The Automated Haunted House Model

Service Manual

EE-490
Electrical Engineering Senior Design Project (Capstone)

Lance Bredthauer
Ernest Estes
Isaac Frampton
Jven-Jay Marie

Fall Term 2006
December 13, 2006

TABLE OF CONTENTS

I. Set Up & Operation

Setup

To insure best viewing experience please follow the following set up guidelines:

1) Set up house on a sturdy, non-slip surface that allows audience to move freely around house during operation. *note: a quiet, dimly lit environment is best.
2) Check to make sure internal supply is switch off, then plug into a standard 120VAC outlet.
3) Ensure that backdrop is completely secured to rear of house.

Operation

House is completely automated. Simply complete set up and switch on directly at power supply located in attic and replace backdrop. The house will immediately commence its homing sequence, which takes approximately 5 seconds. During this time, servos, motor and/or lights may become active; do not disturb their operation as it may interfere with the homing process.

After homing is complete, the house will begin its normal operation sequence. Actions can be followed most effectively by focusing attention on room lighting, as lighting indicates location of next action. Following each complete cycle of actions (immediately following lighting/thunder sound sequence), the house will again enter a short re-initialization sequence prior to resuming normal operation.

II. Description of Sequence

1. Power is applied
    a. All devices are "homed"
        i. No music is played
        ii. All lights are turned off
        iii. Skeleton is raised to top of tower
        iv. Fireplace is turned towards house front
        v. Front door is closed
        vi. "Face" is flush with roof surface
        vii. Fan is not turning
2. First track starts (Welcome)
    a. Front porch light on
    b. Door opens (with sound)
    c. Room light on
    d. Door closes
3. Next track starts (Noose falls from Tower)
    a. Room lights at base of Tower
    b. Noose and skeleton falls from Tower (with sound)
    c. Front light turns on
    d. Tower light turns off
4. Next track starts (eerie porch sequence)
    a. Front door opens
    b. Blue lights on porch
    c. Front door closes (with sound)
    d. Blue lights off
5. Next track starts (Fireplace)
    a. Room light on → off
    b. Fireplace LED's flicker (with sound)
    c. Fireplace rotates to expose secret passage (with sound)
6. Next track starts (Execution)
    a. Fireplace room light on → off
    b. 1st floor execution room light on → off
    c. 2nd floor execution room light on (bells chime 3rd time)
    d. Lights strobe over electric chair (with sound)
    e. Lights off
7. Next track starts (Face in Shingles)
    a. Face pops out of roof (with sound)
    b. Red LED's illuminate eyes and mouth
    c. Face goes flush with roof
8. Next track starts (Storm)
    a. Fan and red LED turns on (with sound)
9. Homing sequence & cycle repeat
    a. Returns to Section 2

III. Bill of Materials

Quantity	Part Number	Manufacturer or Supplier	Description	Where Used
3	PIC18F4550	Microchip	40-pin microcontroller	Main Processor Serial Lighting Fireplace
1	PIC18F2520	Microchip	28-pin microcontroller	Basement
1	Z250	Generic	180deg servo gearmotor	Front Door
1	Z350	Generic	Discrete 180deg servo gearmotor w/steel gears	Fireplace Rotation
45	IRLZ24PBF-ND	Digi-Key	17 AMP N-CH MOSFET	All motors, servos, 12V lights, MP3 Control
20	IRFU5410-ND	Digi-Key	13 AMP P-CH MOSFET	Reversing motor driver, electric chair driver circuit, MP3 player controller
11	441-1009-ND	Digi-Key	LED, 10mm White	Porch lights and all house lights
1		Radio Shack	LED, 10mm Red	Roof face features
1		Radio Shack	DPDT 12V relay	Directional control for Roof face control
1	MP-C827	Coby Electronic / K-Mart	128MB Flash MP3 Player	Sound system (main board)
2	TDA7273A	SGS Thompson	Power Amplifier IC	Sound System (behind subwoofer)

IV. Audio System

Introduction

House features a 4 channel audio system consisting of three full-range drivers and a 5 inch mini-subwoofer for bass frequency reinforcement.

Design

Audio system is based off the Coby Electronic MP-C827 flash-based MP3 player and the SGS-THOMPSON TDA7273A power amplifier IC. Audio content is downloaded to player through USB port and operation is controlled via 3 control leads by the central processor. Audio signal is then filtered at inputs of amplifier board (located in rear of subwoofer enclosure) by a simple RC filter, then amplified through by the TDA7273A and outputted to the appropriate speakers.

Controls/Power

MP-C827 normally operates on 1.5VDC but in our application we apply approximately 1.3VDC through the power supply circuit (see schematic X-X). Power is applied during the initial homing sequence and cycled during the re-initialization process between cycles as a method of forcing a reset of the playlist.

Control

All control leads are switched via mosfet driving circuits on main processor board. Following activation of the supply circuit during the homing sequence, the play lead is held high for approximately 2 seconds to power on the player. At this point the player is ready, the next momentary high sent to the play lead will cause the player to begin its playlist, outputting audio to the amplifier. Sounds are synched with actions by advancing to the next track within the playlist.

Soundtrack

When loading new content, all contents of the mp3 player must be erased completely and new material must be loaded in sequence. Any adjustments to soundtrack will require individual adjustment of track queing in software.

