CPE 325: Intro to Embedded Computer System

Project

Digital Level

Submitted by: Anshika Sinha

Date of Experiment: 04/11/2025

Report Deadline: 04/15/2025

Demonstration Deadline: 04/16/2025

Project Description

This project implements a digital level using the MSP430F5529 microcontroller and a 3-axis accelerometer (ADXL335). A digital level is an electronic tool designed to measure surface tilt angles with high accuracy. This project uses a 3-axis accelerometer to detect the device's orientation, calculates the tilt angle in real time, and displays the results using LED indicators. A calibration button allows the user to set a reference zero position for measurements. The angle values are transmitted to the UAH Serial App using RS-232 communication for monitoring and debugging.

This project begins by configuring Timer A to periodically trigger the accelerometer sampling at a fixed rate (10Hz). The timer generates interrupts at set intervals, ensuring real-time data collection without continuous polling, which optimizes power efficiency. On each interrupt, the system wakes from low-power mode and initiates analog to digital conversion of the analog outputs from the accelerometer using the ADC12 module, connected to pins P6.3 (X), P6.4 (Y), and P6.5 (Z) respectively. The ADC reads voltage values proportional to acceleration, which are converted to tilt angles using trigonometric calculations.

The X, Y, and Z values are read from the accelerometer to compute angles using the following equations:

$$\theta_X = arctan\left(\frac{X}{\sqrt{Y^2+Z^2}}\right) \cdot \frac{180}{\pi} \qquad \theta_Y = arctan\left(\frac{Y}{\sqrt{X^2+Z^2}}\right) \cdot \frac{180}{\pi}$$

A switch (P1.1) is configured with an interrupt to detect presses for calibration. When pressed, the system records the current accelerometer readings as the reference "zero" position, compensating for offsets. Debouncing is implemented to avoid false triggers. If the angles are under 5°, the green LED (P4.7) is on. If the angles are above this threshold but still below 10°, the red LED (P1.0) blinks. Otherwise, the device is not level and the red LED is on. The UART module, connected to pins P3.3 for TX and P3.4 RX, transmits the processed data to the UAH Serial App along with a start bit, where it is displayed on the graph.

This is useful for applications requiring precise angle detection, such as ensuring surfaces are perfectly horizontal/vertical in manufacturing, detecting orientation in robotics, monitoring vehicle inclination in automotive and aerospace, and alignment checks for equipment in machinery. It can be used for testing by hobbyists, engineers, and industrial applications in any orientation.

Theory

Topic 1: Timer A

a) Timer A is a 16-bit, highly configurable timer included in the MSP430F5529. Timer A is commonly used for timing operations, PWM generation, and input capture. It has multiple capture/compare registers that allow precise control over timing events. Timer_A can be clocked using the sources ACLK, SMCLK, and VLOCLK.

b) Timer A can operate in several modes, including Up Mode, Continuous Mode, and Up/Down Mode.

c) In Up Mode, the timer counts from zero to a specified value and resets, generating an interrupt at the end of each cycle.

d) In Continuous Mode, the timer counts from zero to its maximum value (0xFFFF) and rolls over, continuously generating interrupts. This mode is typically used for input capture applications, where the exact timing of an external event needs to be recorded.

e) Up/Down Mode allows the timer to count up to a specified value and then down to zero, useful for generating symmetrical PWM waveforms for motor control.

Topic 2: Interrupts

a) Interrupts allow a microcontroller to temporarily pause its main execution and handle specific events automatically. When an interrupt occurs, the processor jumps to a predefined Interrupt Service Routine (ISR) to handle the event. After execution, it resumes from where it left off using the RETI (Return from Interrupt) instruction.

b) To set up an interrupt for an I/O port, the following steps must be performed:

   a. Enable Global Interrupts: Set the GIE (General Interrupt Enable) bit in the status register (SR).

   b. Enable Specific Interrupts: Enable interrupts for the required port and pin

   c. Specify the Interrupt Edge: Define whether the interrupt triggers on a falling edge (high-to-low transition) or rising edge (low-to-high transition) using P1IES |= BIT1;.

   d. Clear Interrupt Flag: Ensure the interrupt flag is cleared at initialization

c) The Interrupt Vector Table (IVT) stores addresses of ISRs. The PORT1 Vector is ".int47" for the MSP430. Using interrupts allows the MSP430 to enter low-power mode until data is received, increasing efficiency.

