Carlton Duffett
Neeraj Basu
EC450 Final Project
5/1/2015

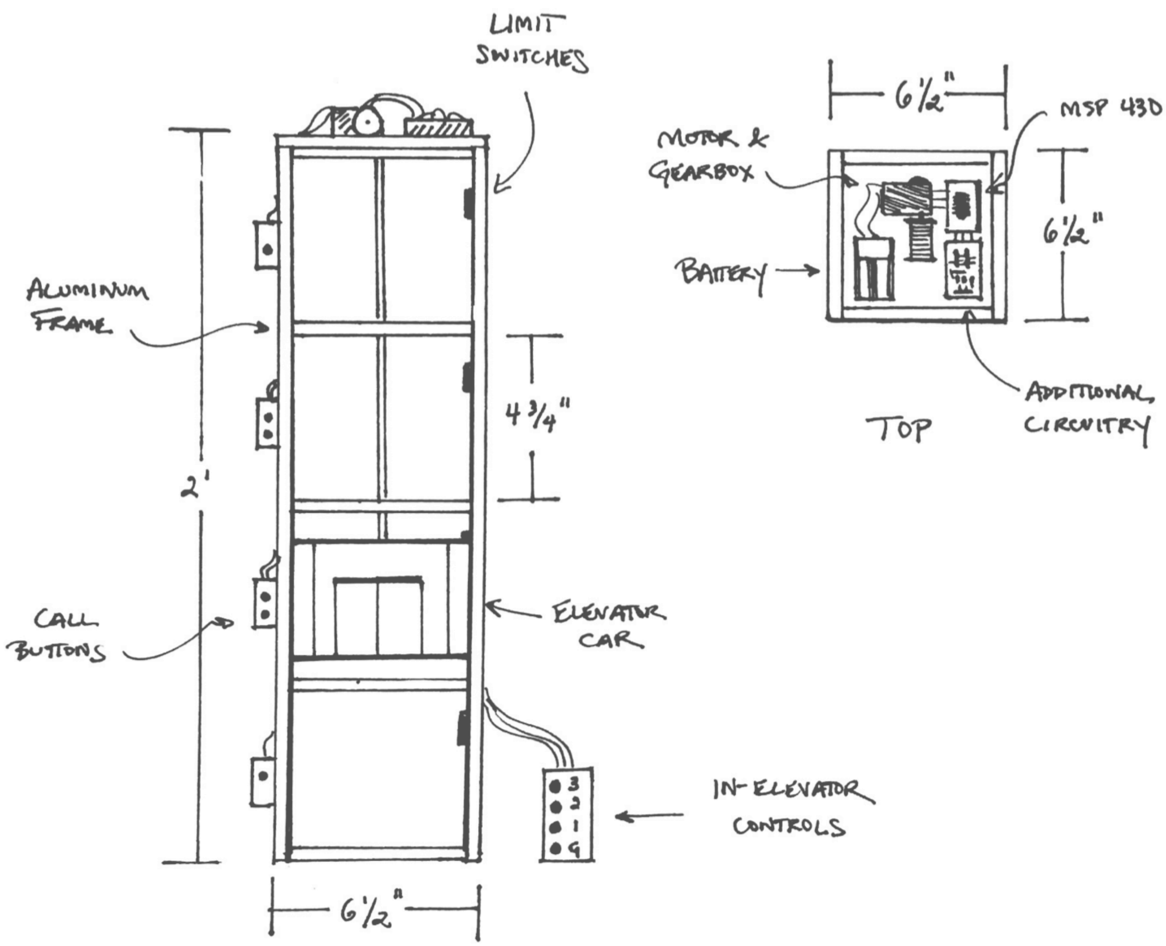
Final Project Report

I. Project Goal

The goal of our project was to design and build a four-floor model elevator driven by a single MSP430. Our goals included:

1. A 2-foot tall aluminum structure with 4 floors and a moving elevator car.
2. A motor, gearbox, and wire spool driven at variable speeds using pulse width modulation, operating in both directions using an h-bridge system.
3. Bi-directional limit switches to accurately detect the position of the elevator car.
4. Additional circuitry to encode the switches, buttons, and to drive a 7-segment display.
5. A sophisticated control algorithm that dynamically adjusts elevator speed and optimizes stops when ascending and descending.

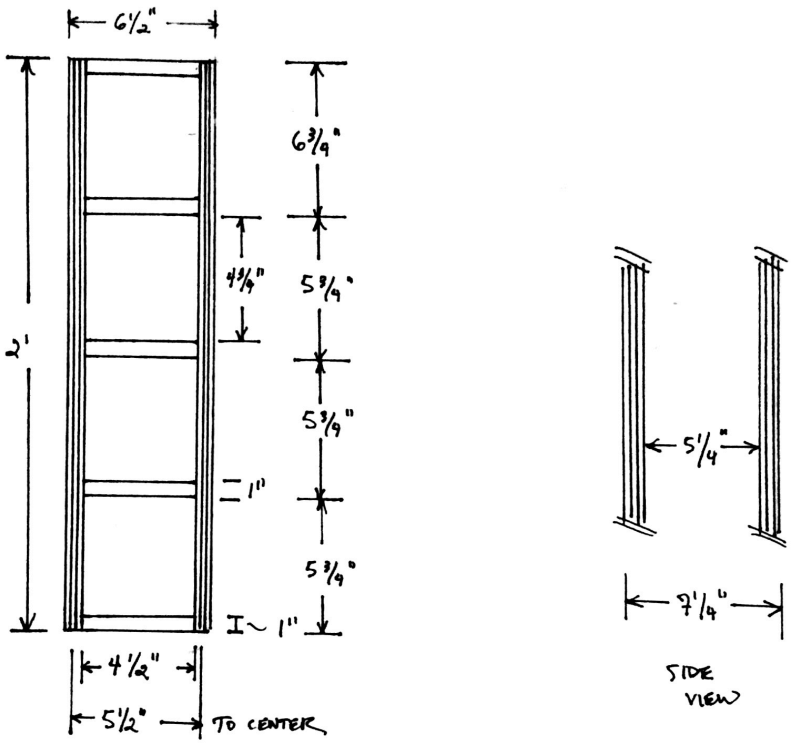
The original sketches of our design:



