MECH 423 Final Project:
Bed Occupancy Alarm Clock

Dana Deutsch
deutschdana@gmail.com

Project Repo:
github.com/danadeutsch/BedOccupancyAlarmClock

December 10th 2020

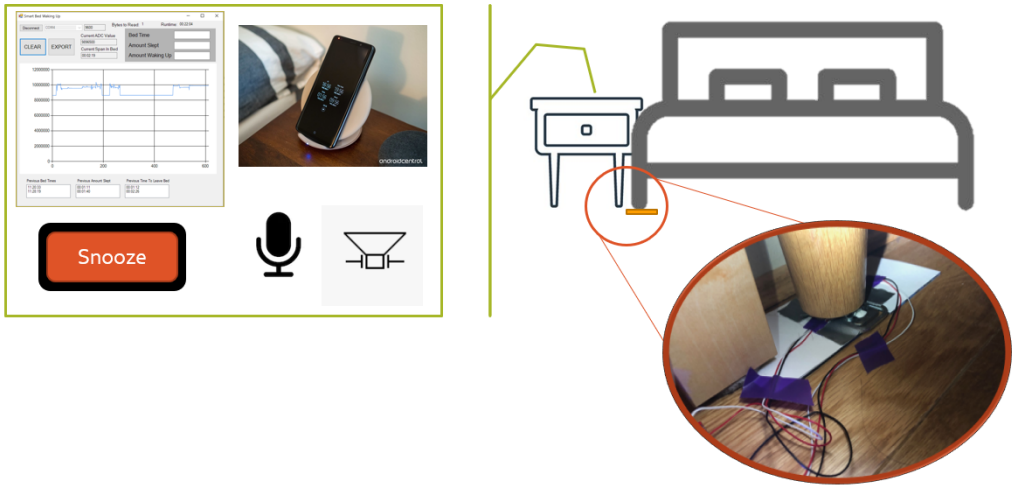


Figure 1: Bed Occupancy Alarm Clock System

Abstract

Many people struggle to get out of bed in the morning. Whether its because they fall back asleep after turning off their alarms or because they lack motivation to get up it doesn't matter. This product's goal is to get get the user out of bed using an alarm system that senses if the user is in bed. Since nowadays people use their phones for everything, including for alarm clock, this product interfaces with the phone using a sound sensor for detecting the phone alarm. This external phone alarm sound is detected in order and begins a MSP430 controlled repetitive alarm system that would keep intermittently ringing regardless of how many times you press the off button unless you leave the bed. Furthermore it will keep monitoring the bed occupancy for an hour after you leave the bed to ensure you don't return. The system is design to set the user out of bed and remain out of bed for at least an hour.

The design uses two 50kg load cells positioned under a leg of the bed to sense the bed occupancy. The voltage signal is amplified using the HX711 amplifier and ADC that is collected by the MSP430 by bit banging. The MSP430 also interfaces with two other inputs that are used for creating the alarm system: the snooze button and the sound sensor. It also uses a piezo buzzer to output the alarm sound and its own internal timers and ISRs to control them all in a set state machine. The MSP430 controls the system but it also sends data values via serial to a laptop for displaying key sleep cycle information.

Contents

1 Objectives

The goal of this project is to create an alarm clock system for people who have a hard time waking up and getting out of bed. Many people simply turn off their alarms and fall back asleep or just lack the motivation to get up. They must then set multiple alarms to remind themselves to get up. This is a temporary solution since you can still turn off all the alarm and stay in bed. My objective of my system is to eliminate the possibility of remaining in bed by taking away the power to fully "turn off" the alarm while keeping a short "snooze" functionality people can enjoy. A consistent alarm that rings while in bed is not beneficial to waking up as it gets annoying fast and stops people from using the device because they are forced to get out of bed right away. Since most people don't use a physical alarm clocks and instead use their phones, this projects goal is to also interfaces with the phone people already use in order to add functionality to it instead of removing an already fully perfect functional system.

Creating a phone application interface is the preferred solution for this but its outside the scope of this project as it adds a lot complexity. Instead this project uses the phone alarm sound as an input point for a bed detecting alarm system which operates separate from the phone on the MSP430. After the original alarm, the device will periodically ring after the user "snoozes" until the user leaves the bed and doesn't return to bed for a set amount of time. This way the main alarm can be set on the phone which users already have set up since it has a better interface and more alarm features, but the rest of the system alarm control is run on the a microcontroller. The device will sense when the phone alarm goes off in order to begin its functions instead of having its own internal real time clock and alarm setting capacity. I do want to simulate the possibilities of sending data to a phone app because I believe its an integral part of how this product can be used in real production not DIY usage. So I created a serial connection with a PC that displayed the sleep cycle data visually and shows one of the many functionalities this data can be converted into with a phone app. One of those functionality if a sleep cycle tracker of how long you sleep or spend in bed which I show. Another use for the bed sensing is connecting the device to other wireless devices to automate home products like the making coffee when you wake up. The future possibilities is endless.

2 Rationale

I want to make this device since it solves a problem I face daily along with many other people. Instead of setting up 5 different alarms which are close together and spread out to make sure I get up, I can use this device. I sometimes ignore my alarms and just decide to keep sleeping so this way I will be forced to get up and hopefully fix my sleeping scheduled. This product isn't available on the market right so its only achievable by making it myself. I hope this project will provide a concept functionality prototype for a possible market device that uses this method or a better phone app interface. As well many home automation can be made using the bed occupancy detection function like starting the coffee maker when you leave the bed and opening the blinds and more. In reality this produce can be created as a fully functional phone application with a Bluetooth connection to the load sensor which would allow for way more features. I want to only simulate this connection with serial to just show how sensing bed values can provide useful data. This project is quite different then other labs and project because I actually apply the material into something useful and not predefined labs. This is particularly seen from load sensors which we learn about it MECH 420 and I apply in this project which also uses the material from MECH 423.

3 Functionality Summery

The hardware components on the system include the microcontroller (MCU), a input bed detection sensor, input audio sensor for detecting the phone alarm, output buzzer to create its own alarm sounds, and laptop for the showing the data-steam functionality. The main controller is the MCU for every functionality. The MCU is used to to control the logic for when the buzzer goes off which are based on port interrupts and force sensor inputs. Timers are used for waiting between states and well as creating sounds with the peizo buzzer. The MCU code reads the output of the force and sound sensor to detect if the user is in bed and if the phone has rung based on calibration values. The MCU sends an output signal to play a changing frequency buzzer sound for the alarm and a data packet to the laptop. I created a few requirements for the functions ordered by importance. They were also developed in this order and therefore combined in this order after the completion of each function. The functional requirements are found in Table 1.

The key feature of the functionality is the bed occupancy detection created by load cells and its relative controlling code. I list below a few requirements I had for the system functions needs to meet.