V. Troubleshooting Guide

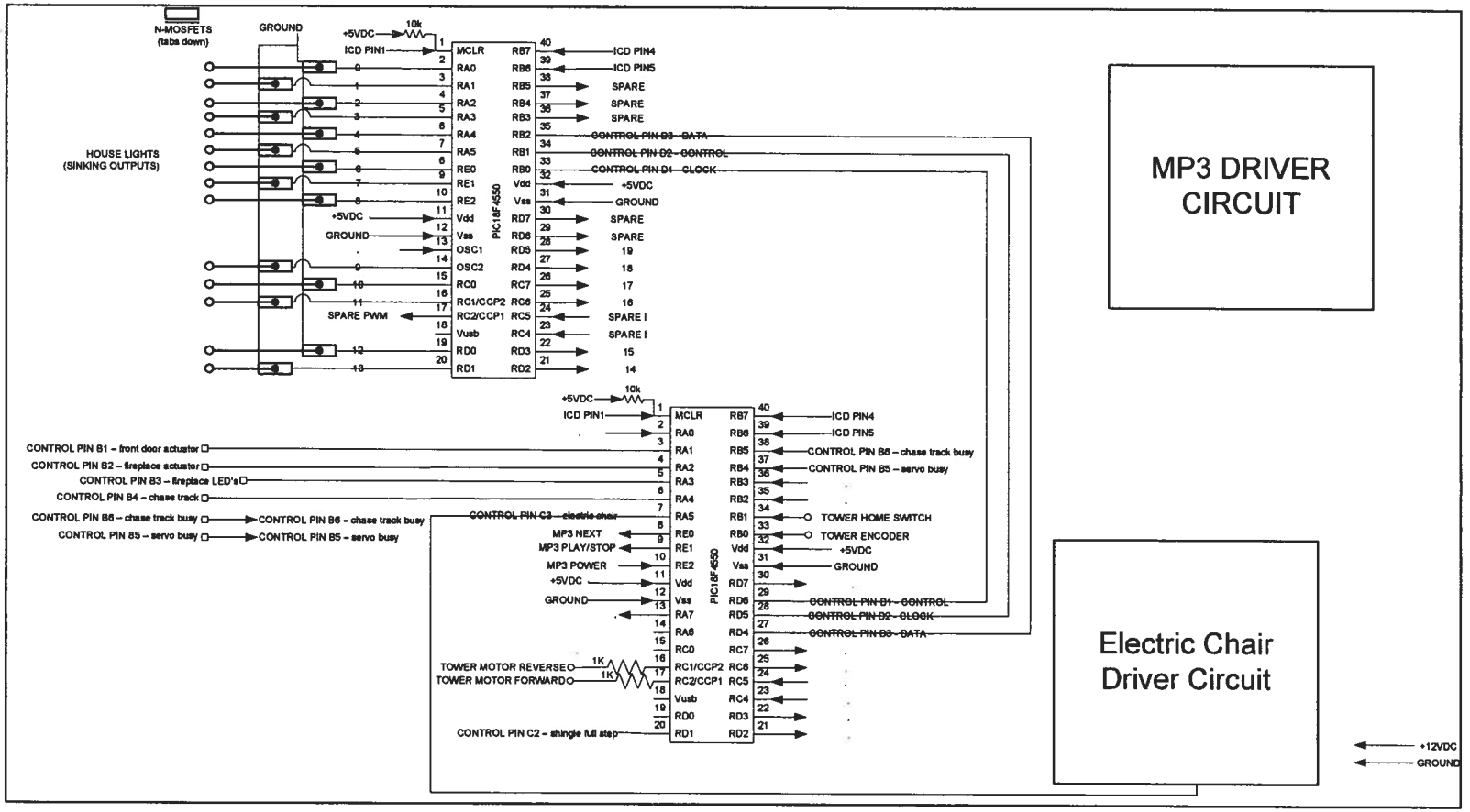
Please note that most problems with the haunted house can be repaired by interrupting the power supply to the house for 2 minutes. The following guide is to be referenced only after cycling power.

Problem	Possible Issues	Solutions
Sound doesn't play	MP3 player has lost power	Verify +5V source to Main board; verify 1.35VDC to the top center pin; make sure the player is firmly plugged into the socket on the board; make sure no FETs are at a high temperature in that area of the board; replace bad FETs
	MP3 player does not receive "play" command	Make sure the player is firmly plugged into the socket on the board; make sure no FETs are at a high temperature in that area of the board; replace bad FETs
	Speakers do not have 12V, 5V, or Ground	Verify voltages and wire connections on 12, 5, and ground on main board and connections behind subwoofer enclosure
	Signal wire between Speakers and MP3 player is loose	Ensure that the audio cable is firmly pressed into the socket of the MP3 player
	Main processor is not functioning	Reload program into main processor; replace processor
Room lights don't turn on	12V supply not correct	Verify +12V supply to Main board and common +12 rail; reconnect loose wires
	Lighting processor does not respond	Cycle power to system; reload program into processor; replace processor
	Signal wire from CPU to lighting processor is bad	Verify continuity to pins as assigned by the electrical schematics for the 3 control wires
	N-CH FETs are bad	Replace bad FETs
	Main processor is not functioning	Reload program into main processor; replace processor
Tower noose does not come down or retract	12V supply not correct	Verify +12V supply to Main board and remote driver board
	9-pin connector loose	Securely connect 9-pin connector to main board and remote driver board – Verify green power LED on remote driver board
	FETs are bad	Replace bad FETs

