Implementing an MSP430 Accelerometer-Based Data Acquisition System
Andreas Dannenberg

Agenda

- Accelerometer Basics
- Interfacing to the MSP430
- Lab: Accelerometer and MSP430 Setup
- Lab: Low-Power Anti-Theft Alarm
- Lab: Tilt Ball
MSP-EXP430F5438 Accelerometer Overview

- Two- or three-axis analog-output accelerometer are used
- Measures dynamic and static acceleration

### MSP-EXP430F5438 Accelerometer Specs

<table>
<thead>
<tr>
<th></th>
<th>ADXL322</th>
<th>ADXL330</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Axis</td>
<td>2 (X, Y)</td>
<td>3 (X, Y, Z)</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>420mV/g</td>
<td>300mV/g</td>
</tr>
<tr>
<td>Measurement Range</td>
<td>±2g</td>
<td>±3.6g</td>
</tr>
<tr>
<td>Current Consumption</td>
<td>450μA</td>
<td>320μA</td>
</tr>
<tr>
<td>Wakeup Time</td>
<td>20ms</td>
<td>1ms</td>
</tr>
</tbody>
</table>

Which one is used on your board?
Output Response vs. Orientation

Output voltages at $V_{CC} = 3V$

$X_{OUT} = 1.08V$
$Y_{OUT} = 1.50V$

$X_{OUT} = 1.55V$
$Y_{OUT} = 1.92V$

$X_{OUT} = 1.50V$
$Y_{OUT} = 1.08V$

$X_{OUT} = 1.92V$
$Y_{OUT} = 1.50V$

$X_{OUT} = 1.50V$
$Y_{OUT} = 1.50V$

Accelerometer Output Voltage Distribution

Example: ADXL322

Zero-g Output Performance @ $V_{CC} = 3V$
Agenda

- Accelerometer Basics
- Interfacing to the MSP430
- Lab: Accelerometer and MSP430 Setup
- Lab: Low-Power Anti-Theft Alarm
- Lab: Tilt Ball

MSP430F5438 – Accelerometer Connections

- Accelerometer is powered by the MSP430 – allowing on-demand operation
MSP430F5438 ADC12 Enhanced Features

- Low power modes
  - Selectable speed vs. power mode
  - References automatically shut down to conserve power
- High clock dividers for fast system clocks
- Low $I_{CC}$
  - 130$\mu$A for ADC
  - 100$\mu$A for $V_{REF}$

MSP430F5438 GPIO Features

- Programmable Reduced/Full drive strengths (PxDS)
- Internal programmable pull-up/down pin resistors on every port pin
Accelerometer Turn-On Behavior (PxDS = 0)

Example: ADXL322

ACC_PWR

VX or VY

Agenda

• Accelerometer Basics
• Interfacing to the MSP430
• Lab: Accelerometer and MSP430 Setup
  • Lab: Low-Power Anti-Theft Alarm
  • Lab: Tilt Ball
Lab 1: Goal

• Configure and use the MSP430 to periodically read out accelerometer sensor data using Timer_A and ADC12
• Calculate the physical g-force values from the ADC12 conversion results
• Display the X and Y g-force readings on the LCD in real-time every 100ms

Before we get started…let’s setup CCE!

• Open the TI CCE IDE
• Select “Import…” from the “File” menu
• Select “General/Existing Projects into Workspace”
• Select “Browse…” and navigate to the folder containing the lab / demo project folders
• The dialog box should now list three lab projects
• Select “Copy projects into workspace”
• Click on “Finish”
Lab 1: Port-Pin Configuration

// Power the accelerometer through P6.x
P6OUT __________;
P6DIR __________;

// Configure P6.x, P6.x, and P6.x as analog pins
P6SEL __________;

// Configure P6.x, P6.x, and P6.x as inputs
P6DIR __________;

• Check the interconnections in the schematic
• Power the accelerometer through a port pin
• Configure analog pins for peripheral function
• Note: Configure pins for 3-channel accelerometer

Lab 1: Timer_B Setup

// Use TBCLK = ACLK, clear TBR
TBCTL = ____________________;
// Set OUT1 on EQU1, reset on EQU0
TBCCTL1 = ____________________;
// Set period to 1/10Hz
TBCCR0 = ____________________;
// Set EQU1 event
TBCCR1 = ____________________;