- The alarm code cannot be tricked to stop the loop without waiting for an hour past the alarm
- The detection of the user must be integrated without compromising the bed comfort or permanently changing the bed structure
- The detection should not interfere with any bed sheet changes and be easily removable
- Buzzer must be laud enough to wake a person from a light sleep since deep sleep wont be achievable in the "snooze period"
- The sound detection shouldn't get accidentally triggered by a person talking while on the bed

Table 1: Functional Requirements for The Wake Up System

	Functions	Effort %
1	Bed Occupancy Detection Circuitry and Reader	45%
2	MCU Alarm Timer Control Code	40%
3	Audio Detection and Output	5%
4	Data Stream Display	10%

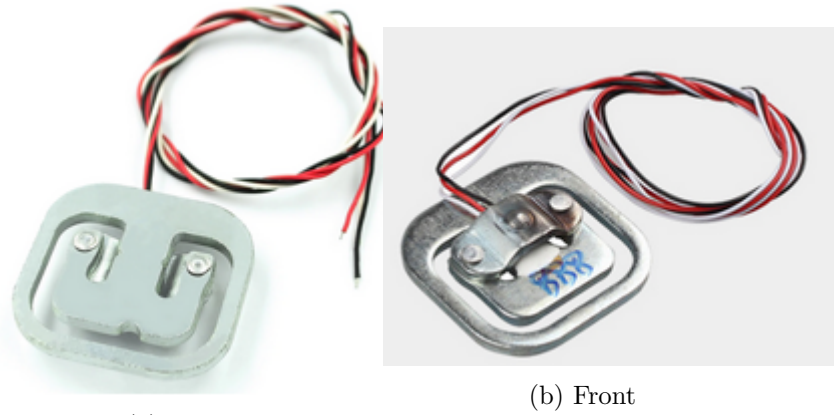
4 Bed Occupancy Detection Circuity and Reader

In order to detect that the user is in bed, I use load cells that are typically found in bathroom scales. I initially considered over solutions such as the thin film load sensitive resistors, but these are both more expensive and fail in a few of the requirements I wanted to meet which are listed above. The load cells I used are actually half bridge strain gauged load cells made from riveted sheet metal. Figure 2 shows an individual cell. As seen in the image the sensor is placed on one side on the metal part on the side the experiences tension when loaded. The cell is one piece of metal with a 'flex' center so we load the metal on the flexing center while mounting only the outer ring of the metal and keeping the part above ground to allow for room to have strain and for the extruding rivets. In order to use strain gauges, we connect them in a wheatstone bridge circuit seen in figure 3. We had learned this in both MECH 420 and MECH 306 along with the need to amplify bridge signals due to the small changes it experiences. In the circuit we sense the voltage across the center of the bridge with a provided input voltage. The bridge voltage changes as the strain gauge resistor in the bridge increases with tension. The voltage I measured across an empty bridge is around 3mV and this value rises slightly when loaded. Since this values are so small we need an amplifier as the ADC on the MSP430 won't be sensitive enough to capture this change in voltage. I am using the HX711 IC on a breakout board that is meant for load cells and is an amplifier

and a 24bit ADC, Figure 4.

Figures 6 and 5 show the sensor block diagram and the board schematic. The data is read from this chip using an output clock pulse signal and reading digital input in a method known as bit banging. The chip has three setting for different gains which are selected using the number of clock pulses in each read. I am using the 25 pulse version which has the largest gain (x128) which allows for the smallest range of voltage detection (0-20mV) which by using I saw that it's more than enough for my use case. Figure 7 shows this communication method as seen from the data sheet. The final circuit for this functionality is then made up of two load cells forming a whetstone bridge read by the HX711 connected to the MSP430 port 1.0 and 1.1 which are input and output ports respectively. Figure 8 shows the entire circuit diagram which has these load cell connections shown with the HX711 breakout board inputs and outputs. The figure also shows the inputs and outputs for our other connection to the MSP430 which will be connected in later sections.

Since I don't need to measure the exact weight of the user for this function, only a change is the overall detected weight, I place the sensors at only one of the bed corners. This makes the device easier to use, install, and over all more adjustable for other beds. From preliminary testing described in later section I knew the loads at a corner is around 40kg in magnitude which is sufficient for two load cells at one corner. In order to mount the load cells I needed to raise it above the ground to provide round for the strain and to hold the rivets above the ground. Since I didn't have access to a 3D printed, I mounted the two load cells across the gap of two piles of sticky notes and duct taped only the outer edge in place. Then I positioned the leg of the bed to press on both directly. The set up is seen in figure 9 both in the loaded and unloaded positions. This mounting isn't very consistent as the load of the bed isn't always placed on the same spot but it works for the detecting functionality. The function of code for reading and averaging the ADC values, global variables named LoadCellVal and Average respectively, via bit banging is seen in listing 1 below.