	Shaft encoder is not transmitting pulses to main processor	Verify free motion of flag through optical slot detector; monitor +0.5V to +4.5V signal change through Main board; replace optical slot detector
	Main processor is not functioning	Reload program into main processor; replace processor
	Homing limit switch has failed	Verify free mechanical motion of limit switch; monitor +5V to 0V signal change through Main board; replace limit switch
Front Door does not open or close	Transformer has operated (moving servo past designed limit)	Reach through lattice with a long, thin screwdriver or rod and manually move the servo to the clockwise extremity (closing the door)
	Faulty servo motor	Replace Servo motor if completely seized or unresponsive
	Servo motor does not have +5V supply	Verify +5V motor supply to basement processor board
	Loose 9-pin connector to basement processor	Securely connect 9-pin connector to main board and basement processor board – Verify green power LED on remote driver board
	Main processor is not functioning	Reload program into main processor; replace processor
	Mechanical linkages are out of alignment	Verify free mechanical motion of door action and secure connection to steel wire linkages; fabricate new linkages
	Basement processor is not functioning	Reload program into basement processor; replace processor
Fireplace LEDs do not light	+5V power is missing	Verify +5V power in basement board (Logic & motor); Connect Red wire to Pin 2 on first terminal block; black wire to Pin 6 on first terminal block
	Loose 9-pin connector to basement processor	Securely connect 9-pin connector to main board and basement processor board – Verify green power LED on remote driver board
	Main processor is not functioning	Reload program into main processor; replace processor
Fireplace fails to turn	Faulty servo motor	Replace Servo motor if completely seized or unresponsive
	Servo motor does not have +5V supply	Verify +5V motor supply to basement processor board
	Loose 9-pin connector to basement processor	Securely connect 9-pin connector to main board and basement processor board – Verify green power LED on

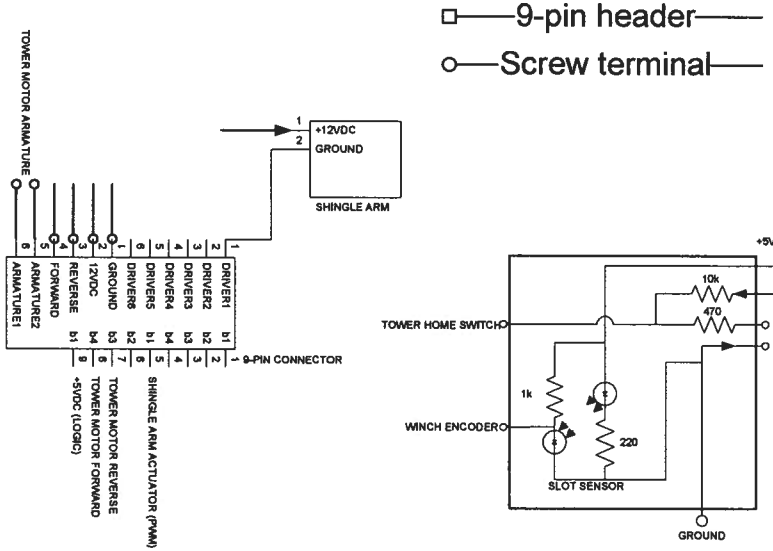
		remote driver board
	Main processor is not functioning	Reload program into main processor; replace processor
	Rotating disk is dismounted from servo	Reconnect rotating disk to servo motor with glue, screw, etc.
	Basement processor is not functioning	Reload program into basement processor; replace processor
Ghost fan does not rotate and/or red LED does not light	+12V supply is lost to fan	Verify +12V supply to Main board and common +12 rail; reconnect loose wires
	Serial lighting processor is not functioning	Cycle power to system; reload program into processor; replace processor
	Blockages prohibit fan from turning (if red LED is on)	Remove blockages
Face remains in or out and LEDs do not light	Serial Lighting processor is not functioning	Cycle power to system; reload program into processor; replace processor
	DPDT 12V relay is bad	Replace relay
	Obstruction prohibits motion (if red LED's are on)	Remove obstruction
	+12V supply is bad/loose	Verify +12V supply to Main board and common +12 rail; reconnect loose wires
Sound is incorrectly timed	MP3 player connection is loose	Firmly press 9 pin connector into main board
	Another mechanical action is obstructed (i.e. Tower Noose)	Resetting the system usually fixes this (if not see above)
House does not begin sequence and no LEDs are lit in attic	Power was cycled too quickly	Turn power off for 5 minutes before turning on
	Enable jumper wire is loose	Reconnect jumper on 20-pin connector (green to black)
	+12V/+5V/ground short exists	Check for continuity from 12V to 5V to ground; remove short
	Switch at power supply is off	Turn switch on
	Main processor is not functioning	Reload program into main processor; replace processor

