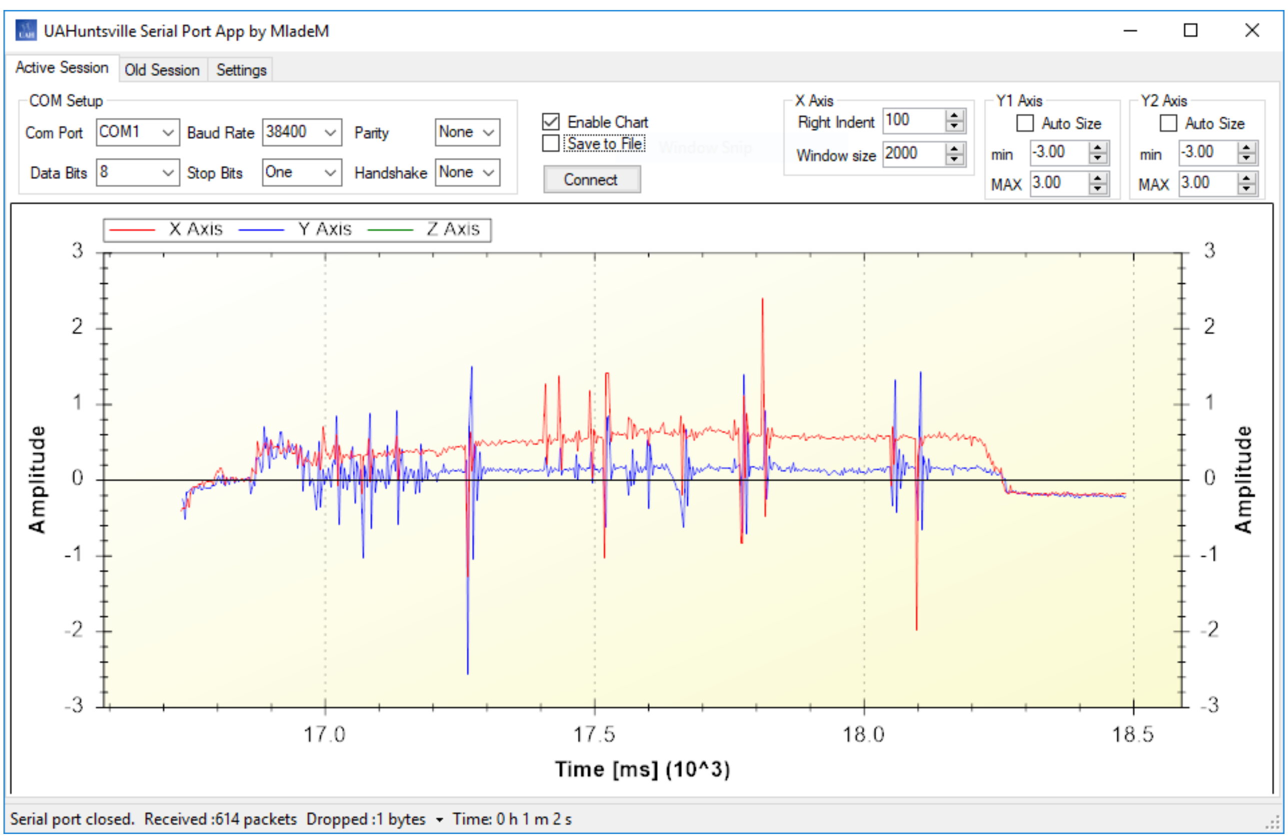
Austin Bumbalough
CPE 325-08
Lab 10
10/31/2019

Lab 10 Solution

Part 1 Solution

In Lab 10 Part 1, I interfaced a three-axis accelerometer with the ADC12 peripheral of the MSP430FG4618. I then converted the ADC values to acceleration and sent them over UART to the UAH Serial App for plotting the three values against time.