(a) Back

(b) Front

Figure 2: The Load Cells Used

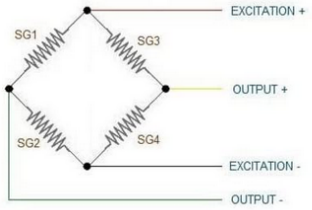


Figure 3: Measuring Across a Wheatstone Bridge

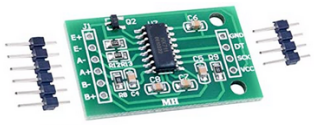


Figure 4: The HX711 Breakout Board

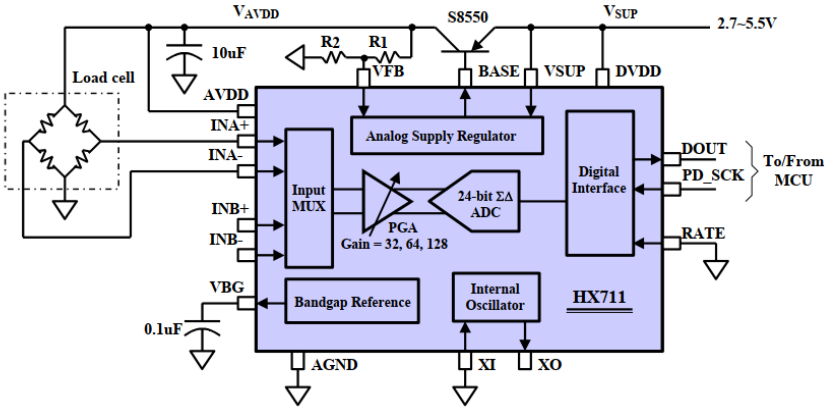


Figure 5: Function Block Diagram for the HX711 Chip

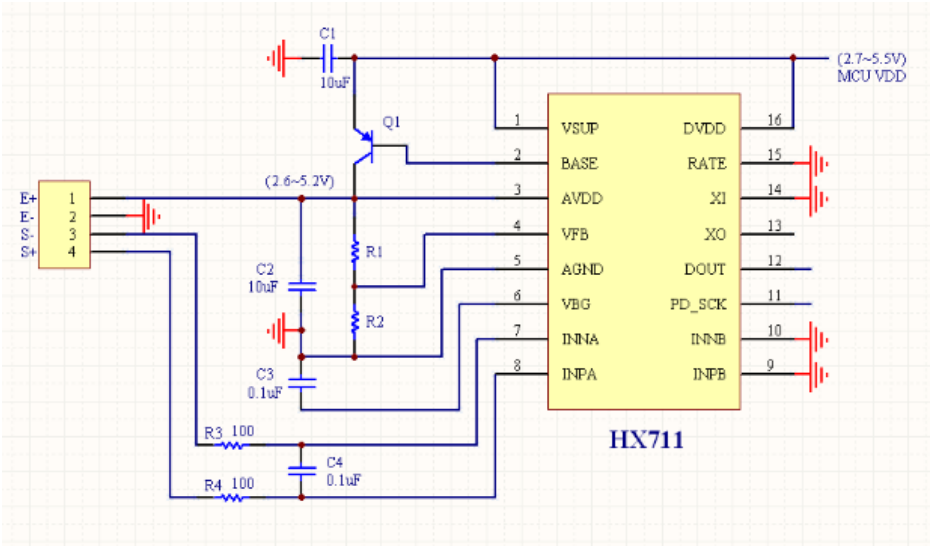


Figure 6: Schematic for the HX711 Load Sensor Board

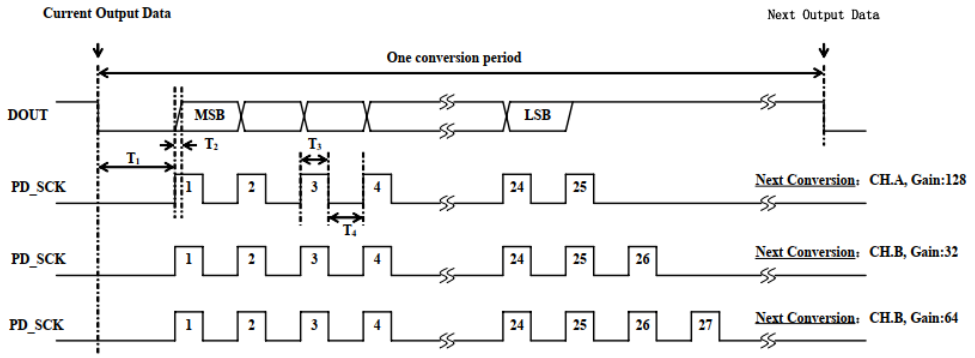


Figure 7: Communication for the HX711 Example from Data Sheet

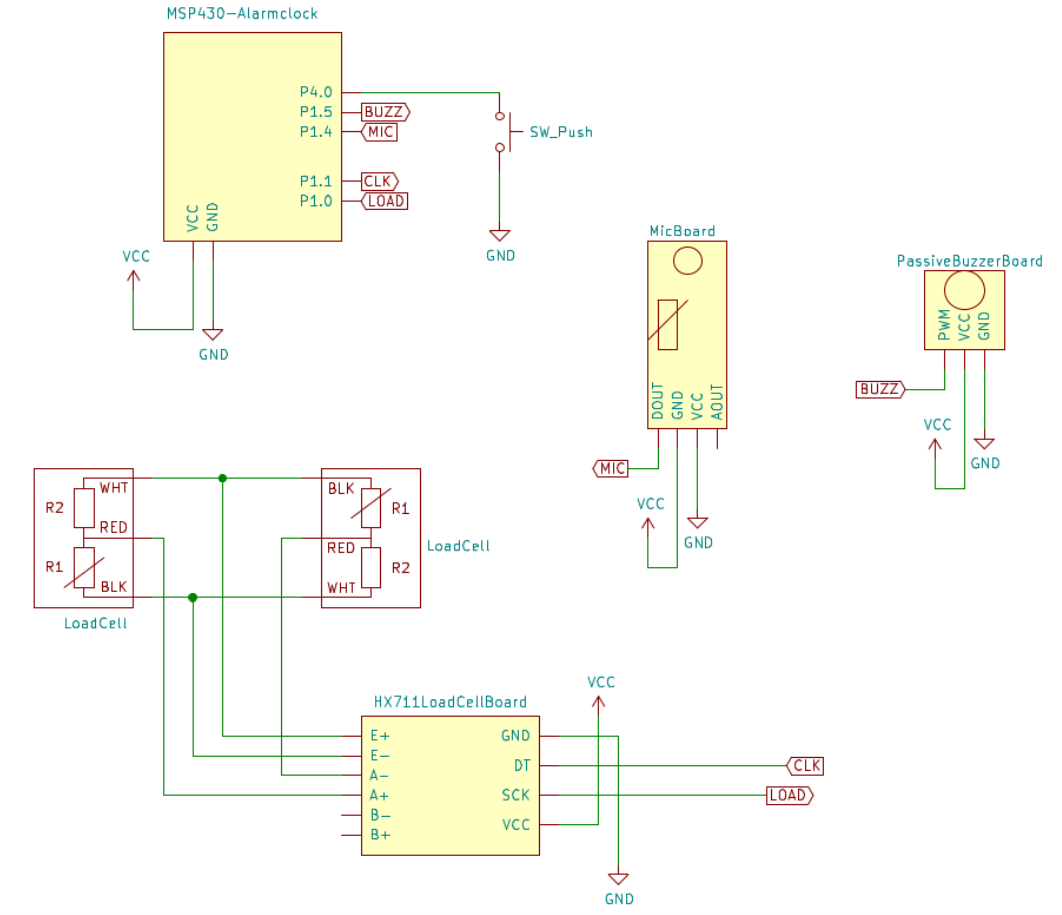
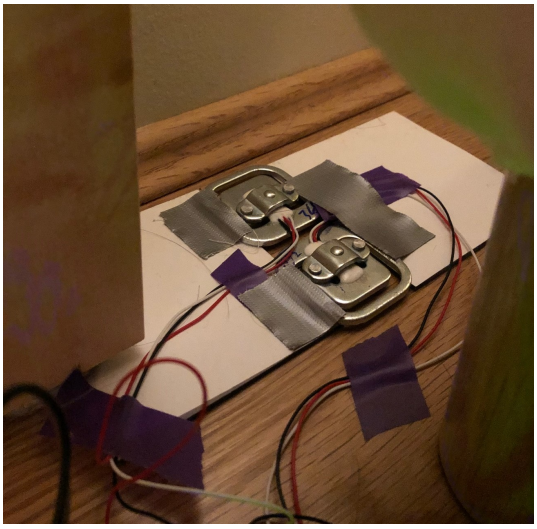
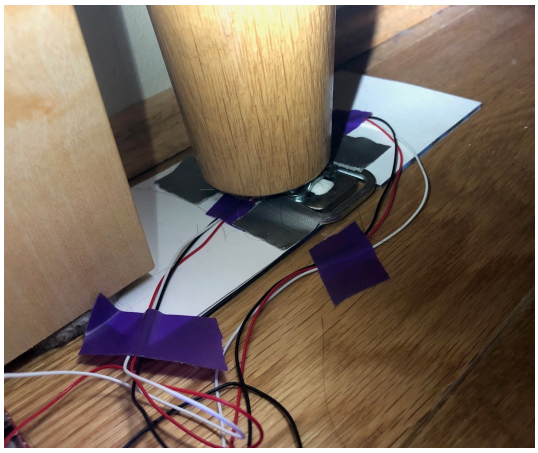


Figure 8: Full Schematic of the System Sensors



(a) Unloaded



(b) Loaded

Figure 9: The Load Cells Used

Listing 1: AX711 Reader Functions For Single ACD Value and Average

```
        void ReadAverageForce(){
                unsigned char count = 0;
                unsigned char numtoAve = 5;
                LoadCellVal = 0;
                for(count = numtoAve; count>0;count--){
                        ReadForce();
                        Average +=LoadCellVal
                }
                Average = Average/numtoAve;
        }

        void ReadForce(){
                unsigned char counter =0;
                unsigned long ReadVal=0;
                while(P1IN & BIT0){};
                ReadVal =0;
                for(counter = 24 ; counter >0;counter--){
                        P1OUT |= BIT1;
                        ReadVal = ReadVal<<1;
                        P1OUT &= ~BIT1;
                        if(P1IN & BIT0){
                                ReadVal |= 0x01;
                        }
                }
                P1OUT |= BIT1;
                ReadVal = ReadVal ^ 0x800000;
                P1OUT &= ~BIT1;
                LoadCellVal = ReadVal;
        }
```

Inputs and Outputs

The input to this function is the person providing force by laying on the bed. This translates to the voltage across the bridge and transfered by the HX711 into a digital input for the MSP430. This input

value is read from the HX711 digital output using an output clock signal for the HX711 bit banging. As well, we provide the supply voltage VCC from the MCU for the sensor circuit.