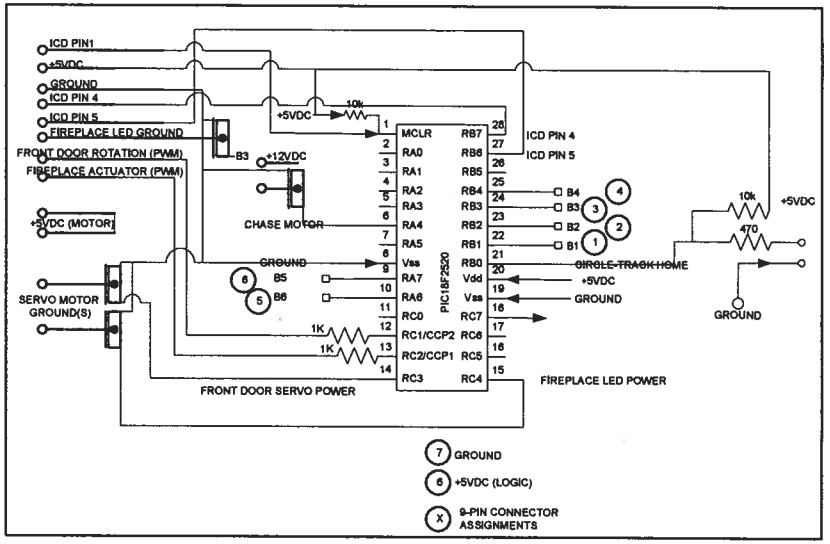
VI. Electrical Schematics



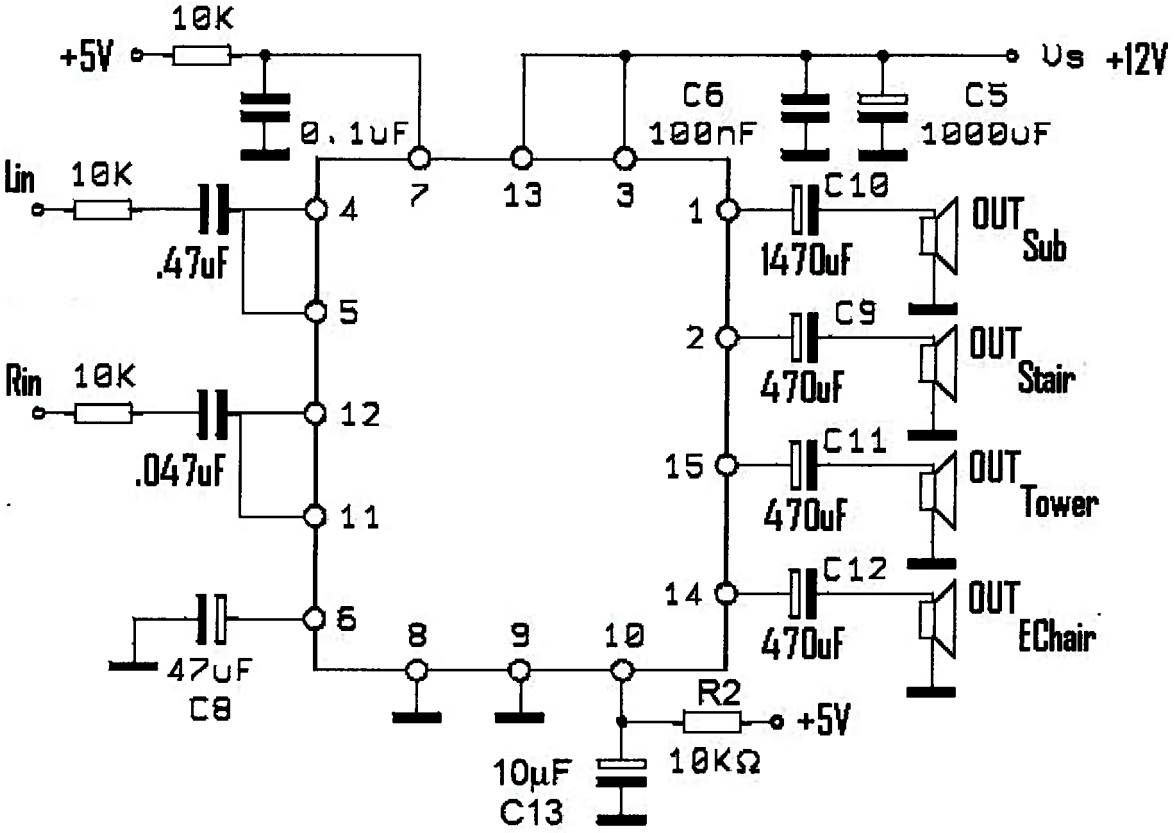
9-pin header

Screw terminal