Topic 3: Clock Module

a) The MSP430 microcontroller family provides flexible clocking options through its Unified Clock System (UCS). The UCS allows users to control processor and peripheral clock frequencies by configuring various clock sources and signals.

b) The MSP430F5529 uses a digitally controlled oscillator (DCO) to generate clock frequencies. The clock system consists of UCS (Unified Clock System), where UCSCTL1 controls the DCO range and UCSCTL2 sets the DCO multiplier.

c) Clock sources are used to generate three primary clock signals:
   1. ACLK (Auxiliary Clock)
      - Can be sourced from XT1CLK, REFOCLK, VLOCLK, or DCOCLK.
      - Often used for low-power peripherals like timers. This project uses ACLK.
   2. MCLK (Master Clock)
      - Drives the CPU and system.
      - Selectable from XT1CLK, REFOCLK, VLOCLK, DCOCLK, or XT2CLK.
   3. SMCLK (Subsystem Master Clock)

- Used for peripherals such as timers and communication modules.
- Configurable similar to MCLK.

Topic 4: Serial Communication and UART

a) Serial communication is a method of transferring data between two devices one bit at a time over a single communication line. The MSP430 microcontroller supports both synchronous and asynchronous communication.

b) This project focuses on asynchronous communication via UART (Universal Asynchronous Receiver-Transmitter). The system clock (1,048,576 Hz) is divided by the desired baud rate, with a modulation register adjusting for fractional values to minimize transmission errors.

c) The UAH Serial Port Application (SPA) is a Windows-based tool for plotting and logging data received via a PC's serial COM port. It is mainly used for visualizing and recording data from embedded systems connected through serial communication.

d) The UAH Serial App allows customizable data packet reception, supporting multiple channels, where each channel is plotted as a separate line. It supports various data types such as uint8, uint16, and uint32 for flexibility in data representation.

Topic 5: ADC12 & 3-axis Accelerometer

a) Analog to Digital Converters(ADCs) convert analog signals (e.g., voltage) into digital values for processing by microcontrollers or computers.
   i) The MSP430F5529 includes 16 configurable input channels, with 12 mapped to external analog pins and the rest to internal voltages and an on-chip temperature sensor.
   ii) The ADC12 has a 12-bit resolution, distinguishing between 4096 voltage levels (0 to 4095).
   iii) The ADC uses a timer trigger for periodic sampling. The sampling frequency should be at least twice the highest harmonic of the input signal to avoid aliasing.

b) An accelerometer is a sensor used to measure acceleration forces in one or more directions. The ADXL335 is a small, low-power, 3-axis accelerometer with analog voltage outputs for direct interfacing with ADCs. It has a minimum full-scale range of ±3 g, making it suitable for both static and dynamic acceleration measurements.

c) The ADXL335 offers customizable bandwidth through external capacitors connected to the XOUT, YOUT, and ZOUT pins, with bandwidth ranges of 0.5 Hz to 1600 Hz for the X and Y axes and 0.5 Hz to 550 Hz for the Z axis.

d) Internally, it uses a polysilicon surface-micromachined structure suspended by springs over a silicon wafer. When subjected to acceleration, the moving mass deflects, altering the capacitance between fixed and movable plates. This change is detected using phase-sensitive demodulation, generating an analog voltage proportional to the acceleration.