Parameters

The main parameter is the voltage value which is measured across the bridge which translates as to the value we read from the ADC by the HX711. I chose the largest gain for the HX711 because I knew it always stays below 20mV from testing which is my second parameter. The third parameter I chose was the number of load cells to use and at how many legs. As I mentioned before and seen in the next section this came from testing the wight of the bed and the user and finding the supported values and adding a safety margin. It is also better to not load the cell at its full scale output because they causes more drift and error. The final parameter is the variable in the code i use to record the 24bit value

Testing

I began my tested with concept testing by taking apart a bathroom scale and placing the load cells under the bed, figure 10. From concept testing I devised that placing two sensors under the corner of the bed should be sufficient since each sensor supports up to 50kg. The rough measurements I got from my preliminary test was around 32kg appearing at the far most edge of the bed corner with myself laying on the bed. In this picture I show an empty bed at 17kg. So my set up can allow for my full mass to be placed on one leg and still remain below the full scale output load of the load cell.

While building the circuit I tested to make sure my load cells are outputting voltage reading correctly using the Analog Discovery. Connecting the circuit I measured the bridge voltage to check that Im getting voltage readings in the mV and that pressing on the cells increases the reading signifying my ADC values will be increasing as desired. Figure 11 shows this reading which was around 2mV in this image and fluctuated as it was noisy before the low pass filter and amplifier on the hx711 board.

Once I had the circuit working I tested the reader functions by measuring outputs while in bed at different positions and over a 5hr period to make sure I don't see a huge drift in the value for some reason. I recorded the empty and occupied bed values to note measurements change. I did notice that any repositioning of the load cell will change the values so its important for me to set a passing criteria value that is close to the empty bed value but not too close to have accidental triggers. I decided to make the final adjustments for passing criteria after the full system is built since I wanted to use a bed leg that is harder to reach and position and therefore the testing for this was done on another leg. The final passing parameter for the load cell reader code is determined right before conducting a test by measuring an empty bed, adding 0.1million to the value and rounding up to nearest 0.05million. The value I used for the chosen bed leg was 8,650,000. Figure 12 shows the set up I used for testing the working circuit using the most accessible leg of the bed to check circuit functionality. This set up was also used to check I can detect is the user is in any position on the bed including but not limited to sitting at the most opposite corner. I walked on top of my bed and made sure the value can still be detected at any location.

Figure 10: Preliminary Test With Bathroom Scale

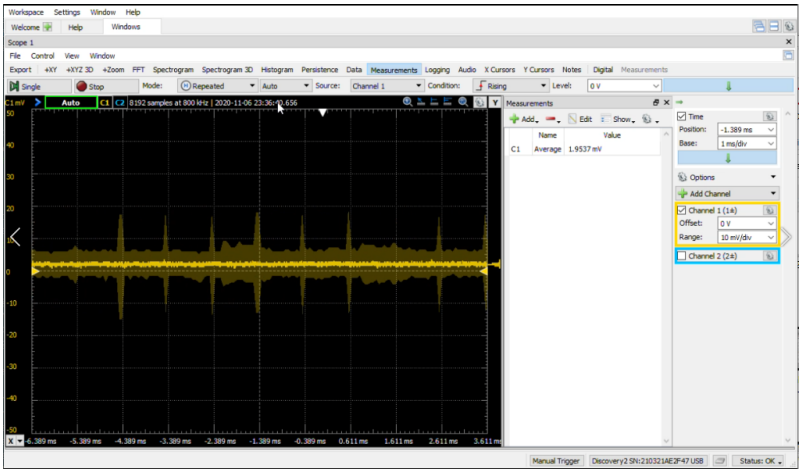


Figure 11: Voltage Reading of Unloaded Bridge Powered with 5V

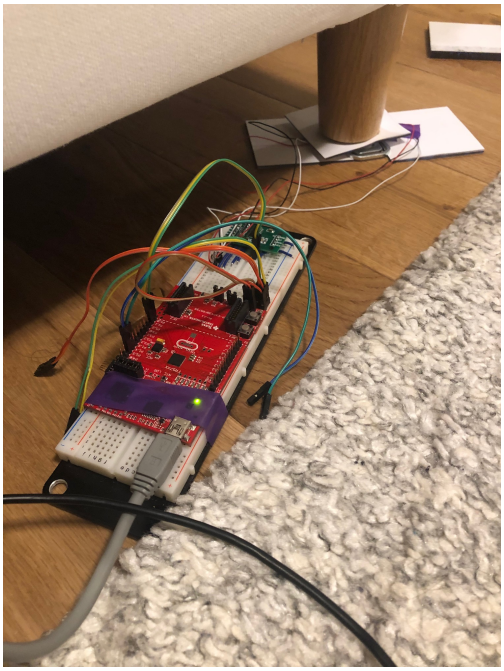


Figure 12: Testing of the Load Cell Circuitry and Paper Holder

5 Main MCU Timing Code for Alarms

Approach and Design

The MCU is the control behind when the alarms go off based on our hardware inputs. The flowchart to the logic behind the alarms are found in figure **??**. The premise of the flow of the states it that there are 4 main stages: Waiting for alarm, waiting between buzzers , getting the user up to press snooze while ringing, and checking that the user remains out of bed. First we have the waiting for the sound input, which needs to reset some values before waiting for an ISR on Port 1 to initiate by the sound sensor. The ISR checks if the user is in bed at the time of the alarm. If they are then then we move to the next stage of getting them out of bed, but if they aren't then we move to the stage of ensuring they stay away from the bed. In order to get the user up the buzzer goes off intermittently which forces the user to keep getting up to shut it off which doesn't let them fall back into a real sleep. We wait for a set amount of second using TB1 and then ring buzzer and wait for an ISR on Port 4. We repeat this loop checking every time our wait finished whether the buzzer is required based on bed status. Finally our last stage is counting up to an hour while the user is out of bed. In this stage we check intermittently so the user can still sit down on the bed maybe to check notification for a min before being reminded to get up. I found that an intermittent reminder is more user friendly then a constant check and does a better job to get you up.

The code uses 7 states in an infinite loop which relays on the use of a one second timer interrupt, and port interrupts. The one second timer interrupt is used to count the 'waiting' time between the alarms and counting the second 1hr out of bed check sequence. The timer is always running so when ever we need to use it we can count the number of interrupts and preform tasks accordingly. Its also used to send serial data to the computer and to wait between the frequency changes of the buzzer discussed in next sections. The code was first written with the buzzer and force sensor as digital input/outputs simulated using the analog discovery before combining their real functions. The fully final code is found in appendix A and in this repository.