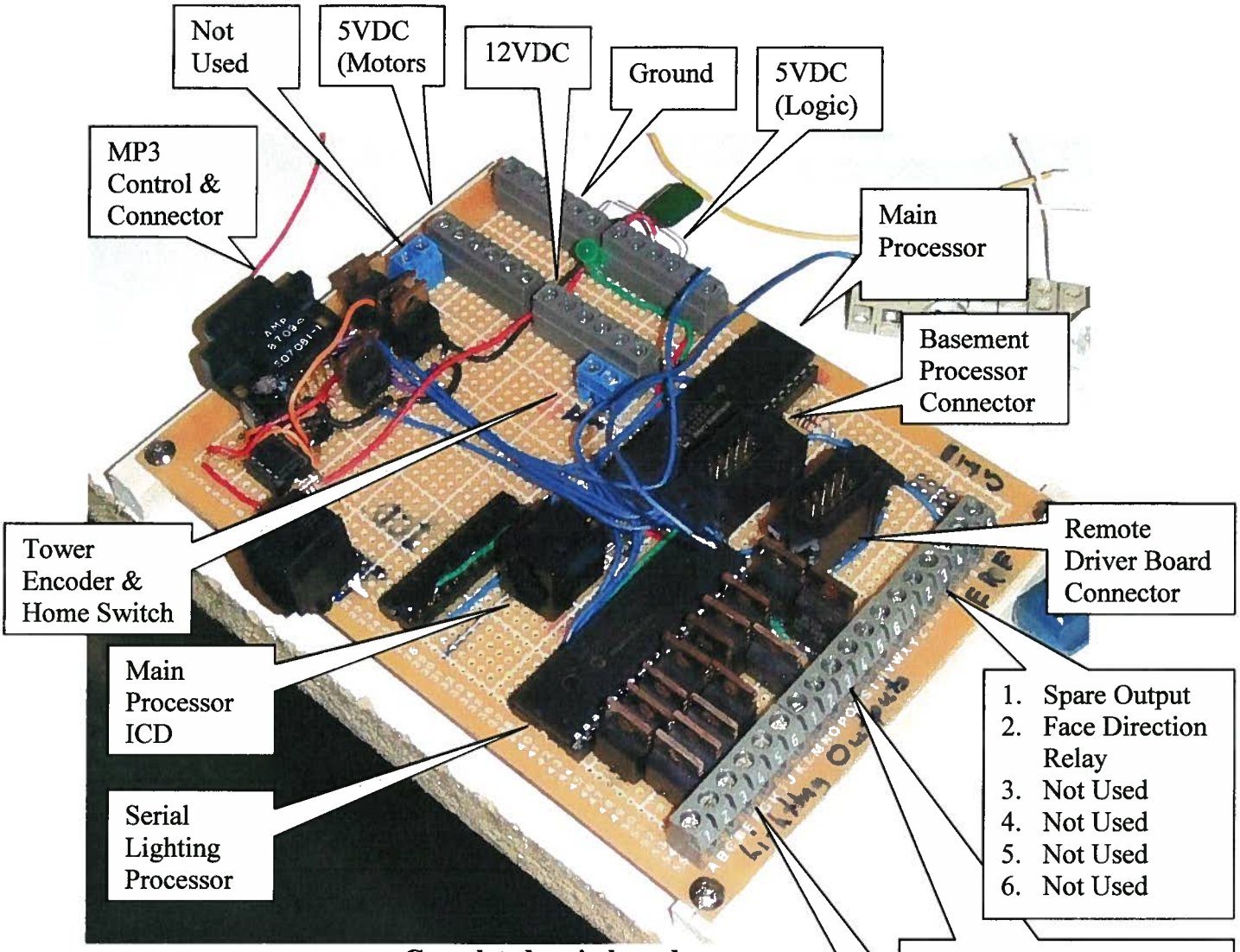




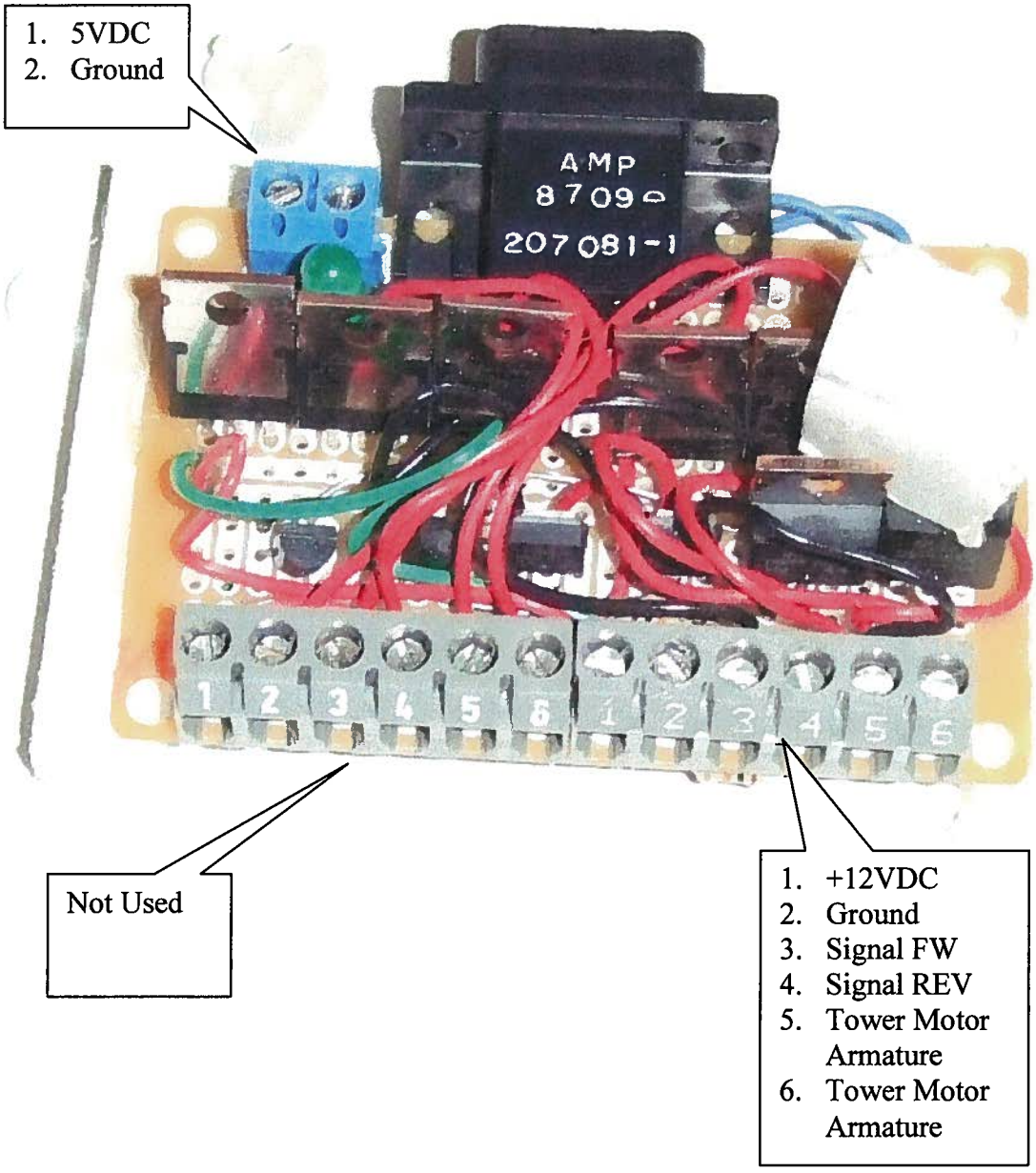
Audio Amplifier Schematics



