bestDcoDelta = 0xffff;
    unsigned int csCtl0Copy = 0;
    unsigned int csCtl1Copy = 0;
    unsigned int csCtl0Read = 0;
    unsigned int csCtl1Read = 0;
    unsigned int dcoFreqTrim = 3;
    unsigned char endLoop = 0;

    do
    {
        CSCTL0 = 0x100;                         // DCO Tap = 256
        do
        {
            CSCTL7 &= ~DCOFFG;                  // Clear DCO fault flag
        }while (CSCTL7 & DCOFFG);               // Test DCO fault flag
```

```
        __delay_cycles((unsigned int)3000 * MCLK_FREQ_MHZ);// Wait FLL lock status (FLLUNLOCK) to
be stable
                                                           // Suggest to wait 24 cycles of divided
FLL reference clock
        while((CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)) && ((CSCTL7 & DCOFFG) == 0));

        csCtl0Read = CSCTL0;                    // Read CSCTL0
        csCtl1Read = CSCTL1;                    // Read CSCTL1

        oldDcoTap = newDcoTap;                  // Record DCOTAP value of last time
        newDcoTap = csCtl0Read & 0x01ff;        // Get DCOTAP value of this time
        dcoFreqTrim = (csCtl1Read & 0x0070)>>4;// Get DCOFTRIM value

        if(newDcoTap < 256)                     // DCOTAP < 256
        {
            newDcoDelta = 256 - newDcoTap;      // Delta value between DCPTAP and 256
            if((oldDcoTap != 0xffff) && (oldDcoTap >= 256)) // DCOTAP cross 256
                endLoop = 1;                    // Stop while loop
            else
            {
                dcoFreqTrim--;
                CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);
            }
        }
        else                                    // DCOTAP >= 256
        {
            newDcoDelta = newDcoTap - 256;      // Delta value between DCPTAP and 256
            if(oldDcoTap < 256)                 // DCOTAP cross 256
                endLoop = 1;                    // Stop while loop
            else
            {
                dcoFreqTrim++;
                CSCTL1 = (csCtl1Read & (~DCOFTRIM)) | (dcoFreqTrim<<4);
            }
        }

        if(newDcoDelta < bestDcoDelta)          // Record DCOTAP closest to 256
        {
            csCtl0Copy = csCtl0Read;
            csCtl1Copy = csCtl1Read;
            bestDcoDelta = newDcoDelta;
        }

    }while(endLoop == 0);                       // Poll until endLoop == 1

    CSCTL0 = csCtl0Copy;                        // Reload locked DCOTAP
    CSCTL1 = csCtl1Copy;                        // Reload locked DCOFTRIM
    while(CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1)); // Poll until FLL is locked
}

void enable_SAC02(void){

    //Configure OA2 functionality
        GPIO_setAsPeripheralModuleFunctionOutputPin(
                GPIO_PORT_SACOA2O,
                GPIO_PIN_SACOA2O,
                GPIO_FUNCTION_SACOA2O);
        GPIO_setAsPeripheralModuleFunctionInputPin(
                GPIO_PORT_SACOA2N,
                GPIO_PIN_SACOA2N,
                GPIO_FUNCTION_SACOA2N);
        GPIO_setAsPeripheralModuleFunctionInputPin(
                GPIO_PORT_SACOA2P,
                GPIO_PIN_SACOA2P,
                GPIO_FUNCTION_SACOA2P);

        /*
         * Disable the GPIO power-on default high-impedance mode
         * to activate previously configured port settings
         */
        PMM_unlockLPM5();

    //Select external source for both positive and negative inputs
       SAC_OA_init(
               SAC2_BASE,
               SAC_OA_POSITIVE_INPUT_SOURCE_EXTERNAL,
               SAC_OA_NEGATIVE_INPUT_SOURCE_EXTERNAL);
```

```
        //Select low speed and low power mode
        SAC_OA_selectPowerMode(
                SAC2_BASE,
                SAC_OA_POWER_MODE_LOW_SPEED_LOW_POWER);
        //Enable OA
        SAC_OA_enable(SAC2_BASE);
        //Enable SAC
        SAC_enable(SAC2_BASE);
}

void enable_ADC01(void){

    // Configure GPIO
        P6DIR |= BIT6;                                          // Set P6.6 to output direction
        P6OUT &= ~BIT6;                                         // Clear P6.6

        // Configure ADC A1 pin
        P1SEL0 |= BIT1;
        P1SEL1 |= BIT1;

