Demonstration: (NOTE: these change based on the lab - enter the demo tasks specified in each lab)

1. Demonstrate running digital clock and blinking of the LED (Requirements 1, 2, and 3)

2. Demonstrate changing the time using push buttons (Requirements 4, 5, 6, and 7)

3. Demonstrate keeping the board idle to clear the display and making a rotation to show the clock display again (Requirements 8 and 9)

Requirements:

1. The LCD should display a running digital clock aligned to the center of the screen with hours, minutes, seconds, and AM/PM
2. The clock display format should be of the format HH:MM:SS XX for showing two digit hours and H:MM:SS XX for showing one digit hour. XX denotes AM or PM
3. The green LED (J4.38) should do a quick blink (fast turn on and then fast turn off) for every change in second
4. The push button S1 (J4.33) should be used to only increase the hour of the running digital clock and the hour should roll over to 1 PM after 12 PM and should roll over to 1 AM after 12 AM
5. The push button S2 (J4.32) should be used to only increase the minute of the running digital clock and the minute should roll over to 00 after 59
6. The minute should not roll over to hour when the push buttons S2 is used to change the minutes
7. When the hour or minute is changed using the push buttons S1 or S2 it should generate a beep tone
8. The LCD screen should be cleared when the MKII board is still and stationary for five seconds
9. When the MKII board is rotated about the y-axis (~45° rotation), the running clock should be displayed again on the LCD
10. Do not use any delay loops in the code and make sure all the push button inputs (S1 (J4.33) and S2 (J4.32)) are configured as interrupt-based inputs.

Learning Objectives:
This lab uses 3-Axis accelerometer and the LCD screen to design a simple clock using MSP432 development kit and the Booster pack MKII board.

General Steps:

1. Setup and run the example codes after reading the MK II setup document posted on Blackboard
2. Test examples AccelerometerSerial and LCD_screen_test from File → Examples → 09.EducationalBP MKII.
3. The x and y directions are marked on the MK II board and you should be able to understand the x and y directions when you do the accelerometer example
4. The beep tone can be generated using the function tone(buzzerPin, 4000, 200); where buzzerPin is the pin number, 4000 is tone pitch, and 200 is the delay
5. Make sure to setup the ADC resolution to 12 bits when using the accelerometer
6. Make sure all the push button inputs (S1 (J4.33) and S2 (J4.32)) are configured as interrupt-based inputs. Refer to the link for more information.
7. Using the example codes, complete the lab to meet all the requirements
8. Record your observations in the report.

Detailed Steps:

Hours_Increment() is my clock function I created. It is the primary logic behind a clock and how it increments. Switches time of day to AM or PM based on the time 12:00:00.

```
// Hours Incrementing Function returns void
void Hours_Increment()
{
  if(minutes > 59 && FLAG_MINUTE == 0){minutes=0;hours++;}
  else{FLAG_MINUTE =0;}
  if(hours > 12){myScreen.clear(blackColour);hours=1;}
}
// Minutes Incrementing Function returns void
void Minutes_Increment()
{
  if(seconds > 59){ seconds=0; minutes++;}
}
// Seconds Incrementing Function returns void
void Seconds_Increment()
{
  seconds++;
}
void AM_or_PM()
{
  if(hours == 12 && seconds == 0){AM_PM = !AM_PM;}

  if (AM_PM == 0){time_of_day = " AM";}
   else{time_of_day = " PM";}
}
```

These are my Interrupt Service Routines. The routines themselves only raise a flag for the program to enter execution routine to increment either Hours or Minutes.

```
}
void ISR_hour(){ISRH = 1;}
void Hour_ISR_Execution()
{
  if(ISRH == 1){TIMEOUT_FLAG = 0;reset_Count(); hours++; if(hours == 12){AM_PM = !AM_PM;}
  tone(buzzer, freq1 , 100);tone(buzzer, freq2 , 100);ISRH=0;}
}
void ISR_min(){ISRM = 1;}
void Min_ISR_Execution()
{
   if(ISRM == 1){TIMEOUT_FLAG = 0; reset_Count(); minutes++;tone(buzzer, freq1 , 100);tone(buzzer, freq2 , 100);
            if(minutes > 59){minutes=0; FLAG_MINUTE = 1;} ISRM=0;}
}
```