• Use TBCLK = ACLK = LFXT1 = 32,768kHz
• Use capture/compare block TB1 in compare mode
• Generate a rising edge on Timer_B.OUT1 every 100ms (will be used to trigger A/D conversions)
Lab 1: ADC12 Setup 1/2

// Configure S&H time, enable multiple conversions,
// enable ADC12
ADC12CTL0 = __________________________________;

// Use Timer_B.OUT1 to trigger conversions,
// pulse mode, single sequence of channels
ADC12CTL1 = __________________________________;

// 12-bit mode, use signed output format
ADC12CTL2 = __________________________________;

• Configure the sample and hold time
• Use Timer_B.OUT1 to trigger the ADC12 start-of-conversion
• Use 12-bit mode
• Configure for signed output format

Lab 1: ADC12 Setup 2/2

// Setup a two-channel ADC12 conversion sequence
// Accelerometer X-channel
ADC12MCTL0 = ___________
;

// Accelerometer Y-channel, end-of-sequence
ADC12MCTL1 = ___________
;

// Enable interrupts on ADC12MEM1
ADC12IE = ______;

// Enable conversions
ADC12CTL0 |= ___________
;

• Setup and convert a sequence of two channels (X and Y)
• Use AVCC and AVSS as reference
• Enable interrupts on ADC12MEM1
• Enable conversions
Lab 1: ADC12 Interrupt Service Function

```c
#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void)
{
    ADCResultX = _________;  // Read out results, clear IFGs
    ADCResultY = _________;
    ADC12CTL0 __________;   // Toggle ADC12ENC
    ADC12CTL0 __________;
    __bic_SR_register_on_exit(LPM0_bits);
}
```

- Move conversion results to global variables
- Toggle the enable conversion bit to allow new SOC

Lab 1: Calculating the g-Force Values

```
// Calculate the g-force values assuming a VCC of 3V
// to use the accelerometer’s datasheet sensitivity factor.
// Since the output is ratiometric the actual VCC level
// doesn’t matter.

AccelX = ______________________ ADCResultX _____;
AccelY = ______________________ ADCResultY _____;
```

- Check the User’s Guide for details on how to interpret the ADC12 conversion data, and calculate the g-force values
- Start the code, and move the board around watching the g-force readings on the LCD.
- LED1 should toggle every 1s during operation
Lab 1: Port-Pin Configuration – Solution

// Power the accelerometer through P6.0
P6OUT |= 0x01;
P6DIR |= 0x01;

// Configure P6.1, P6.2, and P6.3 as analog pins
P6SEL |= 0x0e;

// Configure P6.1, P6.2, and P6.3 as inputs
P6DIR &= ~0x0e;

Lab 1: Timer_B Setup – Solution

// Use TBCLK = ACLK, clear TBR
TBCTL = TBSEL_1 + TBCLR;
// Set OUT1 on EQU1, reset on EQU0
TBCTL1 = OUTMOD_3;
// Set period to 1/10Hz
TBCCR0 = 32768 / 10;
// Set EQU1 event
TBCCR1 = TBCCR0 >> 1;

- Note that the value used for TBCCR1 is somewhat arbitrary however one needs to make sure it is within the count range of TBR
Lab 1: ADC12 Setup 1/2 – Solution

```c
// Configure S&H time, enable multiple conversions, enable ADC12
ADC12CTL0 = ADC12SHT0_6 + ADC12MSC + ADC12ON;
// Use Timer_B.OUT1 to trigger conversions, pulse mode, single sequence of channels
ADC12CTL1 = ADC12SHS_3 + ADC12SHP + ADC12CONSEQ_1;
// 12-bit mode, use signed output format
ADC12CTL2 = ADC12RES_2 + ADC12DF;
```

---

Lab 1: ADC12 Setup 2/2 – Solution

```c
// Setup a two-channel ADC12 conversion sequence
// Accelerometer X-channel
ADC12MCTL0 = ADC12INCH_1;
// Accelerometer Y-channel, end-of-sequence
ADC12MCTL1 = ADC12INCH_2 + ADC12EOS;
// Enable interrupts on ADC12MEM1
ADC12IE = 0x0002;
// Enable conversions
ADC12CTL0 |= ADC12ENC;
```
Lab 1: ADC12 ISR – Solution

```c
#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void)
{
    ADCResultX = ADC12MEM0;  // Read out results, clear IFGs
    ADCResultY = ADC12MEM1;

    ADC12CTL0 &= ~ADC12ENC;  // Toggle ADC12ENC
    ADC12CTL0 |= ADC12ENC;

    __bic_SR_register_on_exit(LPM0_bits);
}
```