        // Configure XT1 oscillator
        P2SEL1 |= BIT6 | BIT7;                                  // P2.6~P2.7: crystal pins

        // Disable the GPIO power-on default high-impedance mode to activate
        // previously configured port settings
        PM5CTL0 &= ~LOCKLPM5;

        CSCTL4 = SELA__XT1CLK;                                  // Set ACLK = XT1; MCLK = SMCLK =
DCO
        do
        {
            CSCTL7 &= ~(XT1OFFG | DCOFFG);                      // Clear XT1 and DCO fault flag
            SFRIFG1 &= ~OFIFG;
        }while (SFRIFG1 & OFIFG);                               // Test oscillator fault flag

        // Configure ADC
        ADCCTL0 |= ADCON | ADCMSC;                              // ADCON
        ADCCTL1 |= ADCSHS_2 | ADCCONSEQ_2;                      // repeat single channel; TB1.1
trig sample start
        ADCCTL2 &= ~ADCRES;                                     // clear ADCRES in ADCCTL
        ADCCTL2 |= ADCRES_2;                                    // 12-bit conversion results
        ADCMCTL0 |= ADCINCH_1 | ADCSREF_0;                      // A1 ADC input select; Vref=AVCC
        ADCIE |= ADCIE0;                                        // Enable ADC conv complete
interrupt
/*
        // Configure reference
        PMMCTL0_H = PMMPW_H;                                    // Unlock the PMM registers
        PMMCTL2 |= INTREFEN | REFVSEL_0;                        // Enable internal 1.5V reference
        __delay_cycles(400);                                    // Delay for reference settling
*/
        ADCCTL0 |= ADCENC;                                      // ADC Enable


}
static int sweep_idx=0;
static int convert=1;
static int i=0, u_d = 1;  //up=1, down =-1

// Timer0_B0 interrupt service routine
#pragma vector = TIMER0_B0_VECTOR
__interrupt void Timer0_B0_ISR (void)
{
  if ((u_d== 1) && (i==PRDS_PER_SWEEP)) {
      u_d=-5;
      P1OUT ^= BIT0;
      convert=0;
  }
  if ((u_d==-5) && (i< 0)) {
      i=0;
      u_d= 1;
      P1OUT ^= BIT0;
      if (++sweep_idx==32) {            // 32--> 128
          sweep_idx=0;
      }
      convert=1;
  }

  TB0CCR0 = TMRCCR[i]-1;
  TB0CCR1 = (TMRCCR[i]>>1)-1;
```

```
  TB0CCR2 = (TMRCCR[i]>>1)-1;

  // ADC conversion trigger signal - TimerB1.1 (32ms ON-period)
  TB1CCR0 = TMRCCR[i]-1;                                          // PWM Period
  TB1CCR1 = TMRCCR[i]-((TMRCCR[i]>>5)*sweep_idx)-1;           // TB1.1 ADC trigger, 5-->7
  TB1CCTL1 = OUTMOD_4;                                       // TB1CCR0 toggle
  TB1CTL = TBSSEL__SMCLK | MC_1 | TBCLR;                      // SMCLK, up mode

//  //****Time1.1 trigger ADC Sample****//
//  TB1CCR0 = TMRCCR[i]-1;                                          // PWM Period
//  TB1CCR1 = (TMRCCR[i]>>1)-1;                                     // TB1.1 ADC trigger

}

// Timer1_B1 interrupt service routine
#pragma vector = TIMER1_B1_VECTOR
__interrupt void Timer1_B1_ISR (void)
{

}

// ADC interrupt service routine
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=ADC_VECTOR
__interrupt void ADC_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(ADC_VECTOR))) ADC_ISR (void)
#else
#error Compiler not supported!
#endif
{
    int sample=0;

    switch(__even_in_range(ADCIV,ADCIV_ADCIFG))
    {
        case ADCIV_NONE:
            break;
        case ADCIV_ADCOVIFG:
            break;
        case ADCIV_ADCTOVIFG:
            break;
        case ADCIV_ADCHIIFG:
            break;
        case ADCIV_ADCLOIFG:
            break;
        case ADCIV_ADCINIFG:
            break;
        case ADCIV_ADCIFG:
            sample = ADCMEM0;
            if (convert == 1) {
                if (sample > peak[i]) {
                    peak[i]=sample;
                    phase[i]=sweep_idx;
                }
            }
            i += u_d;
            ADCIFG = 0;
            break;                                              // Clear CPUOFF bit from 0(SR)
        default:
            break;
    }
}
```

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265