Output Screenshots



Source Code

```
void ADC_Setup(void);

void TimerB_Setup(void);

void UART_PutChar(char);

void UART_Setup(void);

volatile unsigned int ADCX, ADCY, ADCZ;

void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    TimerB_Setup(); // Initialize Timer B
    ADC_Setup(); // Initialize ADC
    UART_Setup();
    _EINT();

    while(1) {
        ADC12CTL0 |= ADC12SC;
        _BIS_SR(LPM0_bits + GIE);
    }
}

void ADC_Setup(void) {
    P6DIR &= ~(BIT2+BIT1+BIT0);             // Configure P6.0, P6.1, and P6.2
    P6SEL |= BIT2+BIT1+BIT0;                // Configure P6.0, P6.1, and P6.2
    ADC12CTL0 = ADC12ON + SHT0_6 + MSC;     // Enable conversion, 128 clk. cyc
    ADC12CTL1 = SHP + CONSEQ_1;             // Use sample timer, single sequer
    ADC12MCTL0 = INCH_0;                    // ADC A0 pin - Acc. X-axis
    ADC12MCTL1 = INCH_1;                    // ADC A1 pin - Acc. Y-axis
    ADC12MCTL2 = INCH_2 + EOS;              // ADC A2 pin - Acc. Z-axis, EOS -
    ADC12IE |= BIT2;                        // Enable ADC12IFG.2, interrupt tr
    ADC12CTL0 |= ENC;                       // Enable conversions
}

void sendData(void) {
    volatile float x_acc, y_acc, z_acc;
    unsigned int i;

    x_acc = ((ADCX / 4095.0 * 10.0) - 5);     // Calculate output x acceleratic
```

```
    y_acc = ((ADCY / 4095.0 * 10.0) - 5);    // Calculate output y acceleratic
    z_acc = ((ADCZ / 4095.0 * 10.0) - 5);    // Calculate output z acceleratic

    // Use character pointers to send one byte at a time
    char *x_acc_pnt = (char *)&x_acc;
    char *y_acc_pnt = (char *)&y_acc;
    char *z_acc_pnt = (char *)&z_acc;

    UART_PutChar(0x55);             // Send header
    for(i = 0; i < 4; i++) {            // Send x acceleration - one byte at a
        UART_PutChar(x_acc_pnt[i]);
    }

    for(i = 0; i < 4; i++) {            // Send y acceleration - one byte at a
        UART_PutChar(y_acc_pnt[i]);
    }

    for(i = 0; i < 4; i++) {            // Send z acceleration - one byte at a
        UART_PutChar(z_acc_pnt[i]);
    }
}

void TimerB_Setup(void) {
    TB0CCR0 = 3277; // 5 Hz
    TB0CTL |= TBSSEL_1 + MC_1; // ACLK source, up mode
    TB0CCTL0 = CCIE; // Enable interrupts
}

void UART_PutChar(char c) {
    while(!(IFG2 & UCA0TXIFG));         // Wait for previous character to be s
    UCA0TXBUF = c;                      // Send byte to the buffer for transmi
}

void UART_Setup(void) {
    P2SEL |= BIT4 + BIT5;               // 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
}

#pragma vector = ADC12_VECTOR
```

```
__interrupt void ADC12ISR(void) {
    ADCX = ADC12MEM0;                  // Move results, IFG is cleared
    ADCY = ADC12MEM1;
    ADCZ = ADC12MEM2;
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

#pragma vector = TIMERB0_VECTOR
__interrupt void TimerB_ISR(void) {
    sendData();
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
```

Part 2 Solution

For part two I implemented a car crash detector. I used the acceleration from each axis to calculate the magnitude of the net acceleration. When the magnitude exceeds 2g, LED1 turns on, indicating that a car crash has occurred.