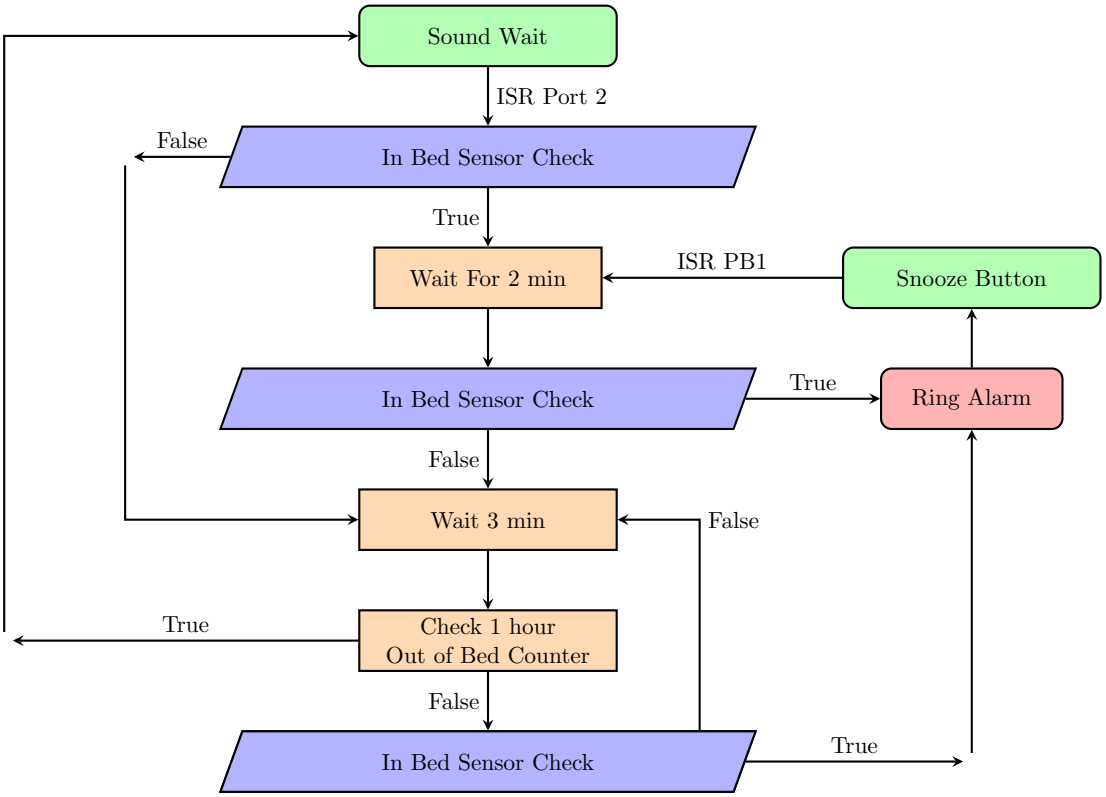


Figure 13: Flowchart for the Alarm Actuation Logic

Inputs and Outputs

The main input is the determined output from the bed occupancy detection. This will be an ADC read value to determine if the level passed the set calibrated value and the person is in bed. The second input is the output from the phone audio sensor which actuates the entire flow chart. The third input is a snooze button for the user to temporarily turn off the alarm. Finally we have the output buzzer to get the user out of bed which will be set with a PWM signal that changes frequencies for different sounds. For this function part they were tested using simple IO signals instead of the real sensors.

Parameters

The software will check and use the following to function and later add the parameters from other sections:

- The state of the bed as a digital high or low
- The state of the audio volume as a digital high or low
- Count in seconds the timer interrupt needs to pass before snooze alarm needs to start
- Count in seconds the timer interrupt needs to pass before secondary check needs to recheck bed status
- Count of number of times the secondary timer check loop has gone off to provide enough time for out of bed detection (1hr= 20*3min)
- The comparison value for the previous three counting variables

Testing

Testing was preformed using test cases based on the state machine. I pretended to be the user using the simulated inputs and outputs and went through the stages of the flowchart. I did this over and over again and kept preforming these as the code changed with additional function combination. All the testing was

preformed using shortened time frames. Instead of waiting an hour I only had to wait 2min and instead of waiting 2min I had to wait 15seconds.

6 Phone Audio Detection and Audio Output

Approach and Design

This function encompasses the two sound related traducers I plan to use and the code controlling them. It also includes connecting them to the previous section of code, so testing this function actually also included combined all my sensors to my code including the load cell reader passing criteria. To detect the phone alarm, as long as the phone is placed right on top of the product, I can measure the volume of sounds and only actuate on laud enough sounds using a microphone. I found a microphone attached with a audio amplifier board seen in figure 14. The schematic of this board is found in figure 15, The board has a both analog and digital inputs where the analog is the non amplified signal. I planned to test both and see which one work best, and I found that adjusting the resistor on the board which controls the level at which the digital output changes is sufficient for the application since the microphone on the board was not picking up quite sound only larger ones like the alarm blaring beside it or dropping a laptop on my desk. I used an ISR to trigger the code when the input signal from the sensor goes high. To create the output alarm I used a piezo buzzer board which has the switching controller on board so I didn't need an external mosfet and just had to provide a digital PWM signal along with my Vcc. The PWM was created by another timer but the frequencies changed to have a non-constant sound. I change between two different frequencies for the buzzer timer which are swapped every 1sec while the buzzer is on. The frequencies represent the real sound frequency we hear.

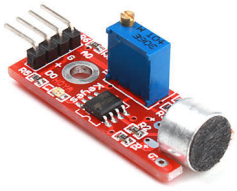


Figure 14: The microphone board used

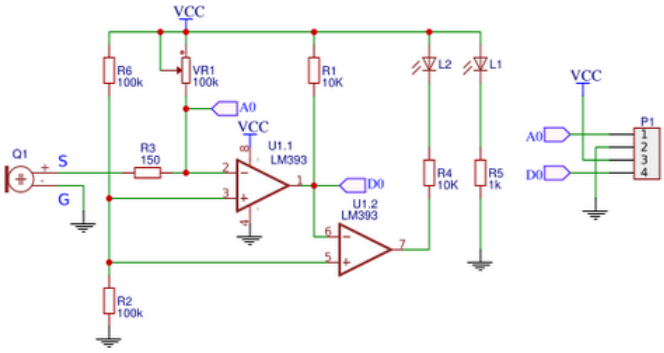


Figure 15: Microphone Board Schematic

Inputs and Outputs

The output to the buzzer system is a PWM signal that switches the piezo buzzer to create the sound. In particular this comes from a timer output from the MSP430 which we change the frequency. As well the input to the microphone is the sound of the phone alarm to the microphone that creates the signal for detection.

Parameters

The first parameter is the resistance we need to calibrate the sound sensor to provide a digital high signal when the alarm goes off but not in other less laud noises. This is adjusted with the variable resistor on the board while the phone alarm is on. The second value is the frequency which we send to the buzzer to create different frequency sounds. I chose this frequencies based on two high frequencies humans can hear and just played around to get what I though was an annoying frequency pair that the user wont want to listen to over and over.

Testing

The test plan for the buzzer is simple, see if the sound is outputted and is heard well enough. I played around with the frequencies before settling just on random sounds I didn't like so I would avoid listening to it and leave the bed faster. For the microphone I tested various phone alarm tones to check it can detect them and determine how close or far away to the microphone I need to be in order to trigger my code. I found that the microphone didn't respond to most talking and general room noise however if I dropped my phone near it or my laptop it would trigger. This created some toning with the variable resistor and I found that it needs to be placed around 15cm away from the phone speaker so just dropping my phone on the charger table wont trigger the alarm but still be heard when the phone rings.