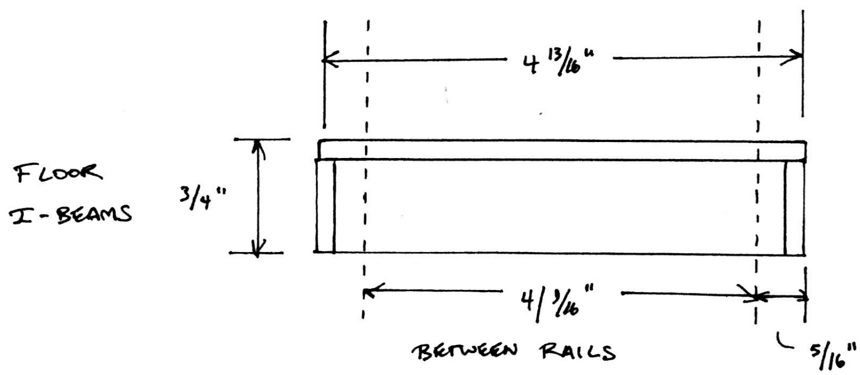
II. Design and Implementation

Structure:

This design started with a series of mechanical drawings for the structure:

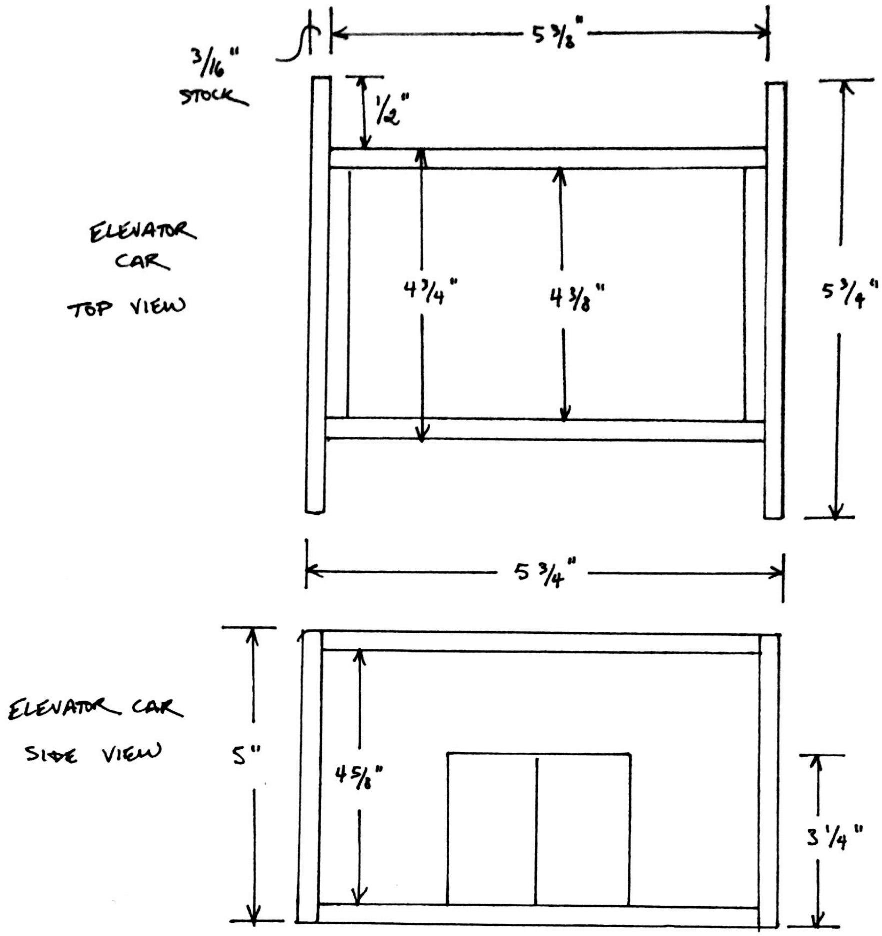


We used 1” aluminum extrusion for the main verticals, since the extrusion already has channels cut into it that serve as a track for the elevator. The elevator car and divisions between floors are made of 3/16” foam board, which is rigid and light. The divisions between floors were designed as I-beams that form a pressure fit wit the channels in the vertical members. These clearly define the four floors in the structure.



Elevator Car:

The elevator was designed to fit in the rails tightly but with enough room to avoid friction against the tower:



The top of the tower was redesigned to 7.25” deep to accommodate the electronics, MSP430, battery, motor, and gearbox. The original square structure was simply too small to accommodate all of our components.

Gearbox and Pulley:

After experimenting with motor speed and pulley dimensions, we settled on a 125:1 gear ratio for our 3V motor and gearbox, made by Cebek. This gave us adequate speed at 30-40% duty cycle and enough torque to overcome gravity and friction in the elevator system. We used 30lb-test monofilament line as our main elevator cable. The speed we selected is ultimately a tradeoff between ascension/rate and response time. Higher speeds cause the elevator car to travel more than an inch past the limit switch on each selected floor before the motor stops rotating. This is mainly due to the inertia of the car, but can be mitigated by using a lower but still adequate speed.

Motor and Controller:

Our motor is driven by one SN754410NE h-bridge motor driver. This is powered using 6V directly from the battery. We use 40% duty cycle when ascending and 30% duty cycle when descending. The extra power during ascension is needed to overcome gravity. As the battery wears down, duty cycle must be increased to 50/40%, then 60/50% up/down to provide the same desired speed. Currently we have no automatic mechanism to adjust this.