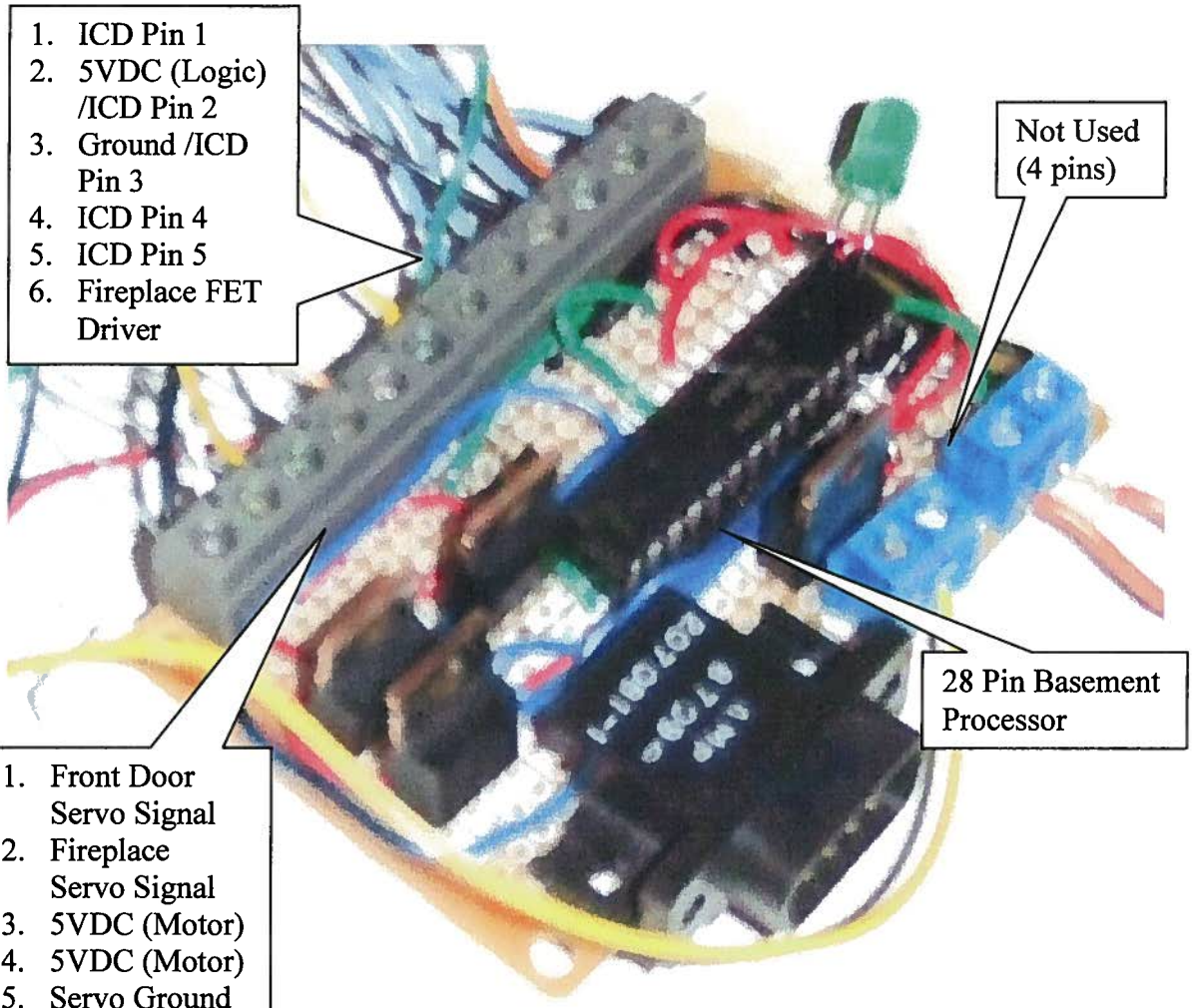
VII. Pictures and Graphical Representation



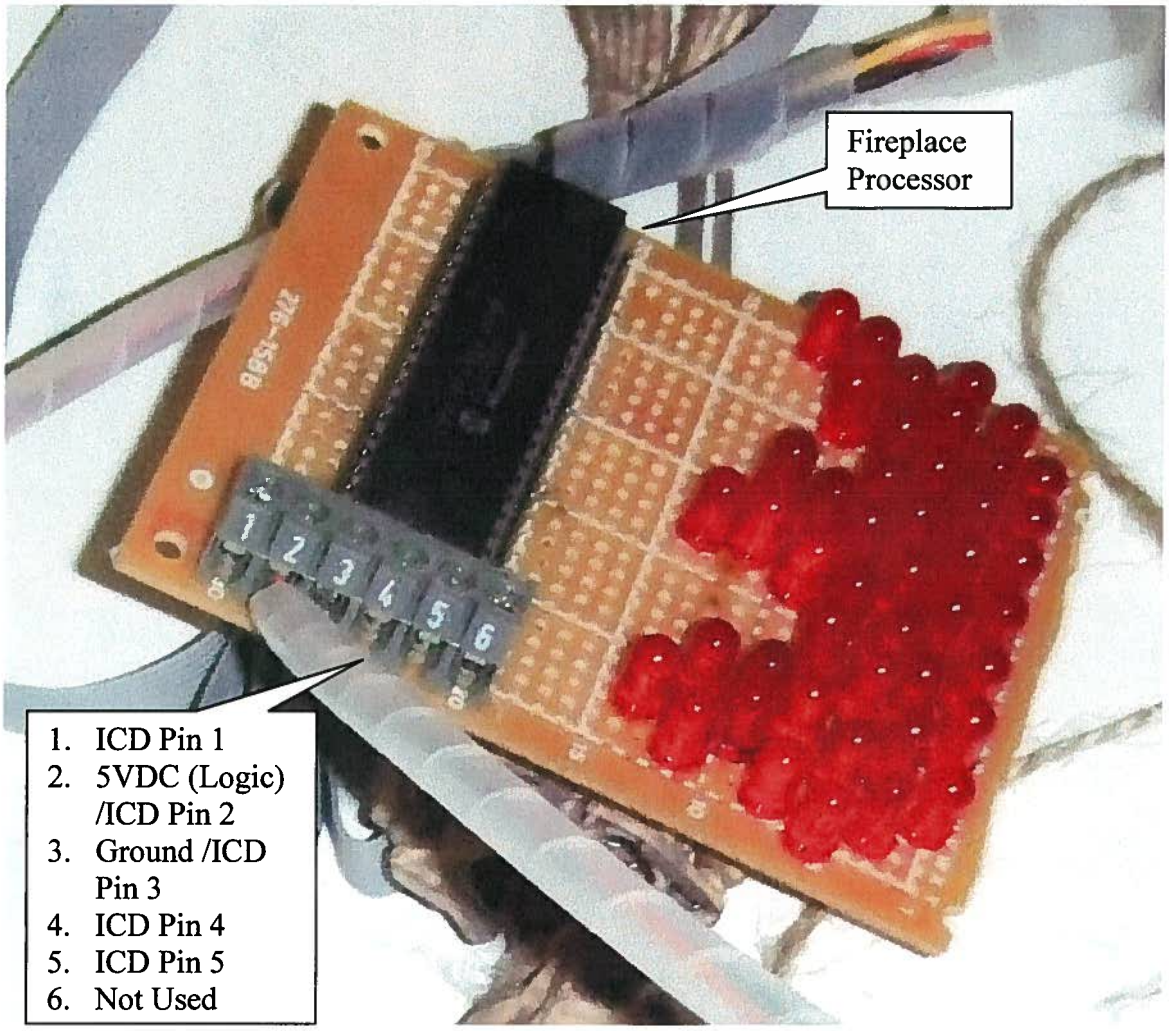
Completed main board.