Source Code

```
void ADC_Setup(void);

void TimerB_Setup(void);

void UART_PutChar(char);

void UART_Setup(void);

volatile unsigned int ADCX, ADCY, ADCZ;

volatile unsigned int crashDetected = 0;

volatile float X_ACC, Y_ACC, Z_ACC;

void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    P2DIR |= BIT2; // Configure P2.2 as output
```

```c
    P1DIR &= ~BIT0; // Configure P1.0 as input
    P1IES |= BIT0; // Select falling edge for interrupt trigger
    P1IE |= BIT0; // Enable interrupts for P1.0 (SW1)
    TimerB_Setup(); // Initialize Timer B
    ADC_Setup(); // Initialize ADC
    UART_Setup();
    _EINT();

    while(1) {
        ADC12CTL0 |= ADC12SC;
        _BIS_SR(LPM0_bits + GIE);
        if (crashDetected) P2OUT |= BIT2;
    }
}

void ADC_Setup(void) {
    P6DIR &= ~(BIT2+BIT1+BIT0);             // Configure P6.0, P6.1, and P6.2
    P6SEL |= BIT2+BIT1+BIT0;                // Configure P6.0, P6.1, and P6.2
    ADC12CTL0 = ADC12ON + SHT0_6 + MSC;     // Enable conversion, 128 clk. cyc
    ADC12CTL1 = SHP + CONSEQ_1;             // Use sample timer, single sequer
    ADC12MCTL0 = INCH_0;                    // ADC A0 pin - Acc. X-axis
    ADC12MCTL1 = INCH_1;                    // ADC A1 pin - Acc. Y-axis
    ADC12MCTL2 = INCH_2 + EOS;              // ADC A2 pin - Acc. Z-axis, EOS -
    ADC12IE |= BIT2;                        // Enable ADC12IFG.2, interrupt tr
    ADC12CTL0 |= ENC;                       // Enable conversions
}

void sendData(void) {
    unsigned int i;

    X_ACC = ((ADCX / 4095.0 * 10.0) - 5);    // Calculate output x acceleratic
    Y_ACC = ((ADCY / 4095.0 * 10.0) - 5);    // Calculate output y acceleratic
    Z_ACC = ((ADCZ / 4095.0 * 10.0) - 5);    // Calculate output z acceleratic

    // Use character pointers to send one byte at a time
    char *X_ACC_pnt = (char *)&X_ACC;
    char *Y_ACC_pnt = (char *)&Y_ACC;
    char *Z_ACC_pnt = (char *)&Z_ACC;

    UART_PutChar(0x55);             // Send header
    for(i = 0;i<4;i++) {             // Send x acceleration - one byte at a tim
        UART_PutChar(X_ACC_pnt[i]);
    }
```

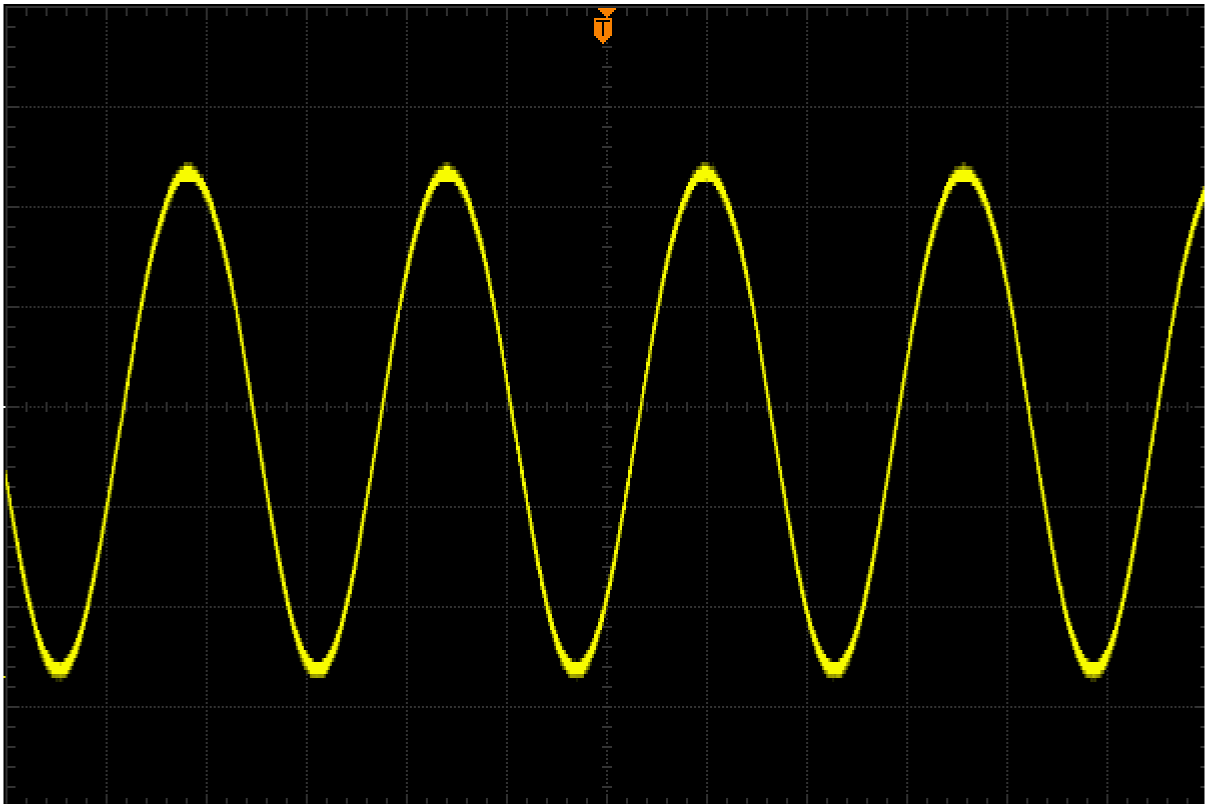
```
    for(i = 0;i<4;i++) {            // Send y acceleration - one byte at a tim
        UART_PutChar(Y_ACC_pnt[i]);
    }

    for(i = 0;i<4;i++) {            // Send z acceleration - one byte at a tim
        UART_PutChar(Z_ACC_pnt[i]);
    }
}

void TimerB_Setup(void) {
    TB0CCR0 = 3277; // 5 Hz
    TB0CTL |= TBSSEL_1 + MC_1; // ACLK source, up mode
    TB0CCTL0 = CCIE; // Enable interrupts
}

void UART_PutChar(char c) {
    while(!(IFG2 & UCA0TXIFG));         // Wait for previous character to be s
    UCA0TXBUF = c;                      // Send byte to the buffer for transmi
}

void UART_Setup(void) {
    P2SEL |= BIT4 + BIT5;               // 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
}

#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR(void) {
    ADCX = ADC12MEM0;                  // Move results, IFG is cleared
    ADCY = ADC12MEM1;
    ADCZ = ADC12MEM2;
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

#pragma vector=PORT1_VECTOR
__interrupt void P1_ISR(void) {
    for (unsigned long int i=20972;i>0;i--); // 20 ms debounce delay
    if ((P1IN & BIT0) == 0) {
        P2OUT &= ~BIT2; // Turn off led
        crashDetected = 0; // reset flag
```