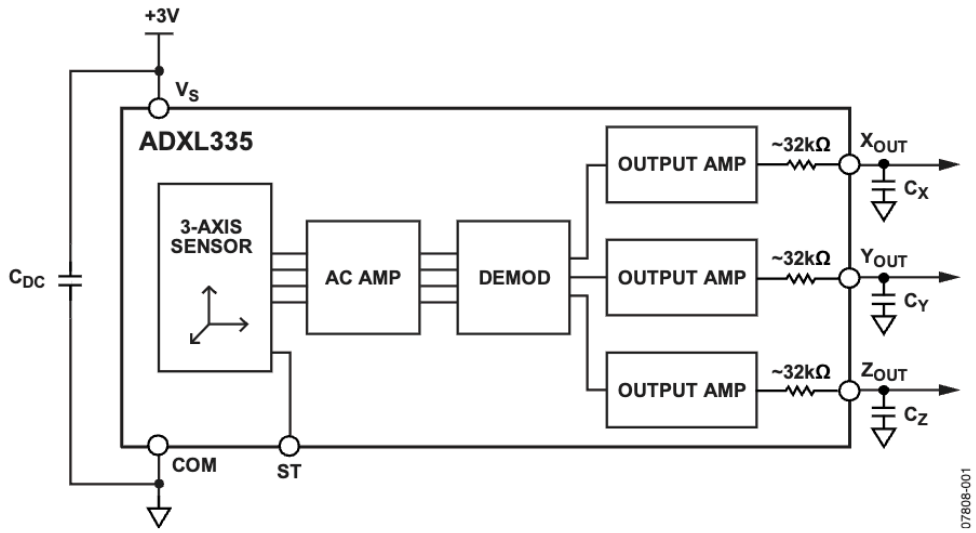

Figure 01: Block Diagram of the ADXL335 Accelerometer

Project Documentation

Set-up:

Hardware Connections: Connect the ADXL335 Accelerometer to the MSP430 LaunchPad.

- Vcc to 3.3 V
- GND to GND
- X to P6.3, Y to P6.4, Z to P6.5 respectively

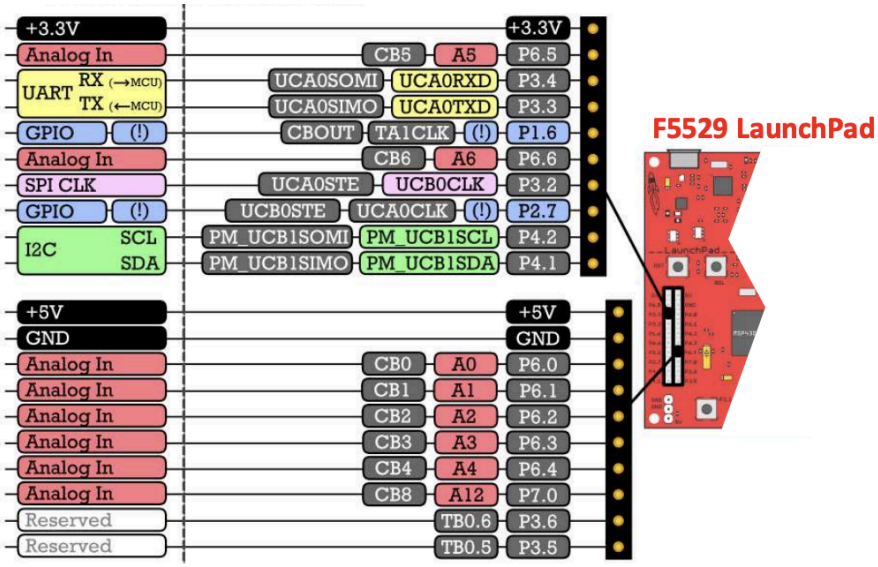

Figure 02: MSP430F5529 LP Pinout

Next, connect the UART cable to pins P3.3 for TX, P3.4 RX, and GND.

Serial App Setup:

Set the following settings in the UAH Serial App: Set the packet size to 9 bytes, with two channels of type Single 32bit, starting at position 1 and 5 respectively. Make sure to enable show on graph.