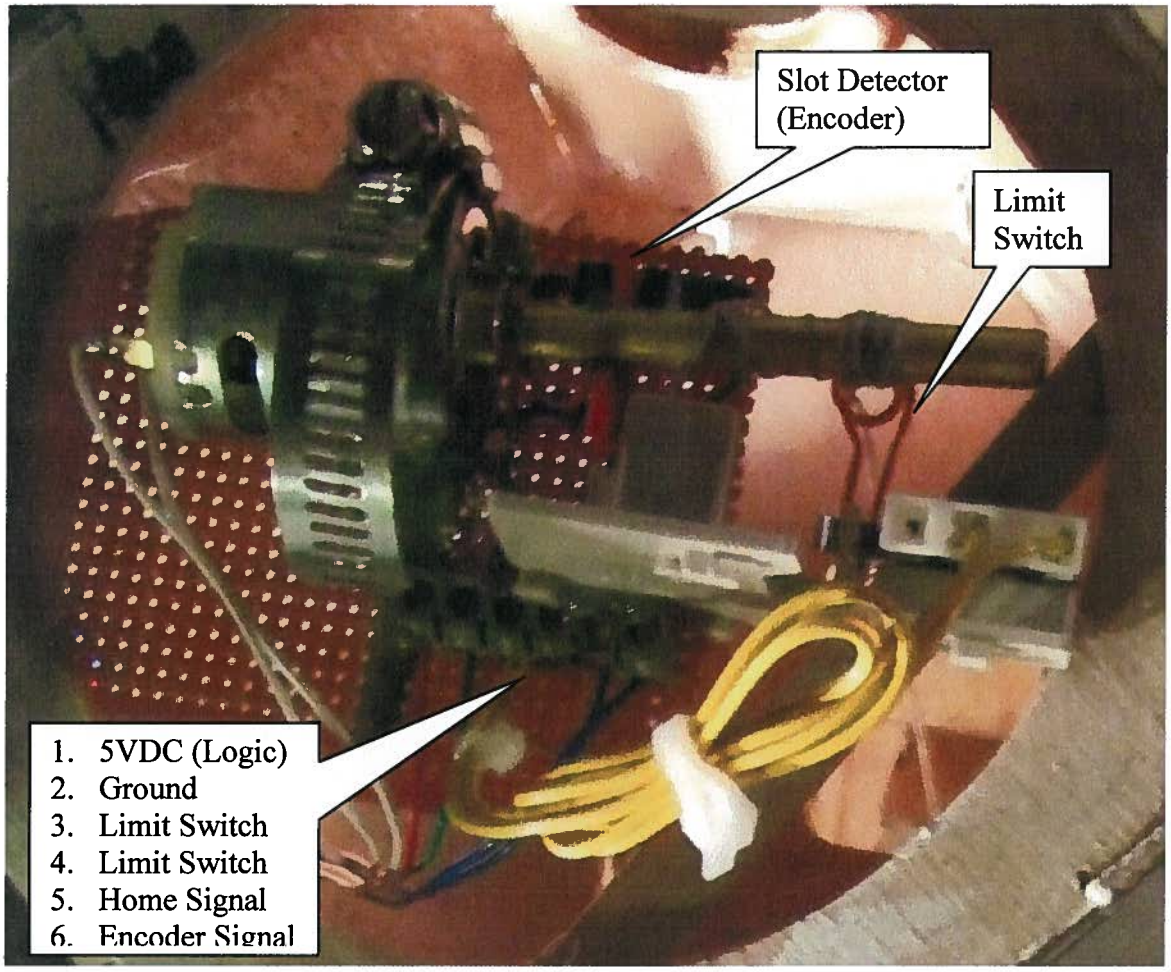
Remote Driver Board with Tower Motor Reversing Circuit