7 Window Form Display

Approach and Design

The goal of the display is to add a data stream from the MSP430 without actually controlling the alarm code. This makes the display optional since having a computer beside the bed is sometimes difficult and wasteful keeping it always on and recording data. The serial connection is therefor a one direction connection where a data packet is send to the PC. This input packet is described in the input and outputs section below. The data link doesn't disconnect from the computer as long as the computer is on which is done by changing the power setting and using a sync byte. The data displayed is the current ADC value along with the current time the user has been in bed for. The two main data values we read from the display is the time it takes the user to wake up and how long they have slept for before the alarm woke them up. This allows the user to see their progress and hopefully they will get more sleep and wake up faster while using the device. Figure 16 shows the displayed output for data collection. As well the data can be outputted as a csv files which is useful in testing.

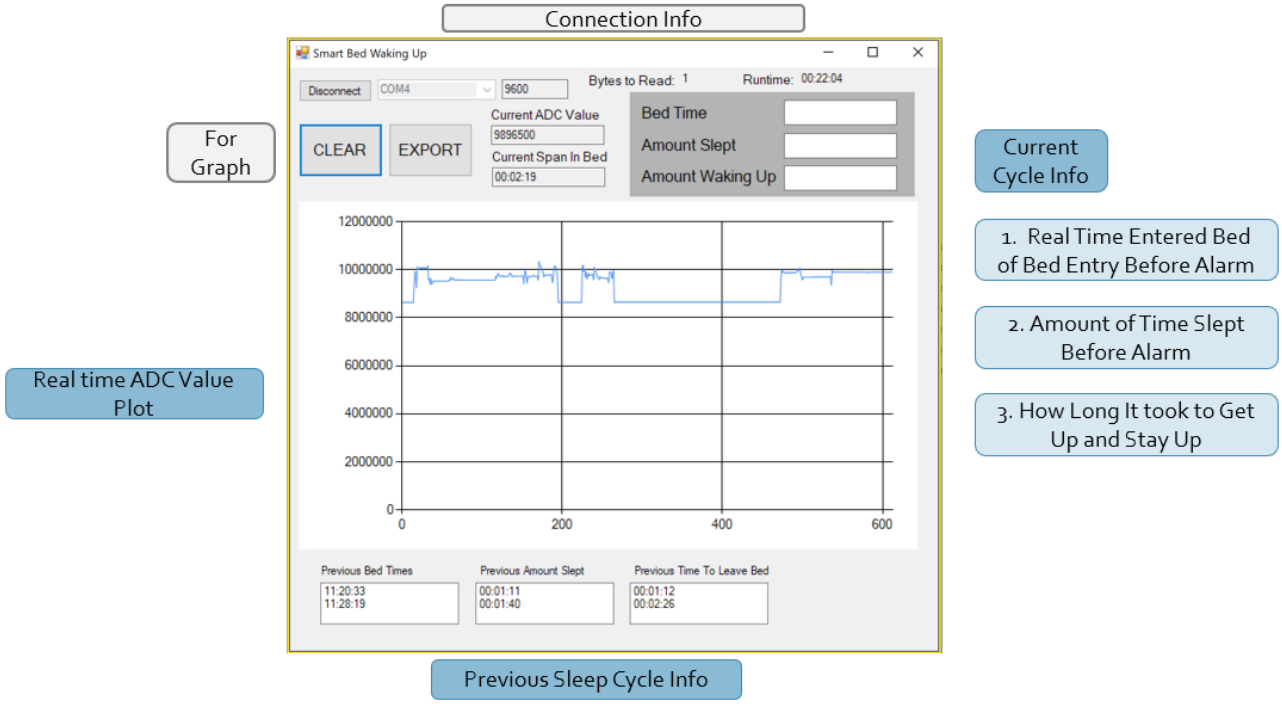


Figure 16: User Data Display

Inputs and Outputs

The input of this function is a serial packet sent by the code in the one second timer. The packet is described in Table 2. The data bytes are the 24bit ADC value and the command byte gives information on the state of the alarm cycle. I did notice some bytes get lost when the computer is unlocked after a long time so the Sync byte is very important. This serial connection isn't a perfect solution its unreliable and require continuous connection via a cable. Maybe in the future I will connect it to a wifi or bluetooth module instead. The output is the display user interface and a CSV output recording of the ADC values.

Table 2: Approximate Contour Errors relative to Feed Rate

Sync	Data 1	Data 2	Data 3	Command
255	ADC 2 Compliment Value			0 - nothing 1 - alarm sounded 2 - cycle ended

Parameters

the first parameter I chose was how often to send the data packet. Since the program operates in minutes and hours, no one really measures how long they sleep in seconds, I determined that it doesn't need to be too precise and so that using DateTime objects is better then the serial sending rate. I decided to send the pack once a second in the same timer interrupt that is always on because it is just good enough for testing functionality in the range of tens of seconds while still working well in the range of minutes. Sending the packet too fast will just cause too many useless data points anyways. Below are a list of parameters used in the C# program. They are used to determine the time values displayed on the interface.

- DateTime at start of program

- DateTime at the time the user entered the bed detected by ADC value changing from the data stream
- DateTime when the alarm rung detected via the the steam command byte
- DateTime when the user left the bed detected by ADC value changing from the data stream
- Current ADC value
- DateTime when the alarm cycle has finished
- Current time

Testing

To test the User interface I tried using it while running the system state machine cycles. I started by cheeking that I am receiving the correct packets at the correct states using the mech 423 serial reader. Then I transitioned to checking that the values appear on the UI as expected. I checked for example that the time it took to wake up not only appears correct value wise but that it also appears right as you leave the bed. As well checked that it updates when you exit the bed the bed after returning. The testing of the system for the UI were a repetition of the state machine testing except I focused on the information displayed on the screen.

8 Full System Setup and Evaluation

The system evaluation consisted of running the cycles of the state machine over and over in different combination of exiting and returning to bed to check that the system works with all the attached complements. The main evaluation and testing was already discussed and preformed in order as the functions were completed and combined. Since the first three functions where combined and the last function required to be used with them working, the final testing was really just the testing to make sure the real values are being displayed correctly on the user interface with the testing from the second function. I also just checked it while in the correct bed leg and with the physical setup by the bed.

The following figures show the full system setup. Figure 17 shows the bed side Table placement of the sensors. This includes the microphone, push button, and buzzer placed beside the wireless phone charger. Figure 18 shows how the rest of the system is set up under the bed side table. The close up on the MSP430 which is connected to a bread board with all the circuity is seen in Figure 19. The figure also shows how the Load cells are placed under the bed leg connected to the breadboard.

Figure 17: Bed Side Table Set Up

Figure 18: System Setup Underneath the Bed Side Table

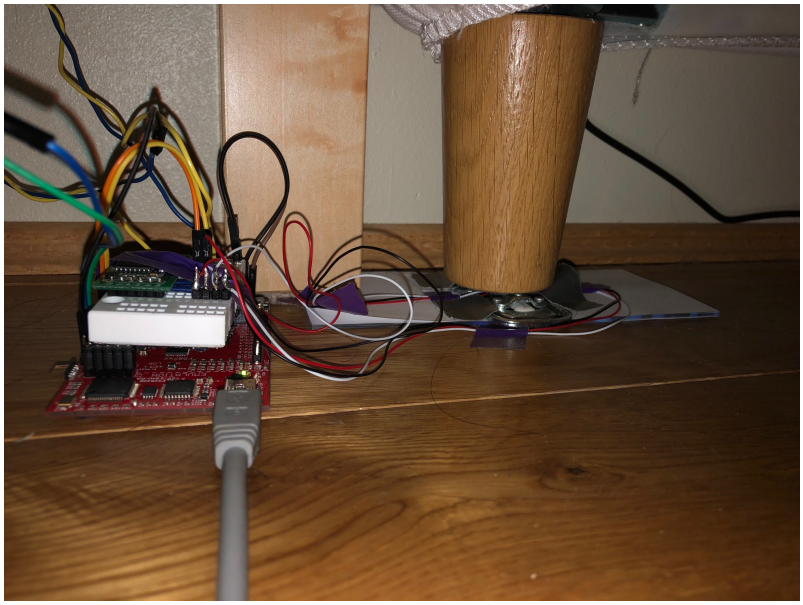


Figure 19: System Beside the Measured Bed Leg

9 Reflection Questions

What worked and what didn't work? Why? What would you do differently if you could do it again?