Priority Encoders:

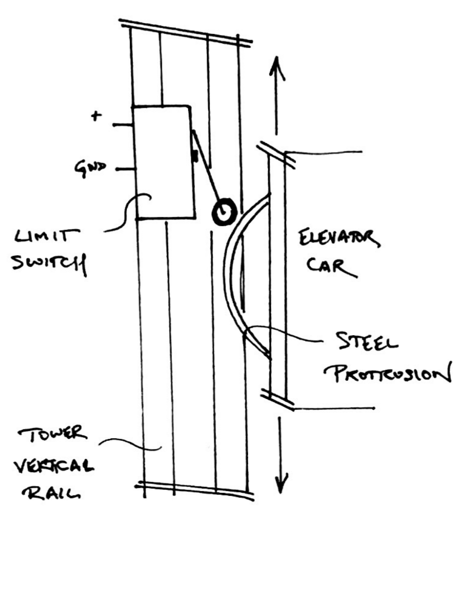
Since we have 20 total I/O devices and only 16 available I/O pins on the MSP430, we used three 74LS148N priority encoders to encode all of our inputs (on-structure buttons, in-elevator buttons, and limit switches) into a series of enable signals (indicating that a button was pressed) and addresses (indicating which button/switch was pressed). All encoders are powered by 3.3V provided by the Launchpad. This is to ensure that any logic signals sent to the MSP430 do not exceed Vcc. Although this is at the bottom limit of their operating voltage, the encoders perform well.

Seven-Segment Display:

To add a visual element and to indicate which floor the elevator is currently on, we used a large common-anode seven-segment display, driven by a 74LS47 BCD to 7-Segment decoder. This displays the current floor the elevator is on, determined by the last limit switch depressed on the tower (the last known location of the elevator car). We chose to drive the display and driver using 6V for better brightness and contrast.

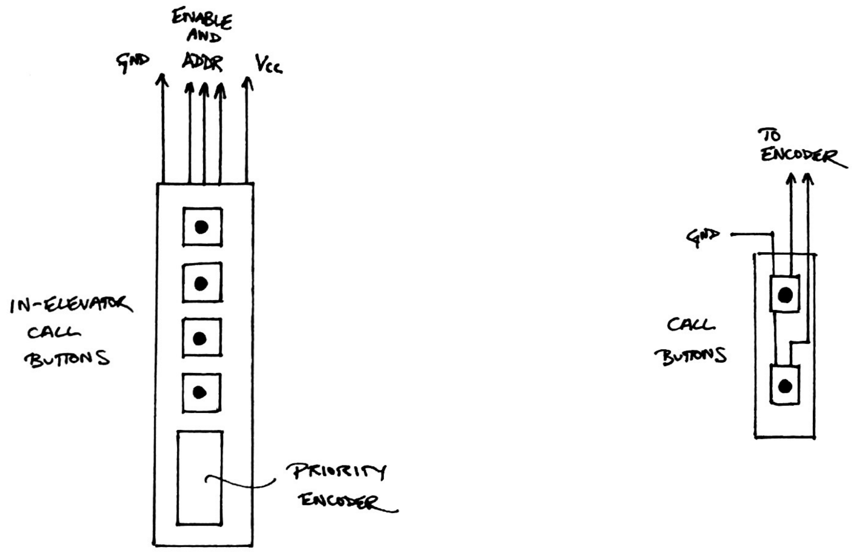
Limit Switches:

We use four SS-5GL2 bi-directional, roller limit switches to detect the absolute position of the elevator car during travel. These provide feedback to the controller about elevator position, contacting the elevator on a metal cam that protrudes from the back of the car. Supplemental photos of this mechanism are provided below.



On-Structure Call Buttons:

Six buttons attached to the tower allow the user to call the elevator car to each floor. These buttons are used by passengers *outside* of the elevator who wish to enter it. The first and fourth floors have only one button each (up and down respectively). The second and third floors have two buttons each, allowing the user to select either an upward or downward destination.



In-Elevator Call Buttons:

Once the elevator is called to a floor, the passenger enters the elevator and selects his desired destination (floors 1-4). This selection is restricted by the passenger's direction of travel. If going up, only floors above the current floor may be selected. If going down, only floors lower than the current floor may be selected.

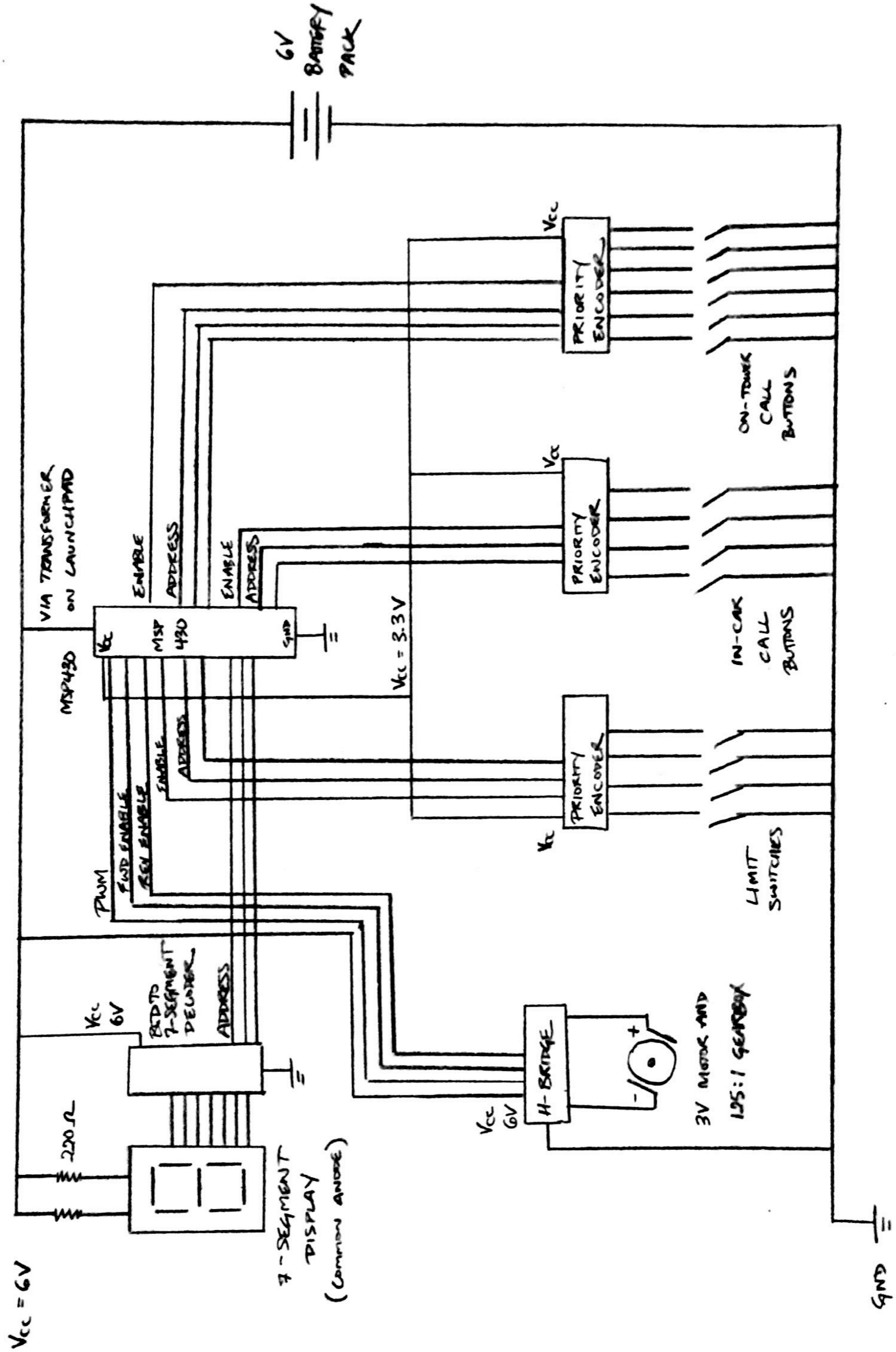
6V Battery and Power:

We use a 6V, AA battery pack to power our elevator. This directly powers the motor driver and seven-segment display. The battery also indirectly powers the encoders and MSP430 through the Launchpad's onboard transformer, originally designed to convert 5V USB power into 3.3V for the controller. When idle, the control system draws 60-100mA. When the motor is running, the whole system draws 400mA. This depletes the 6V battery pack quicker than we would like. Overall we are conservative with our power usage. We run the MSP430 with a 1MHz clock and 8K WDT divisor. This wakes the CPU up every 8ms. This is relatively low power but still fast enough for our needs.

Initialization:

When the system starts, the elevator position is unknown. The elevator is automatically lowered to the first floor to verify its position before accepting user input.

III. Circuit Schematic



IV. Assessment of Success

We feel that this project was entirely successful. Our elevator performs reliably and as expected. Our hardware interfaces well with the MSP430 and all I/O devices. Our software is comprehensive and safely guards against user and system errors. Overall we are thrilled with our final product.

V. Next Steps

As our battery wears down, voltage in the system decreases. Fresh batteries start at around 6.3V and decay to 5V over time. When voltage decays, our motor speed slows. Currently, we manually adjust the duty cycle of the motor to maintain speed on weak batteries. In the future we would like to detect battery voltage using the ADC and dynamically adjust duty cycle as the voltage decays.

Currently our battery life is poor. A 6V set of AA batteries lasts only 6-8 hours under normal operation. Reducing power consumption or using a larger, rechargeable battery (with more mAh storage) could also improve battery life and longevity.

Our current control algorithm is basic. Only one passenger may call the elevator to a floor and ride it to a destination at any time. Priority is granted on a first-come, first-serve basis. In the future we would like to develop a better control algorithm that allows multiple passengers to call the elevator simultaneously, prioritizing stops when ascending and descending.

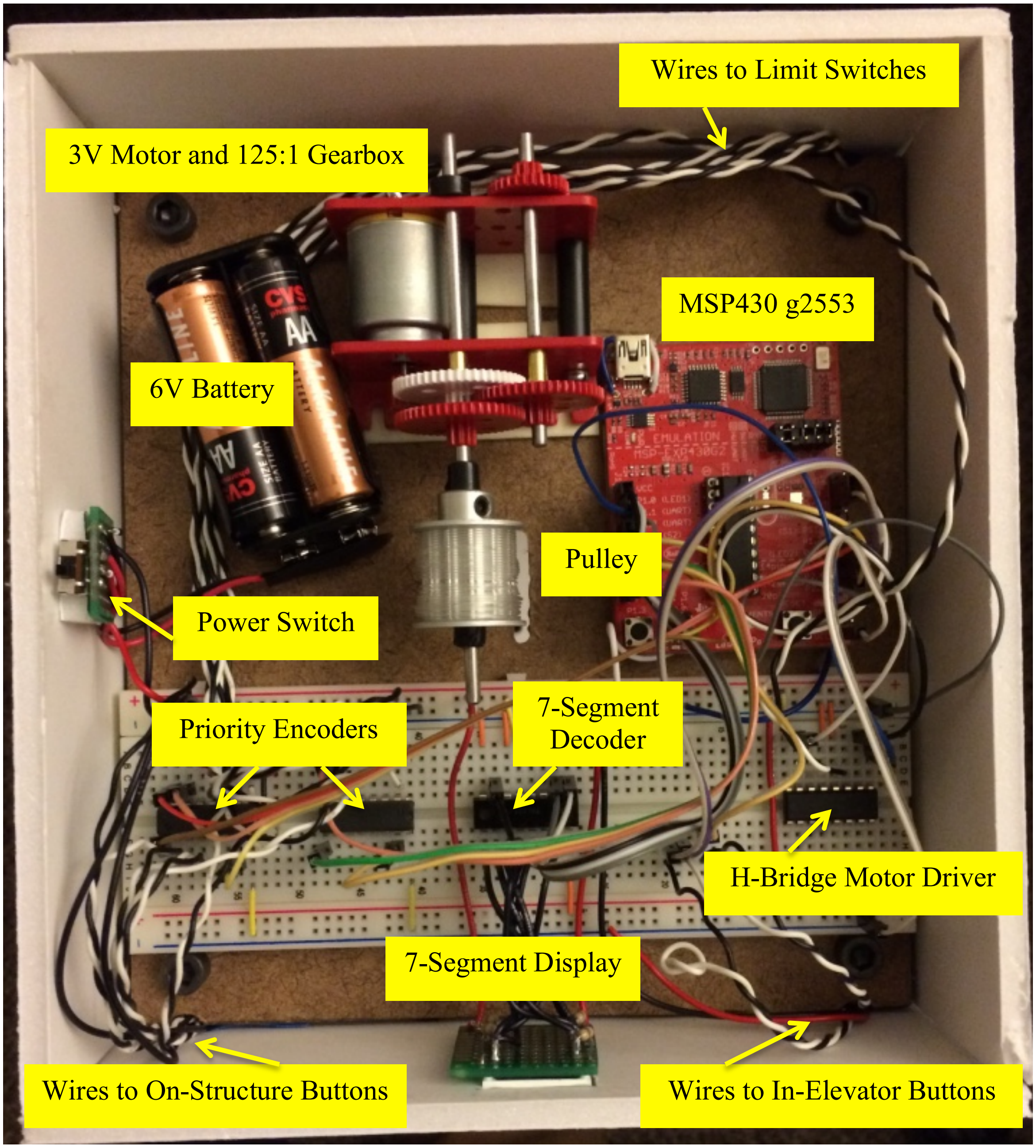
When the car reaches its destination there is a small delay between when the limit switch is depressed and the motor stops turning. This causes the car to travel past its stopping point by a quarter inch in either direction. Dynamically reducing motor speed as the elevator nears its destination could reduce this effect.