For the cock to go to sleep after 5 seconds, I had to calculate the average movement over 5 seconds on the XYZ accelerometer values. It reads accelerometer across 5 seconds and stores each second in an array, takes summation, and divides by (i+1). It takes that value and compares it the first second's value with a tolerance of +- 10 using the Sleep().

```
void Acc_Average()
{
          if(i <= 4)
          {
                timeout_arrayX[i] = analogRead(xpin)-2048;
                timeout_sumX += timeout_arrayX[i];
                timeout_arrayY[i] = analogRead(ypin)-2048;
                timeout_sumY += timeout_arrayY[i];
                timeout_arrayZ[i] = analogRead(zpin)-2048;
                timeout_sumZ += timeout_arrayZ[i];
                i++;
          }
                if(i == 4)
                {
                  timeout_avgX = timeout_sumX/(i+1);
                  timeout_avgY = timeout_sumY/(i+1);
                  timeout_avgZ = timeout_sumZ/(i+1);

                }// resets i back to 0 after taking average, each iteration of i is one second

}

void Sleep()
{
  if(
      TIMEOUT_FLAG == 0 &&
      ((timeout_avgX <= (timeout_arrayX[0] + 10)) ||  (timeout_avgX >= (timeout_arrayX[0] - 10))) &&
      ((timeout_avgY <= (timeout_arrayY[0] + 10)) ||  (timeout_avgY >= (timeout_arrayY[0] - 10))) &&
      ((timeout_avgZ <= (timeout_arrayZ[0] + 10)) ||  (timeout_avgZ >= (timeout_arrayZ[0] - 10))) &&
       i == 4)
       {
          TIMEOUT_FLAG = 1;
          i=-1;
       }
}
```

The Lift() checks if the Y accelerometer value is lifted to a +- 45-degree range, if in range flips the TIMEOUT_FLAG back down to zero to trigger Display() to turn on.

```
void Lift()
{
  Serial.print(TIMEOUT_FLAG);
   if(TIMEOUT_FLAG == 1 && MOVE_FLAG == 0 && (((analogRead(ypin) - 2048 ) <= -350)))
              {
                    TIMEOUT_FLAG = 0;

              }
}
void AnyMovement()
{
    if(TIMEOUT_FLAG == 0 && (prevX > ((analogRead(xpin) - 2048 ) + 10)) || (prevX < ((analogRead(xpin) - 2048 ) - 10)) &&
            (prevY > ((analogRead(ypin) - 2048 ) + 10)) || (prevY < ((analogRead(ypin) - 2048 ) - 10))&&
            (prevZ > ((analogRead(zpin) - 2048 ) + 10)) || (prevZ < ((analogRead(zpin) - 2048 ) - 10)))
            {
              reset_Count();
              MOVE_FLAG = 0;
            }
            else{ MOVE_FLAG = 1;}
}
```

Display() displays all the data such as hours, minutes, seconds, and time of day. I used snprintf to be able to use the formatter for the digits of the clock, stores as a string. Using gText to actually send the display data to the screen.

```
void Display()
{
  if(TIMEOUT_FLAG == 0){
    snprintf(buffer, sizeof(buffer),"%d:%02d:%02d" , hours, minutes, seconds); // Converts PrintF into a String
    myScreen.gText(32,54,buffer + time_of_day); }
  else{ myScreen.clear(blackColour);}
}
```

Observations:

Some important observations while completing/testing this lab were I now realize that we should work using functions, each function will be responsible for one task, and we can integrate all the functions with a common purpose. The code is much easier to read and to be able to debug.
I also learned how to interpret the accelerometer data and be able to utilize the information to get an idea of how the user will interact with the clock, it took many attempts to find correct values to use.

Summary:

In this lab, we learned how to integrate an Interrupt routine, bink an LED, use accelerometer data, and how to use all these features perceived concurrently without using any dead loops.