I believe the part of the system that worked the best was using load cells and their respective circuitry. They were way more accurate and sensitive then I expected and after figuring their setup which was a little tricky initially I could see how great they were. What I think worked the least well was the serial connection for the sleep data. It sometimes lost bytes for long period of couple hours and isn't truly beneficial in a real use case. If I were to do something differently I would want to learn how to set up a connection to the phone. Forgo the display and maybe a simplified MSP state machine and learn about sending wireless signal to a phone to just detect the bed occupancy instead of the whole sleep cycle information. Its more work but its something that I think would be valuable to just learn if I had more time and resources. Its not something we had learned in our classes but by simplified the system I could have focused on it instead.

Identify 3 things you learned in MECH 423 that you consider the most useful and why

I learned how to write embedded C for the MSP430 which I think is important for knowing how microprocessors work and their functionalities and how to set them up. This is not particular to the specific chip itself but applicable stuff for other chips. The second useful thing I learned was how to use encoder to control a DC motor and create the feedback controller. I found this a good challenge and learned application skills that weren't just from 423 but based on 467 as well. I can see myself wanting to do something with controls in the future because of how I enjoyed the controls classes we had so I see this as an important experience. The third thing I found useful was creating C sharp programs. C sharp is a very used language and creating these stand alone programs can be very important as a future mechatronics engineer. I had the change to create a matlab application at work before so I can see ways were knowing to make these C sharp apps can be useful. Its also a language that I have seen requested in more software related jobs.

What are some limits of your knowledge and expertise as a mechatronic engineer? Identify 3 things you would like to learn going forward. What is your strategy to acquire knowledge in these areas?

The first thing I wish we had more in the curriculum is application circuits and just a hardware focused course. Two particular thing I would have enjoyed learning about in the mechanization curriculum are designing PCBs and considerations when designing application circuits like power circuits or sensor circuits. I would have like to have more hardware experience like this. In a similar aspect to the last few lectures of mech 423, something application based maybe through labs. I have done my own learning about hardware instead in particularly on my design team. I used my chance on the team to learn material I don't in class like using circuit design CAD software and learning about buck-boost converters, LDOs, and reverse polarity protection. These first two things are more electrical based and our limitations to more hardware related jobs which i would be interested in. We are also limited to software positions and I think learning SQL and maybe also python would be helpful for that path. Personally I am less likely to peruse this path so I don't plan to learn SQL myself, but I have been trying to learn python to work with raspberry pi for personal projects.