- Toggling of ADC12ENC is needed to ready the ADC12 for the next sequence of channels to be converted when the next trigger occurs

Lab 1: Calculating the g-Force Values – Solution

```c
// Solution for ADXL322 (2-axis Accelerometer)
AccelX = 3.0f / 4096 / 0.42f * (ADCResultX >> 4);
AccelY = 3.0f / 4096 / 0.42f * (ADCResultY >> 4);

// Solution for ADXL330 (3-axis Accelerometer)
AccelX = 3.0f / 4096 / 0.30f * (ADCResultX >> 4);
AccelY = 3.0f / 4096 / 0.30f * (ADCResultY >> 4);
```

- Note that both flavors of accelerometers have a slightly different gain
Agenda

• Accelerometer Basics
• Interfacing to the MSP430
• Lab: Accelerometer and MSP430 Setup
• Lab: Low-Power Anti-Theft Alarm
• Lab: Tilt Ball

Lab 2: Goal

• Flash the LCD backlight and output a tone in case your MSP-EXP430F5438 gets moved
• Use low-power best practices to achieve lowest possible average current
  – Timer and interrupt-driven activity
  – On-demand accelerometer operation
  – Maximize time in low-power mode
Lab 2: Timer_B Setup

- Use TBCLK = ACLK, clear TBR, enable TBR
- Use TBCCR1 to trigger start of conversion, and to wait for the accelerometer to settle. How long is good?

```c
#pragma vector = TIMERB1_VECTOR
__interrupt void TIMERB1_ISR(void)
{
    switch (__even_in_range(TBIV, 0x0e))
    {
        case 0x0e: // TBIFG
            P6DIR ________; // Set ACC_PWR to output to power-up accelerometer
            break;
    }
}
```

Lab 2: Accelerometer On-Demand Operation

- Idea: Switch accelerometer power signal between “output high” and “input” to achieve on-demand operation

```c
void System_Init(void)
{
    // Init P6.x output latch to prepare powering the accelerometer via ACC_PWR
    P6OUT ________;
}
```

```c
#pragma vector = TIMERB1_VECTOR
__interrupt void TIMERB1_ISR(void)
{
    switch (__even_in_range(TBIV, 0x0e))
    {
        case 0x0e: // TBIFG
            P6DIR ________; // Set ACC_PWR to output to power-up accelerometer
            break;
    }
}
```
Lab 2: Low-Power Mode Handling 1/2

void main(void)
{
  (...) 
  while (1)
  {
    // Wait in low-power mode X, enable interrupts
    __bis_SR_register(______________);
  }
}

- Which low-power mode is most suitable and will result in the lowest possible stand-by current?
- The interrupts section in the User’s Guide has more info
- Enter selected low-power mode to wait until an ADC12 conversion has been completed

Lab 2: Low-Power Mode Handling 2/2

#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void)
{
  (...) 
  // Exit low-power mode
  __bic_SR_register_on_exit(__________);
}

- Exit the low-power mode upon completion of the ADC12 ISR
- Clearing the low-power mode bits on the top of the stack will wake up the system’s main() context – same as on all MSP430
Lab 2: Finishing Up…

while (1)
{
(...)
while (______________ || // Check limits to trigger
______________ || // alarm
______________)
{
(...)
}

- Determine and fill-in suitable X and Y channel boundaries
to set off alarm by moving the board and using the
debugger to inspect the ADC12 conversion results
- What is the board’s total current consumption measured
through jumper JP2?