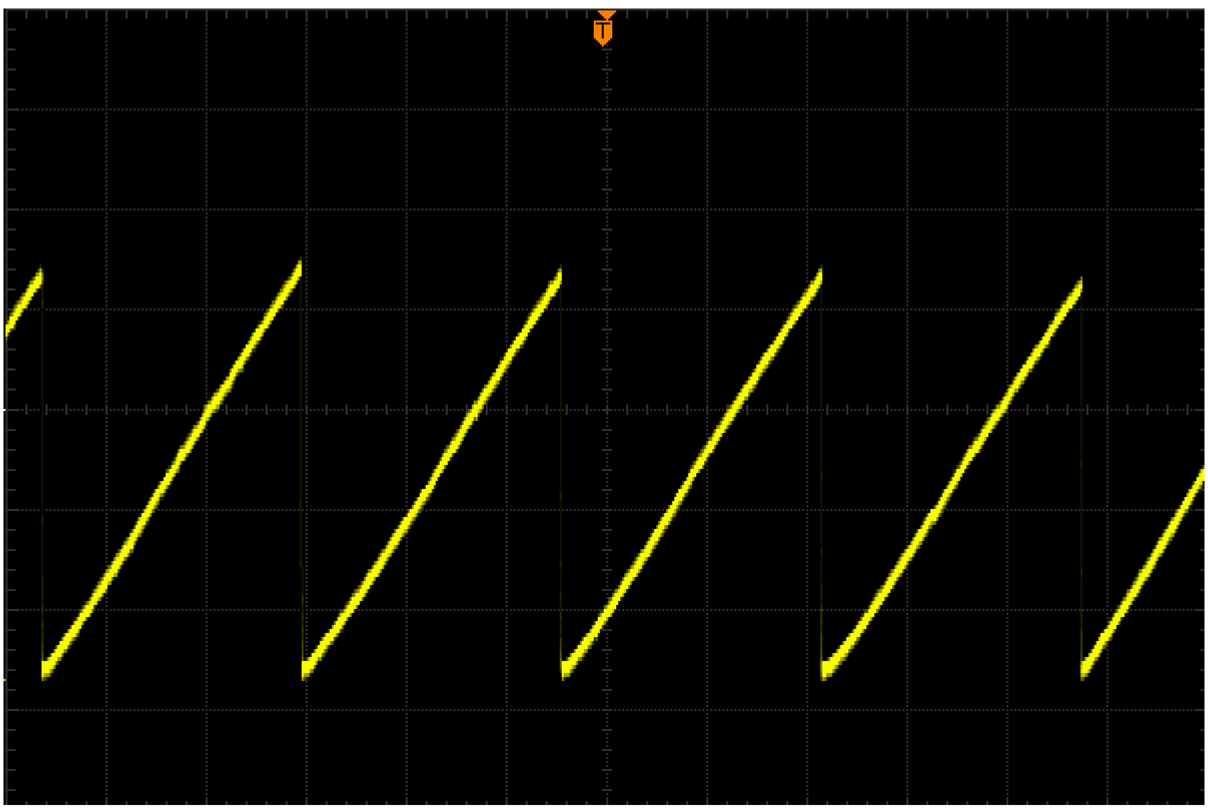
```
    }
    P1IFG &= ~BIT0;
}

#pragma vector = TIMERB0_VECTOR
__interrupt void TimerB_ISR(void) {
    sendData();
    if (crashDetected == 0) {
        float net_acc = sqrt( pow(X_ACC,2) + pow(Y_ACC,2)); // + pow(Z_ACC,2)
        if (net_acc > 2) crashDetected = 1; // Set flag if net acceleration ex
    }
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
```