A Final C State Machine Code

```
#include <msp430.h>
#include <stdbool.h>       // Gives the true and false symbols
#include <iso646.h>        // Gives the and and or symbols
#include <stdint.h>        // Gives the uint32_t symbol

//FUNCTIONS
void generalAlarmsSetup(void);
void checkForSoundState(void);
void bedCheck1(void);
void outOfBedState(void);
void bedCheck2(void);
void ReadForce();
void setupHX711();
void ReadAverageForce(void);
void sendForceToSerial(unsigned char command);

//VARIABLES
volatile bool bedOccupied = false;
volatile bool soundSensed = false;
volatile unsigned int timercounter = 0; // count var for timer
volatile unsigned int waitingTime = 0;  // number of counts for the full timer cycle
volatile char state = 0;
volatile char counterForOutCheck = 0;

#define BUZZERPORT P3OUT
#define BUZZERPIN BIT5
#define BEDPIN  BIT4
#define NOIZEPIN BIT5
#define NOISEPORT P2IN
#define BEDPORT P1IN

#define SNOOZETIME 15  //in seconds
#define CHECKTIME 30   //in seconds
#define OUTOFBEDPASS 4//in check time intervals
#define TONE1 120
#define TONE2 50

volatile int tone = TONE1;
#define BEDLIMIT 8750000
volatile unsigned long long int LoadCellVal;

/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    generalAlarmsSetup();
    setupHX711();
     __enable_interrupt(); // enable global interrupt

    while(1){
        switch(state){
        case 0:
            checkForSoundState();
            break;
        case 1:
            bedCheck1();
            break;
```

```
        case 2:
            break;
        case 3:
            outOfBedState();
            break;
        case 4:
            bedCheck2();
            break;
        case 5:
            break;
        case 6:
            break;
        case 7:
            break;
        }
    };
    return 0;
}


//STATE 0
void checkForSoundState(){
    P1OUT &= ~BIT3; // ending the cycle light
    //TB1CCTL1 &= ~CCIE;      //disable TB ISR
    P4IE &= ~BIT0;          //disable snooze interrupt
    P1IFG &= ~BIT4; // clear this ISR flag
    P1IE |= BIT4;          //enable noise sensor interrupt
    P1IFG &= ~BIT4; // clear this ISR flag

    state = 7;
    sendForceToSerial(3);
    //noise ISR will change so something else that would also be an isr probable
    //Clear all variables for next round
}

//STATE 7
// ISR for polling the noise of the phone alarm
#pragma vector=PORT1_VECTOR
__interrupt void port_1(void){
        P1OUT &= ~BIT3; // starting the cycle light
        state = 1;
        P1IFG &= ~BIT4;
        sendForceToSerial(1);
        P1IE &= ~BIT4; // clear this ISR flag

}
//STATE 1
void bedCheck1(void){
    //Perform the collection of ADC for force sensor
    ReadForce();
    if(LoadCellVal > BEDLIMIT){ // bed high aka occupied
        state =2;
        TB1CTL |= TBCLR;            // clear TB clock
        TB1CCTL1 |= CCIE;           // enable TB counter
        waitingTime = SNOOZETIME;   //set the timer to count up to the snooze time
        timercounter =0;
    }else{
        counterForOutCheck =0;
        state = 3;
    }
}
```

```
121
122
123 //STATE 5
124 //ISR for Snooze Button to turn off alarm and start the snooze timer ISR after
        clearing its clock
125 #pragma vector=PORT4_VECTOR
126 __interrupt void port_4(void){
127
128     BUZZERPORT &= ~BUZZERPIN;   // alarm off
129     P1DIR &= ~BIT5;
130     P4IFG &= ~BIT0;             // clear this ISR flag
131     P4IE |= BIT0;               //disable snooze interrupt
132     state = 2;
133     timercounter =0;
134     TB1CTL |= TBCLR;            // clear TB clock
135     //TB1CCTL1 |= CCIE;           // enable TB counter
136     waitingTime = SNOOZETIME;
137 }
138
139 //STATE 2 and 6
140 //Snooze timer checks if person in bed and activated alarm if in bed,
141 //  and if out of bed disable this ISR
142 #pragma vector = TIMER1_B1_VECTOR
143 __interrupt void TimerB1_snooze(void){
144     timercounter++;
145
146     if(state==5){
147         switch(tone){
148         case TONE1:
149             tone = TONE2;
150             break;
151         case TONE2:
152             tone = TONE1;
153             break;
154         }
155
156         TB0CCR0 =tone;
157         TB0CCR2 = tone/2;
158     }
159     if(timercounter == waitingTime){// when counting time is up
160         if(state == 6){ // out of bed check
161             counterForOutCheck++;
162             state = 4;
163         }else if(state == 2){ //in bed check
164             ReadForce();
165             if(LoadCellVal > BEDLIMIT){ // bed high aka occupied
166                 BUZZERPORT |= BUZZERPIN;    // buzzer light on
167                 P1DIR |= BIT5;              // buzzer on
168                 state = 5;
169                 P4IE |= BIT0;               //enable snooze interrupt
170             }else {
171                 counterForOutCheck =0;
172                 state = 3;
173             }
174         }
175     }
176     ReadForce();
177     sendForceToSerial(0);
178     TB1CCTL1 &= ~CCIFG; //Clear TB flag
179 }
180
```

```
//STATE 3
void outOfBedState(void){
    waitingTime = CHECKTIME;  //set the timer to check
    TB1CTL |= TBCLR;            // clear TB clock
    TB1CCTL1 |= CCIE;           // enable TB counter
    timercounter = 0;
    state = 6;
    sendForceToSerial(2);
}

//STATE 4
void bedCheck2(void){
    if(counterForOutCheck == OUTOFBEDPASS){
        state = 0;

    }else{
        ReadForce();
        if(LoadCellVal > BEDLIMIT){ // bed occupied
            P4IE |= BIT0;               //enable snooze interrupt
            BUZZERPORT |= BUZZERPIN;    // buzzer on
            P1DIR |= BIT5;
            counterForOutCheck = 0;
            state =5;
        }else{
            TB1CTL |= TBCLR;            // clear TB clock
            TB1CCTL1 |= CCIE;           // enable TB counter
            timercounter = 0;
            state = 6;
        }
    }
}


void ReadForce(){
    unsigned char counter =0;
    unsigned long ReadVal=0;

    while(P1IN & BIT0){};
    ReadVal =0;

    for(counter = 24 ; counter >0;counter--){
        P1OUT |= BIT1;
        ReadVal = ReadVal<<1;
        P1OUT &= ~BIT1;
        if(P1IN & BIT0){
             ReadVal |= 0x01;
        }
    }

    P1OUT |= BIT1;
    ReadVal = ReadVal ^ 0x800000;
    P1OUT &= ~BIT1;

    LoadCellVal = ReadVal;
}

void setupHX711(){
    P1DIR &= ~BIT0; // the in values
    P1DIR |= BIT1; // the clock signal
    P1OUT &= ~BIT1;
    P4OUT |= BIT0; //pull up
```

```
    P1REN |= BIT0; //enable pull up

}

void sendForceToSerial(unsigned char command){
    unsigned long sending = LoadCellVal;
    char data;
    data = 255;
    UCA0TXBUF = data;
    while ((UCA0IFG & UCTXIFG)==0);
    data = sending >>16;
    UCA0TXBUF = data;
    while ((UCA0IFG & UCTXIFG)==0);
    data = sending >>8;
    UCA0TXBUF = data;
    while ((UCA0IFG & UCTXIFG)==0);
    data = sending;
    UCA0TXBUF = data;
    while ((UCA0IFG & UCTXIFG)==0);
    UCA0TXBUF = command;
    while ((UCA0IFG & UCTXIFG)==0);
}

void generalAlarmsSetup(void){
    //CONFIGURE CLOCKS
    //**********************
    CSCTL0_H = CSKEY_H;       // Clock password resister to allow writing

    // Set DCO to 8MHz
    CSCTL1 = 0;               // Clear DCO settings
    CSCTL1 &= ~DCORSEL;       // DCOR Mode Select 0
    CSCTL1 |= DCOFSEL_3;      // DCO selection frequency

    CSCTL2 = SELM_3 + SELA_3 + SELS_3;      // All clocks get source of DCO,!(32768
        hz)
    CSCTL3 = DIVS__32 + DIVA__1 + DIVM__32;     // set dividers to 1 or 32
    CSCTL0_H &= ~CSKEY_H;                   //lock the clock


    //CONFIGURE UART
    //***************
    //configure P2.0 ad P2.1 as the Rx and Tx for UART
    P2SEL1 |= BIT0 | BIT1;
    P2SEL0 &= ~(BIT0 | BIT1);

    UCA0CTLW0 = UCSWRST;    //Reset the eUSCI
    UCA0CTLW0 = UCSSEL0;    //set clock source ACLK

    // Configure the baud rate from the 8MHz of DCO to 9600
    UCA0BRW = 52;
    UCA0MCTLW |= 0x4900 + UCOS16 + UCBRF0;


    //CONFIGURE IO PINS FOR FAKE BED INPUTS
    P1DIR &= BEDPIN;
    P1OUT |= BEDPIN;
    P1REN |= BEDPIN;

    //CONFIGURE Noise sensor INPUT for ISR
    P1DIR &= ~BIT4; //input
    P1IES &= ~BIT4; // interrupt edge select low to high
```

```
302     P1IE &= ~BIT4;  //enable noise sensor interrupt
303     P1IFG &= ~BIT4; // clear interrupt
304
305     //CONFIGURE BUZZER IO OUTPUT LED
306     P3DIR |= BUZZERPIN;
307     BUZZERPORT &= ~BUZZERPIN;
308
309     //CONFIGURE SNOOZE BUTTON AS PB
310     P4DIR &= ~BIT0; //input
311     P4OUT |= BIT0; //pullup selected
312     P4IES &= ~BIT0; // interrupt edge select low to high
313     P4IE &= ~BIT0; // disenable port interrupt for now
314     P4REN |= BIT0; //enable pullup/down resistors
315     P4IFG &= ~BIT0; // clear interrupt
316
317     //CONFIGURE TIMER B1 FOR 1s
318     // wanted freq is 1hz
319     TB1CTL |= TBSSEL_2 + ID__8;     //select the SMCLK and /8 divisor
320     TB1CTL |= MC__UP;               // up mode to CTL0 and enable interrupt
321     TB1EX0 |= TBIDEX__8;            // additional /8 divisor
322     //TB1CTL &= ~TBIFG + TBIE; //no pending interrupts and disable it for now
323
324     // f = 8000000/32/8/8 = 3906
325     // f = 3906/3906aka 1sec
326     TB1CCR0 =3906;
327     TB1CCR1 = 1953;
328     TB1CCTL1 |= OUTMOD_3;   // output on set reset
329     TB1CCTL1 |= CCIE;       //dis-enable interrupt OG
330     P3DIR |= BIT4;
331     P3SEL0  |= BIT4;
332
333
334     //CONFIGURE TIMER B0 FOR Buzzer
335     // wanted freq is 1hz
336     TB0CTL |= TBSSEL_2 + ID__8;     //select the SMCLK and /8 divisor
337     TB0CTL |= MC__UP;               // up mode to CTL0 and enable interrupt
338
339     // f = 8000000/32/8 = 31,250
340     // f = 31250/310 = 100hz
341     TB0CCR0 =tone;
342     TB0CCR2 = tone/2;
343     TB0CCTL2 |= OUTMOD_3;   // output on set reset
344     TB1CCTL2 &= ~CCIE;       //dis-enable interrupt OG
345     P1DIR &= ~BIT5;
346     P1SEL0  |= BIT5;
347
348     P1DIR |= BIT3;
349     P1OUT &= ~BIT3;
350
351
352
353 }
```