VI. Summary of Contributions

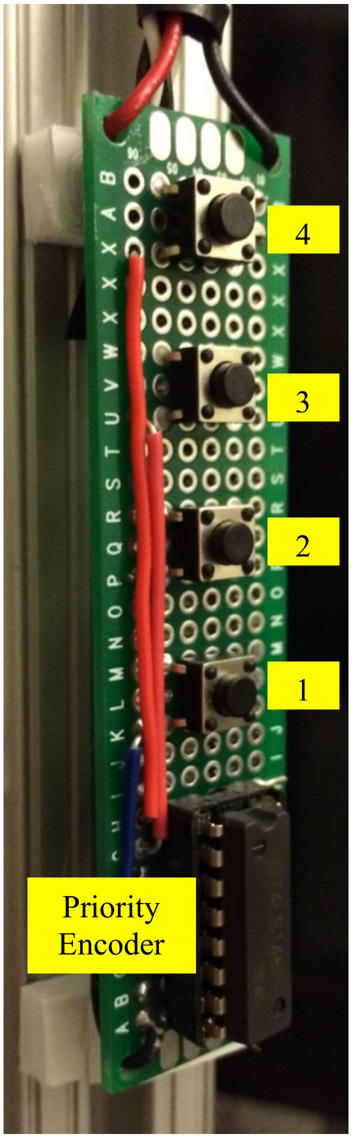
Carlton Duffett designed the system and made all mechanical drawings and circuit schematics. Neeraj Basu implemented the h-bridge motor driver and fabricated the aluminum structure. Both group members contributed equally to all other aspects of this project, including construction, wiring, hardware implementation, software implementation, and debugging.

VII. Supplemental Photos

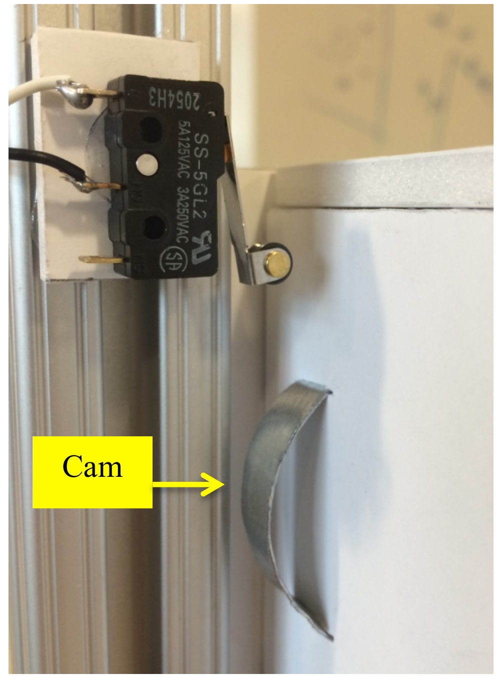
Our full control system, installed on the roof of the elevator:



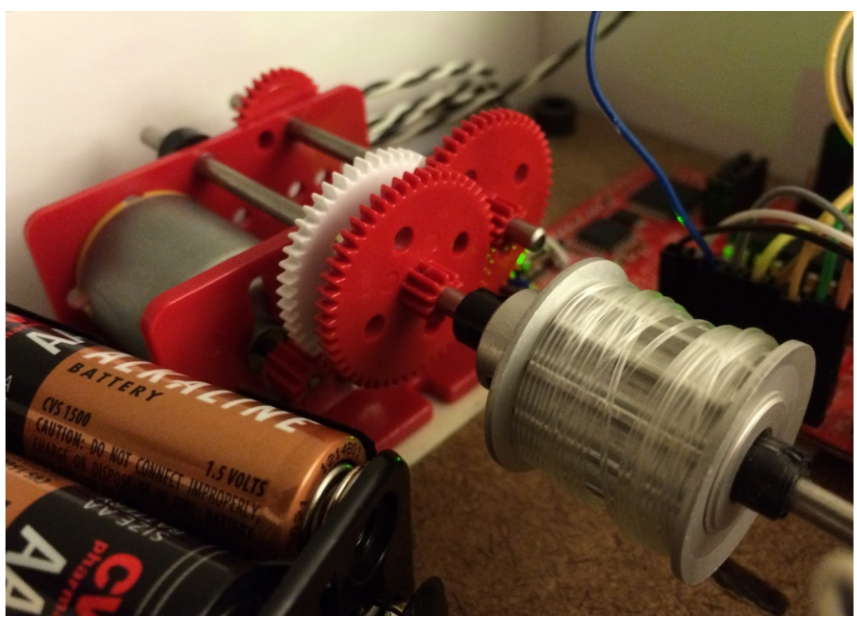
In-elevator call buttons, with priority encoder:



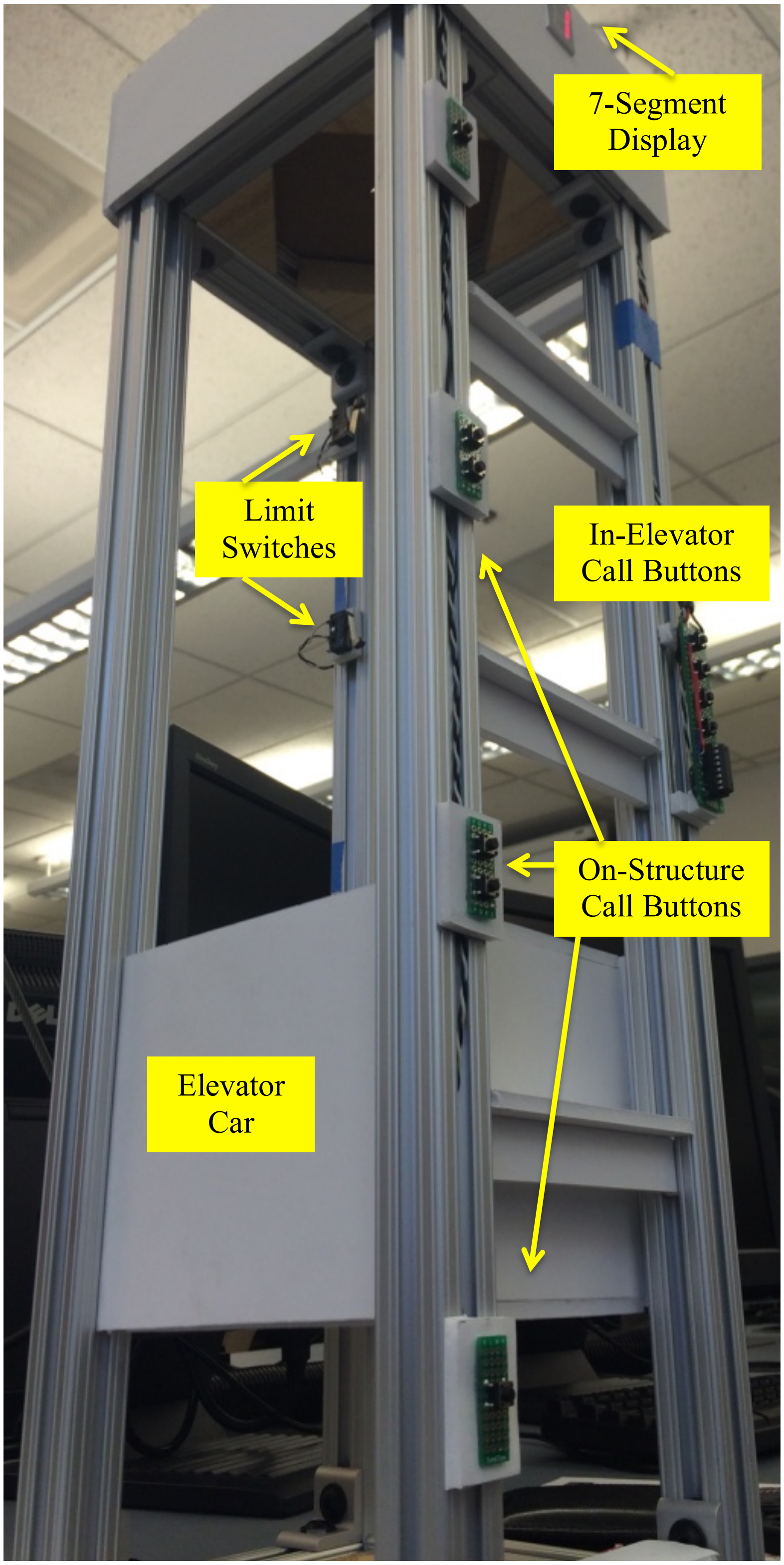
Limit switch and cam on elevator car:



3V motor, 125:1 gearbox, and pulley:



Full structure:



VIII. Code

```
#include <msp430g2553.h>

/*
 * Elevator Control System
 * -----------------------
 * Carlton Duffett
 * Neeraj Basu
 *
 * EC450 Final Project
 * Boston University
 * Spring 2015
 *
 * This program controls a 4-floor elevator system. The control hardware consists
 * of the following devices:
 *
 * 1. 1x 3V motor and 125:1 gearbox
 * 2. 1x SN754410NE Dual H-Bridge Driver
 * 3. 1x 74LS247 BCD to Seven-Segment Decoder
 * 4. 3x 74LS148 Priority Encoder
 * 5. 1x MSP430g2553 Microcontroller
 *
 * The priority encoders encode all call buttons and limit switches on the structure.
 * These are appropriately prefixed:
 *
 * TOWER_   On-tower call buttons that user presses to call the elevator car to each floor
 * ELEV_    In-elevator call buttons that user presses to select desired destination
 * LIMIT_   On-tower limit switches that detect the absolute position of the elevator car
 *
 * This system has a very basic control algorithm. The elevator may be called to only one
 * floor and sent to only one destination at a time. Future versions will have a more
 * sophisticated control scheme.
 *
 * Because of the way the priority encoders work, the P1 and P2 interrupts cannot be used
 * to detect button presses. Polling by the Watchdog Timer (WDT) is used instead.
 *
 * The possible states of the system are:
 *
 * 'i'  - initializing elevator (on reset the car defaults to the first floor)
 * 'x'  - idle, waiting to be called to a floor
 * '^'  - going up to a called floor to receive a passenger
 * 'v'  - going down to a called floor to receive a passenger
 * 'w'  - waiting at called floor for user to select destination
 * 'u'  - going up to selected destination with a passenger
 * 'd'  - going down to selected destination with a passenger
 *
 */

// port 1 bit mask
#define SEVENSEG_A0     0x01    // seven segment display addresses
#define SEVENSEG_A1     0x02
#define PWM             0x04    // pulse-width modulation for motor control
#define SEVENSEG_A2     0x08
#define TOWER_EN        0x10    // on-tower call buttons, enable
#define TOWER_A0        0x20    // on-tower call buttons, addresses
#define TOWER_A1        0x40
#define TOWER_A2        0x80
```

```
// port 2 bit mask
#define LIMIT_EN        0x01    // limit switches, enable
#define LIMIT_A0        0x02    // limit switches, addresses
#define LIMIT_A1        0x04
#define ELEV_EN         0x08    // in-elevator buttons, enable
#define ELEV_A0         0x10    // in-elevator buttons, addresses
#define ELEV_A1         0x20
#define UPCTL           0x40    // up direction selection for motor control
#define DNCTL           0x80    // down direction selection for motor control

// state variables
volatile unsigned char state = 'i';         // state of the system
volatile unsigned char current_floor = 0;   // current location of elevator car
volatile unsigned char called_floor;        // floor elevator was called to
volatile unsigned char destination;         // floor that user selects as destination
volatile unsigned char dest_direction;      // direction (up/down) that user's destination is
in

// initialization functions
void init_motor_control(void);
void init_limit_switches(void);
void init_elev_buttons(void);
void init_tower_buttons(void);
void init_timerA(void);
void init_7segment(void);
void init_WDT(void);

// motor control functions
void stop_motor(void);
void go_up(void);
void go_down(void);

// duty cycle settings for up/down (out of 1000)
#define UP_DUTY_CYCLE   400 // 40 %
#define DN_DUTY_CYCLE   300 // 30 %

// control handlers
void update_display(unsigned char floor);
unsigned char get_tower_addr(void);
unsigned char get_elev_addr(void);
unsigned char get_limit_addr(void);

void handle_tower_button(unsigned char addr);
void handle_elev_button(unsigned char addr);
void handle_limit_switch(unsigned char addr);

// =============== MAIN PROGRAM ===============
int main(void) {

    // 1Mhz calibration for SMCLK clock
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL  = CALDCO_1MHZ;

    // initialize the system
    init_motor_control();
    init_limit_switches();
    init_elev_buttons();
    init_tower_buttons();
    init_7segment();
    init_timerA();
    init_WDT();
```

```
    // turn off CPU and enable interrupts
    _bis_SR_register(GIE+LPM0_bits);
}

// ================ INITIALIZATION FUNCTIONS ================

// initialize the motor control signals and PWM
void init_motor_control(void) {

    // setup motor PWM port
    P1DIR |= PWM;
    P1SEL |= PWM;

    // setup direction control
    P2DIR |= UPCTL;
    P2DIR |= DNCTL;
    P2SEL &= ~UPCTL; // disconnect from XOUT
    P2SEL &= ~DNCTL; // disconnect from XIN

    // setup default PWM length (50%)
    TA0CCR1 = 500;    // on for 8/16 cycles
    TA0CCR0 = 999;    // off for 8/16 cycles
}

// initialize limit switches to monitor elevator position
void init_limit_switches(void) {

    // EN indicates that a switch was pressed
    // A0 - A1 indicates which switch was pressed
    P2DIR &= ~LIMIT_EN;
    P2DIR &= ~LIMIT_A0; // 4 limit switches = 2 bit address
    P2DIR &= ~LIMIT_A1;
}

// initialize in-elevator call buttons for user to select desired floor
void init_elev_buttons(void) {

    P2DIR &= ~ELEV_EN;
    P2DIR &= ~ELEV_A0;  // 4 call buttons = 2 bit address
    P2DIR &= ~ELEV_A1;
}

// initialize on-tower call buttons for user to call elevator to a floor
void init_tower_buttons(void) {

    P1DIR &= ~TOWER_EN;
    P1DIR &= ~TOWER_A0; // 6 tower buttons = 3 bit address
    P1DIR &= ~TOWER_A1;
    P1DIR &= ~TOWER_A2;
}

// initialize timer A to drive a PWM signal
void init_timerA(void) {

    TA0CTL |= TACLR;        // reset clock
    TA0CTL |= (TASSEL_2 +   // clock source = SMCLK
               ID_0 +       // clock divider = 1
               MC_1);       // UP mode

    TA0CCTL1 |= OUTMOD_7;   // reset/set mode
```

```
}

// initialize the seven-segment display
void init_7segment(void) {

    // 3-bit address to drive the correct display number
    P1DIR |= SEVENSEG_A0;
    P1DIR |= SEVENSEG_A1;
    P1DIR |= SEVENSEG_A2;
}

// initialize the watchdog timer
void init_WDT(void) {

    // setup as an interval timer
    WDTCTL =  (WDTPW +    // password
               WDTTMSEL + // select interval timer mode
               WDTCNTCL + // clear watchdog timer counter
               0 +        // SMCLK is the source
               1);        // source/8k

  // enable the WDT interrupt (in the system interrupt register IE1)
  IE1 |= WDTIE;
}

// =============== MOTOR CONTROL FUNCTIONS ===============

void stop_motor(void) {
    // set motor to stop mode
    P2OUT |= UPCTL;
    P2OUT |= DNCTL;
}

void go_up(void) {

    // set motor control signal to UP
    P2OUT |= UPCTL;
    P2OUT &= ~DNCTL;

    // use higher duty cycle in up direction
    TA0CCR1 = UP_DUTY_CYCLE;
}

void go_down(void) {

    // set motor control signal to DN
    P2OUT &= ~UPCTL;
    P2OUT |= DNCTL;

    // use lower duty cycle in down direction
    TA0CCR1 = DN_DUTY_CYCLE;
}

// =============== 7-SEGMENT DISPLAY ===============
void update_display(unsigned char floor) {

    if (floor == 1) {
        // 0b001
        P1OUT |= SEVENSEG_A0;
        P1OUT &= ~SEVENSEG_A1;
        P1OUT &= ~SEVENSEG_A2;
```

```
    }
    else if (floor == 2) {
        // 0b010
        P1OUT &= ~SEVENSEG_A0;
        P1OUT |= SEVENSEG_A1;
        P1OUT &= ~SEVENSEG_A2;
    }
    else if (floor == 3) {
        // 0b011