Part 3 Solution

In part three, I used a lookup table of DAC input values to generate either a sine wave or a saw wave, selected by SW1.

Oscilloscope Output





Source Code

```
unsigned int HALF = 1;

unsigned int WAVE_SEL = 0;
```

```c
int* WAVE_TABLE[2] = {sine_512_lut,saw_512_lut};

void DAC12_Setup(void);

void TimerA_Setup(void);

void main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    P1DIR &= ~(BIT1+BIT0); // Configure P1.0 (SW1) and P1.1 (SW2) as inputs
    P1IES |= BIT1+BIT0; // Trigger interrupt on falling edge
    P1IE |= BIT1+BIT0; // Enable interrupts for P1.0 and P1.1
    TimerA_Setup();
    DAC12_Setup();
    _EINT();

    unsigned int j = 0;
    while(1) {
        _BIS_SR(LPM0_bits + GIE);
        DAC12_0DAT = (WAVE_TABLE[WAVE_SEL][j++ % 512]) / HALF;
    }
}

void DAC12_Setup(void) {
    ADC12CTL0 = REF2_5V + REFON;
    for (unsigned int i = 50000;i>0;i--); // Delay to allow Vref to stabilize
    DAC12_0CTL = DAC12IR + DAC12AMP_5 + DAC12ENC;
}

void TimerA_Setup(void) {
    TA0CTL = TASSEL_2 + MC_1;
    TA0CCR0 = 41; // 25 Hz signal
    TA0CCTL0 = CCIE;
}

#pragma vector=PORT1_VECTOR
__interrupt void P1_ISR(void) {

    if (P1IFG & BIT0) {
        for (unsigned int i = 20952;i>0;i--); // 20 ms debounce using SMCLK =
        if (((P1IN & BIT0) == 0) & (WAVE_SEL == 0)) { // Check if switch is st
            WAVE_SEL = 1; // Toggle between sine and saw wave
            P1IES &= ~BIT0; // Toggle interrupt edge select to catch switch re
        } else if (((P1IN & BIT0) == 1) & (WAVE_SEL == 1)) {
```

```
            WAVE_SEL = 0;
            P1IES |= BIT0;
        }
        P1IFG &= ~BIT0; // Clear IFG
    }

    if (P1IFG & BIT1) {
        for (unsigned int i = 20952;i>0;i--); // 20 ms debounce using SMCLK =
        if (((P1IN & BIT0) == 0) & ((P1IES & BIT1) == 1)) { // Check if switch
            HALF = 2;
            P1IES &= ~BIT1; // Toggle interrupt edge select to catch switch re
        }

        if (((P1IN & BIT0) == 1) & ((P1IES & BIT1) == 0)) {
            HALF = 1;
            P1IES |= BIT1;
        }
        P1IFG &= ~BIT1; // Clear IFG
    }
}

#pragma vector=TIMERA0_VECTOR
__interrupt void TimerA0_ISR(void) {
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
```