ATC 2008

Lab 2: Timer_B Setup – Solution

// Use TBCLK = ACLK, clear TBR, enable TBR
// overflow interrupt
TBCTL = TBSSEL_1 + TBCLR + TBIE;
// Set 1s interval for overflow
TBCCR0 = 32768 - 1;
// Set OUT1 on EQU1, reset on EQU0
TBCCTL1 = OUTMOD_3;
// Set EQU1 event. Used as accelerometer power-on delay and
// ADC12 start of conversion trigger.
TBCCR1 = 655; // For ADXL322
// or
TBCCR1 = 33; // For ADXL330

- Note that both flavors of accelerometers have a different
settling time

ATC 2008
Lab 2: On-Demand Operation – Solution

```c
void System_Init(void)
{
    // Init P6.0 output latch to prepare powering
    // the accelerometer via ACC_PWR
    P6OUT |= 0x01;
}
```

```c
#pragma vector = TIMERB1_VECTOR
__interrupt void TIMERB1_ISR(void)
{
    switch (__even_in_range(TBIV, 0x0e))
    {
        case 0x0e: // TBIFG
            P6DIR |= 0x01; // Set ACC_PWR to output to
            // power-up accelerometer
            break;
    }
}
```

Lab 2: Low-Power Mode Handling 1/2 – Solution

```c
void main(void)
{
    (...) while (1)
    {
        // Wait in low-power mode 3, enable interrupts
        __bis_SR_register(LPM3_bits + GIE);
    }
```

- LPM3 is most suitable, since it leaves the 32,768kHz XTAL on while keeping the high-speed DCO and the CPU off.
Lab 2: Low-Power Mode Handling 2/2 – Solution

```c
#pragma vector = ADC12_VECTOR
interrupt void ADC12_ISR(void)
{
    (...)
    // Exit low-power mode
    __bic_SR_register_on_exit(LPM3_bits);
}
```

Lab 2: Finishing Up… – Solution

```c
while (1)
{
    (...)
    while (ADCResultX > 200 || ADCResultX < -200) // Check limits to trigger
    {
        (...)
    }
```

- A value of 200 was used since it allows for enough headroom to accommodate non-calibrated systems
- The total current consumption measured through jumper JP2 should be in the 3μA range while the system is in LPM3
Agenda

• Accelerometer Basics
• Interfacing to the MSP430
• Lab: Accelerometer and MSP430 Setup
• Lab: Low-Power Anti-Theft Alarm
• Lab: Tilt Ball

Lab 3: Goal

• Load and run the provided code on the MSP-EXP430F5438
• Tilting the board to X and Y will move around the ball on the LCD
• Set the board flat on the desk. What happens to the ball?
• Incorporate a calibration mechanism to establish a zero-g position
Lab 3: Implementing Push-Button Control

// Configure P2.x for 'S1' push-button operation

// Configure P2.x as input
P2DIR __________;

// Prepare P2.x pull-up resistor
P2OUT __________;

// Enable P2.x pull-up resistor
P2REN __________;

• Check where 'S1' is connected to
• Note that an internal pull-up must be used
• Configure GPIO accordingly
• The button will be checked inside the main while() loop

Lab 3: Adding Accelerometer Calibration

while (1)
{
    // Wait in low-power mode 0, enable interrupts
    __bis_SR_register(LPM0_bits + GIE);
    P1OUT ^= 0x01;  // Toggle LED1
    /* INSERT S1 BUTTON HANDLING HERE */
    /* APPLY CALIBRATION VALUES TO ADCResultX&Y HERE */
    dx = ADCResultX >> 8;  // Scale accelerometer
    dy = ADCResultY >> 8;  // readings

    // Add two variables to the code to hold calibration data
    // Use button S1 to capture the ADC12 results at the time of
    // button press and use as zero-g calibration data
Lab 3: Push-Button Control – Solution

// Configure P2.6 for 'S1' push-button operation

// Configure P2.6 as input
P2DIR &= ~0x40;

// Prepare P2.6 pull-up resistor
P2OUT |= 0x40;

// Enable P2.6 pull-up resistor
P2REN |= 0x40;

Lab 3: Accelerometer Calibration – Solution

int ADCResultXCal = 0; // Initialize cal values
int ADCResultYCal = 0;

(...)

while (1)
{
    if (!(P2IN & 0x40)) // Cal button pressed?
    {
        // If yes, store current values
        ADCResultXCal = ADCResultX;
        ADCResultYCal = ADCResultY;
    }

    // Apply calibration values
    ADCResultX -= ADCResultXCal;
    ADCResultY -= ADCResultYCal;
}
Thank you