        P1OUT |= SEVENSEG_A0;
        P1OUT |= SEVENSEG_A1;
        P1OUT &= ~SEVENSEG_A2;
    }
    else if (floor == 4) {
        // 0b100
        P1OUT &= ~SEVENSEG_A0;
        P1OUT &= ~SEVENSEG_A1;
        P1OUT |= SEVENSEG_A2;
    }
}

// ================ CONTROL HANDLERS ================

// address masks
#define TOWER_ADDR_MASK 0xE0
#define ELEV_ADDR_MASK  0x30
#define LIMIT_ADDR_MASK 0x06

// get the current address of the on-tower button that was pressed
unsigned char get_tower_addr(void) {

    // right shift the address bits into LSB position
    return ((P1IN & TOWER_ADDR_MASK) >> 5);
}

// get the current address of the in-elevator button that was pressed
unsigned char get_elev_addr(void) {

    return ((P2IN & ELEV_ADDR_MASK) >> 4);
}

// get the current address of the limit switch that was pressed
unsigned char get_limit_addr(void) {

    return ((P2IN & LIMIT_ADDR_MASK) >> 1);
}

// on-tower call button addresses
#define F1_UP   0x7 // floor 1, up button
#define F2_DN   0x6 // floor 2, down button
#define F2_UP   0x5 // .. etc
#define F3_DN   0x4
#define F3_UP   0x3
#define F4_DN   0x2

// handles a call event requesting the elevator to a specific floor
// to be called, the elevator must currently be idle
void handle_tower_button(unsigned char addr) {

    switch (addr) {
```

```
// First floor, up button
case F1_UP:

    if (state == 'x') { // elevator is currently idle

        called_floor = 1;
        dest_direction = 'u';

        // get current position of elevator and signal movement
        if (current_floor != 1) {
            go_down();
            state = 'v'; // going down to called floor
        }
        else {
            state = 'w'; // waiting for floor selection
        }
    }

    break;

// second floor, down button
case F2_DN:

    if (state == 'x') { // elevator is currently idle

        called_floor = 2;
        dest_direction = 'd'; // elevator's destination is down

        if (current_floor != 2) {

            if (current_floor > 2) {
                go_down();
                state = 'v';
            }
            else {
                go_up();
                state = '^';
            }
        }
        else {
            state = 'w';
        }
    }
    break;

// second floor, up button
case F2_UP:

    if (state == 'x') { // elevator is currently idle

        called_floor = 2;
        dest_direction = 'u'; // elevator's destination is up

        if (current_floor != 2) {

            if (current_floor > 2) {
                go_down();
                state = 'v';
            }
            else {
                go_up();
```

```
                state = '^';
            }
        }
        else {
            state = 'w';
        }
    }
    break;

// third floor, down button
case F3_DN:

    if (state == 'x') { // elevator is currently idle

        called_floor = 3;
        dest_direction = 'd';

        if (current_floor != 3) {

            if (current_floor > 3) {
                go_down();
                state = 'v';
            }
            else {
                go_up();
                state = '^';
            }
        }
        else {
            state = 'w';
        }
    }
    break;

// third floor, up button
case F3_UP:

    if (state == 'x') { // elevator is currently idle

        called_floor = 3;
        dest_direction = 'u';

        if (current_floor != 3) {

            if (current_floor > 3) {
                go_down();
                state = 'v';
            }
            else {
                go_up();
                state = '^';
            }
        }
        else {
            state = 'w';
        }
    }
    break;

// fourth floor, down button
case F4_DN:
```

```
        if (state == 'x') { // elevator is currently idle

            called_floor = 4;
            dest_direction = 'd';

            if (current_floor != 4) {

                go_up();
                state = '^';
            }
            else {
                state = 'w';
            }
        }
        break;
    } // switch
}

// in-elevator call button addresses
// currently unused
#define F1_SELECTED   0x00
#define F2_SELECTED   0x10
#define F3_SELECTED   0x20
#define F4_SELECTED   0x30

// handles a call event where the elevator passenger selected a destination floor
void handle_elev_button(unsigned char addr) {

    destination = addr + 1; // valid destinations are 1 - 4

    if (state == 'w') { // waiting for user to select destination

        if (destination == current_floor) {
            state = 'w'; // already at destination
        }
        else if (dest_direction == 'u' && (destination > current_floor)) {

            state = 'u'; // going up with passenger
        }
        else if (dest_direction == 'd' && (destination < current_floor)) {

            state = 'd'; // going down with passenger
        }
    }
}

// limit switch addresses
// currently unused
#define LIMIT_1    0x00
#define LIMIT_2    0x01
#define LIMIT_3    0x02
#define LIMIT_4    0x03

// handles the event where a limit switch on the tower is depressed, indicating elevator
position
void handle_limit_switch(unsigned char addr) {

    current_floor = addr + 1; // valid floors are 1 - 4


    if (current_floor == 1 || current_floor == 4) {
```

```
        stop_motor(); // redundant, ensure elevator does not travel past structural limits
    }
    update_display(current_floor);
}

// =============== WDT INTERRUPT HANDLER ===============

interrupt void WDT_interval_handler() {

    // poll the sensors to check for user input
    if (P2IN & LIMIT_EN) {

        // limit switch depressed
        handle_limit_switch(get_limit_addr());
    }
    if (P2IN & ELEV_EN) {

        // in-elevator button pressed
        handle_elev_button(get_elev_addr());
    }
    if (P1IN & TOWER_EN) {

        // on-tower button pressed
        handle_tower_button(get_tower_addr());
    }

    // handle system state
    switch (state) {

    case 'i': // initialize elevator position, runs only at power-on

        if (current_floor == 1) {

            // elevator initialized to first floor, ready for service
            stop_motor();
            state = 'x';
        }
        else {

            // initially send elevator to first floor
            go_down();
        }
        break;

    case 'x': // elevator idle

        // do nothing for now
        stop_motor();
        break;

    case '^': // *up arrow* going up to called floor

        if (called_floor == current_floor) {
            stop_motor();
            state = 'w';
        }
        else {
            go_up();
        }
        break;
```

```
    case 'v': // *down arrow* going down to called floor

        if (called_floor == current_floor) {
            stop_motor();
            state = 'w';
        }
        else {
            go_down();
        }
        break;

    case 'w':

        // waiting for user input
        stop_motor();
        break;

    case 'u': // going up with passenger

        if (destination == current_floor) {
            stop_motor();
            state = 'x';
        }
        else {
            go_up();
        }
        break;

    case 'd': // going down with passenger

        if (destination == current_floor) {
            stop_motor();
            state = 'x';
        }
        else {
            go_down();
        }
        break;

    } // switch
}
ISR_VECTOR(WDT_interval_handler, ".int10")
```