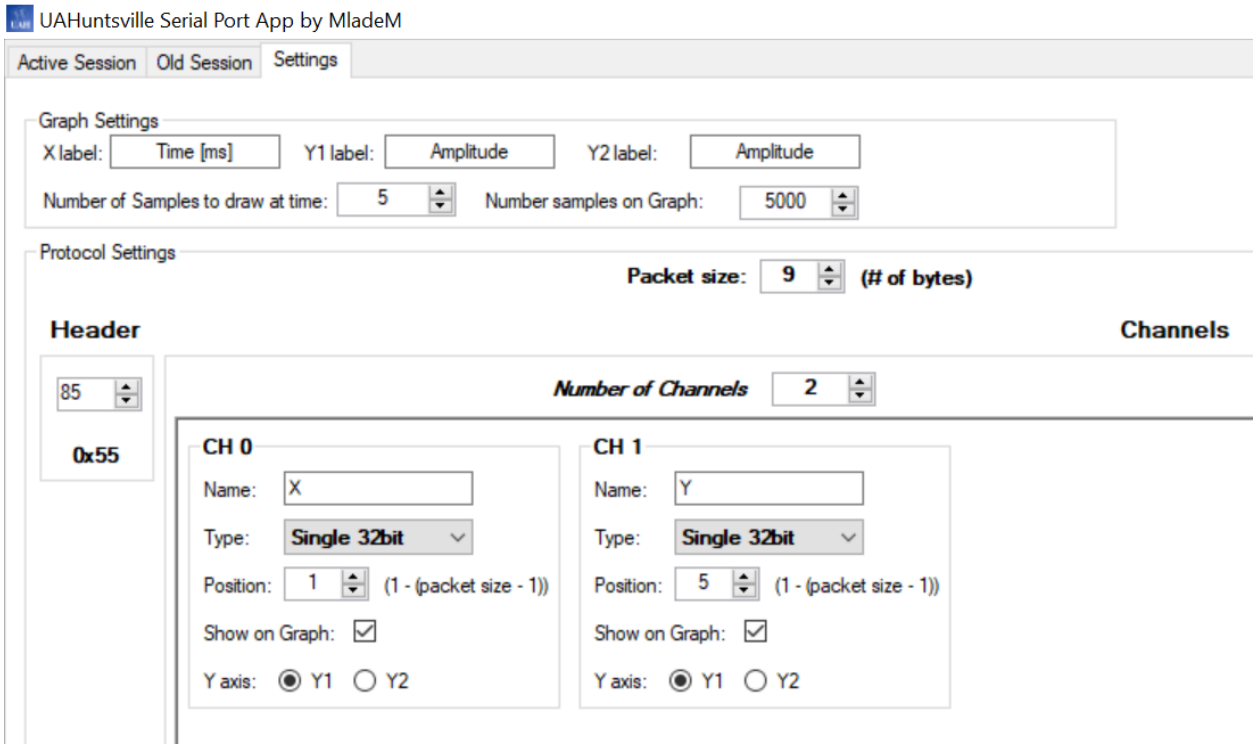

Figure 03: UAH Serial App Settings

Now, connect to the serial COM port and set the Baud Rate to 38400 with one stop bit. Make sure to enable chart and start connection.

Figure 04: UAH Serial App COM Setup

Now, build and flash the program (refer to Appendix) to the microcontroller using Code Composer Studio (CCS).

Program Flowchart:

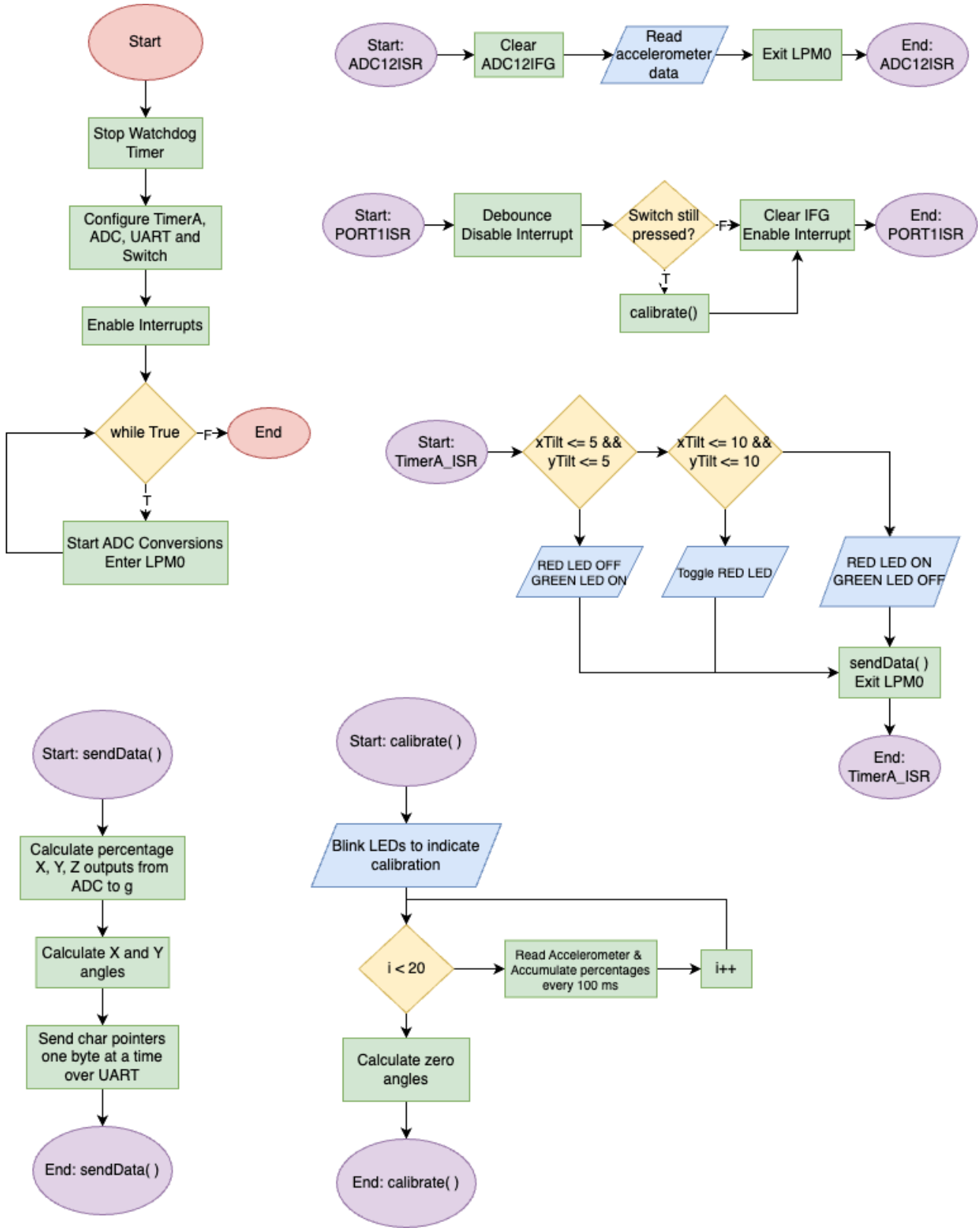

Figure 05: Program Flowchart

Operation of project:

Once serial communication with the UAH Serial App is established, run the program. The X and Y angles will be displayed on the graph in real time. Press the Switch at P1.1 to calibrate the device to its current position. Values will change on the graph accordingly. Monitor the green and red LEDs to check whether current angles are under the threshold.

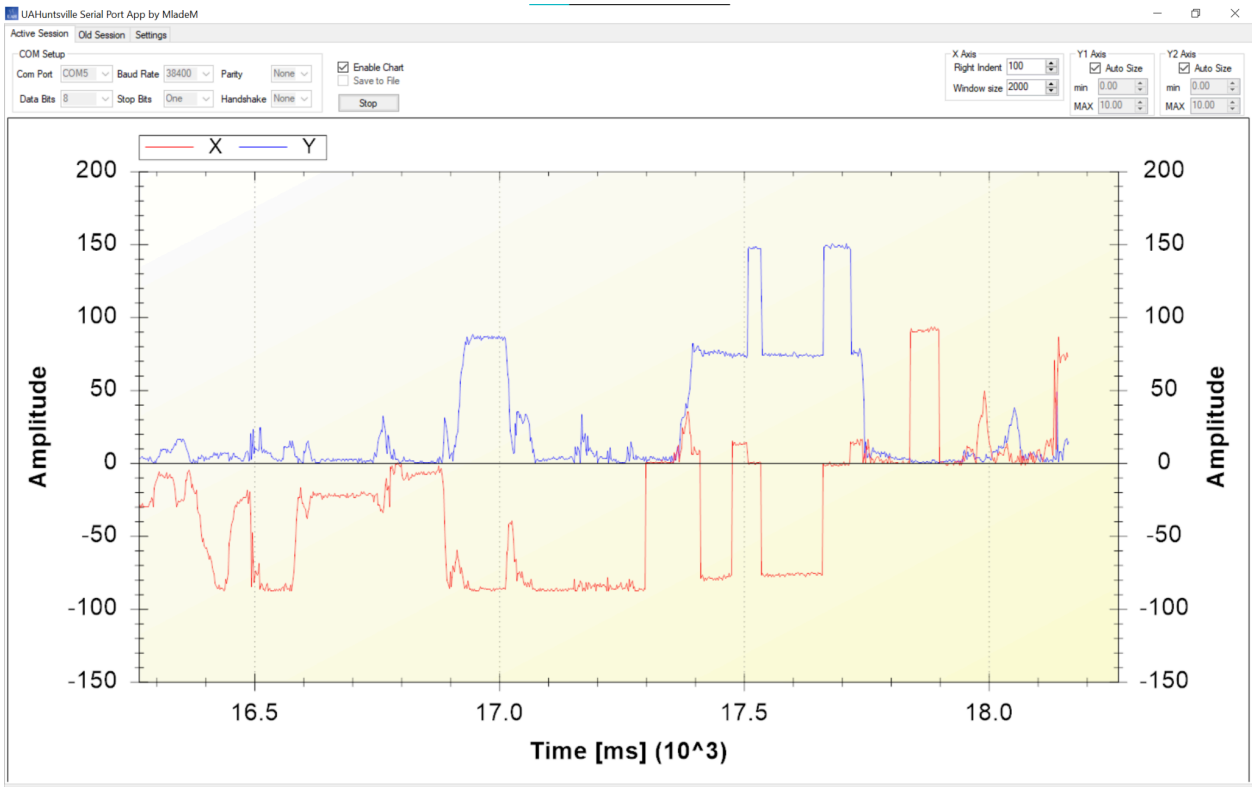

Figure 06: Program Output on UAH Serial App

Conclusion

This project creates a low-cost digital level with real-time feedback and calibration capabilities. The peripherals used are: Timer A for periodic sampling of accelerometer data, port I/O for LEDs and interrupt for switch, and the ADC12 for reading accelerometer data. The external components used are: a switch to set reference point, LEDs - GREEN LED (P4.7) for tilt < 5° and RED LED (P2.1) for tilt > 5°, and a 3-axis accelerometer(ADXL335) to measure orientation in X, Y, Z directions. Values recorded might not be perfectly precise due to noise and hardware limitations in analog conversions. This project can be expanded to include further functionality such as advanced filtering, a custom GUI, or bluetooth communication.

Appendix

Table 01: Program source code

```
/*-------------------------------------------------------------------
* File:        main.c
* Function:    Digital Level
* Description: This project implements a digital level using the MSP430F5529
           microcontroller and a 3-axis accelerometer (ADXL335). It measures
tilt angles in real-time and provides visual and serial output feedback.
* Input:       Accelerometer signal
* Output:      LEDs and serial output on UAH Serial App
* Author:      Anshika Sinha
* Date:        04.15.2025
*-------------------------------------------------------------------*/
// ---------- Include libraries -----------------------------------------
#include "intrinsics.h"
#include "msp430f5529.h"
#include <msp430.h>
#include <math.h>

// ---------- Definitions and global variables --------------------------
#define ADC_TO_G(n) ((3.0 * n / 4095 - 1.5) / 0.3) //convert to g

volatile long int ADCXval, ADCYval, ADCZval;
volatile float Xper, Yper, Zper, xTilt, yTilt;
volatile float xZero, yZero, zZero, baseX, baseY;

// ----------- Function prototypes ---------------------------------------
void TimerA_setup(void);
void ADC_setup(void);
void UART_setup(void);
void UART_putCharacter(char c);
void Switch_setup(void);
void readADC(void);
void sendData(void);
void calibrate(void);

//*******************************************************************
```

```
// MAIN
//*********************************************************************vo
id main (void)
{
   WDTCTL = WDTPW + WDTHOLD;             // Stop WDT

   TimerA_setup();                       // Setup timer to send ADC data
   ADC_setup();                          // Setup ADC
   UART_setup();                         // Setup UART for RS-232
   Switch_setup();                       // Setup switch for calibration
   _EINT();                              // Enable interrupts

   while (1){
       ADC12CTL0 |= ADC12SC;                // Start conversions
       __bis_SR_register(LPM0_bits + GIE); // Enter LPM0
   }
}
// ----------- Setup Timer A ---------------------------------------------
void TimerA_setup(void) {
   TA0CCTL0 = CCIE;                      // Enabled interrupt
   TA0CCR0 = 3276;                       // 3277 / 32768 = .1s for ACLK
   TA0CTL = TASSEL_1 + MC_1;             // ACLK, up mode
   P1DIR |= BIT0;                        // Setup LED1 at P2.1
   P4DIR |= BIT7;                        // Setup LED2 at P4.7
}
// ----------- Setup Accelerometer ADC convertor ---------------------------
void ADC_setup(void) {
   int i =0;
   P6DIR &= ~(BIT3 + BIT4 + BIT5);          // Configure P6.3, P6.4, P6.5 as
input pins
   P6SEL |= BIT3 + BIT4 + BIT5;             // Configure P6.3, P6.4, P6.5 as
analog pins
   // configure ADC converter
   ADC12CTL0 = ADC12ON + ADC12SHT0_6 + ADC12MSC;
   ADC12CTL1 = ADC12SHP + ADC12CONSEQ_3;    // Use sample timer, single
sequence
   ADC12MCTL0 = ADC12INCH_3;                // ADC A3 pin -X-axis
   ADC12MCTL1 = ADC12INCH_4;                // ADC A4 pin - Y-axis
```

```
    ADC12MCTL2 = ADC12INCH_5 + ADC12EOS;    // ADC A5 pin - Z-axis

                                        // EOS - End of Sequence for Conversions
    ADC12IE = ADC12IE0;                 // Enable ADC12IFG.1
    for (i = 0; i < 0x3600; i++);       // Delay for reference start-up
    ADC12CTL0 |= ADC12ENC;              // Enable conversions
}

// ----------- Setup UART at 38400 Baud rate -------------------------------
void UART_setup(void) {
    P3SEL |= BIT3 + BIT4;               // Set up Rx and Tx bits
    UCA0CTL0 = 0;                       // Set up default RS-232 protocol
    UCA0CTL1 |= BIT0 + UCSSEL_2;        // Disable device, set clock
    UCA0BR0 = 27;                       // 1048576 Hz / 38400
    UCA0BR1 = 0;
    UCA0MCTL = 0x94;
    UCA0CTL1 &= ~BIT0;                  // Start UART device
}
// ----------- Send message through UART -----------------------------------
void UART_putCharacter(char c) {
    while (!(UCA0IFG&UCTXIFG));    // Wait for previous character to transmit
    UCA0TXBUF = c;                 // Put character into tx buffer
}
// ----------- Setup switch for calibration --------------------------------
void Switch_setup(void) {
    P1DIR &= ~BIT1;                 // Set P1.1 as input for Switch input
    P1REN |= BIT1;                  // Enable pull-up register at P1.1
    P1OUT |= BIT1;

    P1IE |= BIT1;                   // Enable interrupt for Switch (P1.1)
    P1IES |= BIT1;                  // Set interrupt from high to low
    P1IFG &= ~BIT1;                 // Clear interrupt flag
}
// ----------- Read ADC values from accelerometer --------------------------
void readADC(void) {
    ADCXval = ADC12MEM0;            // Move results to x, y, z respectively
    ADCYval = ADC12MEM1;
    ADCZval = ADC12MEM2;
```

```
}

// - Converts ADC values, calculates tilt angles, and transmits characters -
void sendData(void) {
   unsigned int i;

   // Get samples from ADC
   Xper = (ADC_TO_G(ADCXval));    // Calculate percentage outputs
   Yper = (ADC_TO_G(ADCYval));    // Calculate percentage outputs
   Zper = (ADC_TO_G(ADCZval));    // Calculate percentage outputs

   // Calculate angles using arctangent formulas
   xTilt = abs(atan2f(Xper, sqrtf(Yper*Yper + Zper*Zper)) * (180.0f / M_PI)) -
baseX;
   yTilt = abs(atan2f(Yper, sqrtf(Xper*Xper + Zper*Zper)) * (180.0f / M_PI)) -
baseY;

   // Use character pointers to send one byte at a time
   char *xpointer=(char *)&xTilt;
   char *ypointer=(char *)&yTilt;

   UART_putCharacter(0x55);            // Send header
   for(i = 0; i < 4; i++)
   {
       UART_putCharacter(xpointer[i]); // Send x percentage - one byte at a
time
   }
   for(i = 0; i < 4; i++)
   {
       UART_putCharacter(ypointer[i]);// Send y percentage - one byte at a
time
   }
}
// ----------- Calibrate to current position ------------------------------
void calibrate(void) {
   float xSum, ySum, zSum;
   unsigned int i = 0;
```

```
    // Blink LEDs to indicate calibration
    P1OUT ^= BIT0;                    // Toggle RED LED
    P4OUT ^= BIT7;                    // Toggle GREEN LED
    __delay_cycles(32768);             // 1 s delay

    // Get samples from ADC
    for (i = 0; i < 20; i++) {
        readADC();
        xSum += (ADC_TO_G(ADCXval));     // Calculate percentage outputs
        ySum += (ADC_TO_G(ADCYval));     // Calculate percentage outputs
        zSum += (ADC_TO_G(ADCZval));     // Calculate percentage outputs
        __delay_cycles(3276);            // 100ms between samples
    }

    // Set zero positions
    xZero = xSum / 20;
    yZero = ySum / 20;
    zZero = zSum / 20;

    baseX = atan2f(xZero, sqrtf(yZero*yZero + zZero*zZero)) * (180.0f / M_PI);
    baseY = atan2f(yZero, sqrtf(xZero*xZero + zZero*zZero)) * (180.0f / M_PI);

    // Turn off LEDs
    P1OUT &= ~BIT0;
    P4OUT &= ~BIT7;
}

//*********************************************************************
// INTERRUPTS
//*********************************************************************
#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR(void) {
    ADC12IFG = 0x00;                  // Clear IFG
    readADC();
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

#pragma vector = TIMER0_A0_VECTOR
```

```
__interrupt void timerA_isr() {
   if (xTilt <= 5 && yTilt <= 5)
   {
       P1OUT &= ~BIT0;         // Turn off RED LED
       P4OUT |= BIT7;          // Turn on GREEN LED

   }
   else if (xTilt <= 10 && yTilt <= 10) // If tilt > 5 && <= 10
   {
       P1OUT ^= BIT0;          // Toggle RED LED
   }
   else                         // Tilt > 10
   {
       P1OUT |= BIT0;          // Turn on RED LED
       P4OUT &= ~BIT7;         // Turn off GREEN lED
   }
   sendData();                              // Send data to serial app
   __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

#pragma vector = PORT1_VECTOR
__interrupt void Port1_ISR(void) {
   __delay_cycles(40000);      // Debounce
   P1IE &= ~BIT1;              // Disable interrupt

   if ((P1IN&BIT1) == 0){      // If switch is still pressed
       calibrate();            // Start calibration
   }
   P1IFG &= ~BIT1;             // Clear interrupt flag register
   P1IE |= BIT1;               // Enable interrupt
}
```

Table 02: External connections