Basement Processor Board



Fireplace Processor and Board



Tower Motor Board

VIII. Microcontroller C Code

```
/*
Program for light control for the EE-490 Haunted House Project.
This processor simply receives a 3-wire serial signal and simply
converts the serial data into a parallel output for the lights.

Built for a PIC18F4550 processor.

Lance Bredthauer
Rev00   10/28/06
*/

#include <p18cxxx.h> /* for the special function register declarations */
#include <portb.h> /* for the RB0/INT0 interrupt */

/* Set configuration bits for use
* - set internal oscillator
* - disable watchdog timer
* - disable low voltage programming
* - enable background debugging
*/

#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = ON
#pragma config PBADEN = OFF //turns off A-D converter for PortB
#pragma config FOSC = INTOSCIO_EC //sets the internal oscillator (PIC18F4550)


#define Control_rise INTCON3bits.INT1IF //INPUT CONTROL PIN D2 The low-high Control pin runs a high
priority interrupt
#define Clock_fall INTCONbits.INT0IF    //INPUT CONTROL PIN D1 Data is stored on a high-low Clock pin
#define Data PORTBbits.RB2              //INPUT CONTROL PIN D3 Data is sent to this pin


//global variable declaration.  Command array collects the incomming data on the serial connection
//indexbit counts clock pulses for bit location
unsigned char Command[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
unsigned char indexbit=0;


/*
* For high interrupts, control is transferred to address 0x8.
*/
void bit_index (void); /* prototype needed for 'goto' below */
#pragma code HIGH_INTERRUPT_VECTOR = 0x8

void high_ISR (void)
{
_asm
goto bit_index
_endasm
}

#pragma code /* allow the linker to locate the remaining code */
#pragma interrupt bit_index

//This function runs on Control rise and Clock fall
void bit_index (void)
{   //beginning of bit_index function (interrupt)

    if(Clock_fall)  //stores bit value and increases bit index on a clock Fall
        {
        Command[indexbit]=Data;
        indexbit++;
        }

    if(Control_rise) indexbit=0;    //Starts over with bit zero

    Control_rise = 0;   /* clear flag to avoid another interrupt */
    Clock_fall = 0; //clear Interrupt flag
```

```
}   //end of bit_index function (interrupt)


void EnableHighInterrupts (void)
{   //initialization of Interrupt function
RCONbits.IPEN = 1; /* enable interrupt priority levels */
INTCONbits.GIEH = 1; /* enable all high priority interrupts */
INTCONbits.INT0IE = 1;  //enables INT0 pin interrupt for Clock
INTCON2bits.INTEDG0=0;  //falling edge for Clock
INTCON2bits.INTEDG1=1;  //rising edge for initiating READ
INTCON3bits.INT1IP=1;   //External Interrupt HIGH priority
INTCON3bits.INT2IE=0;   //Disable INT2 Interrupt
INTCON3bits.INT1IE=0;   //Enable INT1 Interrupt
}   //initialization of Interrupt function



void main (void)
{   //open main routine

EnableHighInterrupts ( );   //Open and configure Interrupt pins

OSCCON=0b11110010;  //sets maximum internal clock frequency
//Set all ports to output pins, except Communication on PORTB
TRISA = 0;
TRISB = 0b11000111; //ICD pins on 6,7; Serial on 0,1,2
TRISC = 0;
TRISD = 0;
TRISE = 0;

ADCON1 = 0b00001111;    //sets all PORTA bits to Digital


//loop applies received data array to output pins in packaging sequence
while(1)
    {
    PORTAbits.RA0 = Command[0];
    PORTAbits.RA1 = Command[1];
    PORTAbits.RA2 = Command[2];
    PORTAbits.RA3 = Command[3];
    PORTAbits.RA4 = Command[4];
    PORTAbits.RA5 = Command[5];
    PORTEbits.RE0 = Command[6];
    PORTEbits.RE1 = Command[7];
    PORTEbits.RE2 = Command[8];
    PORTAbits.RA6 = Command[9];
    PORTCbits.RC0 = Command[10];
    PORTCbits.RC1 = Command[11];
    PORTDbits.RD0 = Command[12];
    PORTDbits.RD1 = Command[13];
    PORTDbits.RD2 = Command[14];
    PORTDbits.RD3 = Command[15];
    PORTCbits.RC6 = Command[16];
    PORTCbits.RC7 = Command[17];
    PORTDbits.RD4 = Command[18];
    PORTDbits.RD5 = Command[19];
    ]


}   //end main routine
```

```
/*This is the program for the main timing processor in the EE-490
Haunted House Project.  A lighting processor is connected serially,
the basement processor, tower processor, and right-wing processor
 are connected in bi-directional parallel.  The
processor used is a PIC18F4550.
Lance Bredthauer
Rev00   10/26/06
Rev01   10/30/06
Rev02   12/05/06
*/

#include <p18cxxx.h>
#include <pwm.h>

#pragma config WDT = OFF    //disable watchdog timer
#pragma config LVP = OFF    //disable low voltage programming
#pragma config FOSC = INTOSCIO_EC   //Set Internal Oscilator (PIC18F4550)

#pragma config PBADEN = OFF //turns off A-D converter for PortB


#define serialdata PORTDbits.RD4        //Output Control Pin D3 - Data
#define serialclock PORTDbits.RD6       //Output Control Pin D2 - Clock
#define serialcon PORTDbits.RD5         //Output Control Pin D1 - Control
#define winch_encoder INTCONbits.INT0IF //Input Encoder from noose winch
#define winch_home PORTBbits.RB1        //Input Limit Switch - noose home
#define front_door_actuator PORTAbits.RA1   //Output Control Pin B1 - Front Door Actuator ON=Open
#define fireplace_actuator PORTAbits.RA2    //Output Control Pin B2 - Fireplace Actuator ON=Open
#define fireplace_LED PORTAbits.RA3         //Output Control Pin B3 - Fireplace Flame Control
#define chase_on PORTAbits.RA4              //Output Control Pin B4 - Chase Track ON=Move
#define servo_busy PORTBbits.RB4            //Input Control Pin B5 - Servo ON=Busy
#define chase_busy PORTBbits.RB5            //Input Control Pin B6 - Chasing Track ON=Busy
#define shingle_step PORTDbits.RD0          //Output Control Pin C1 - Step shingle arm half-way
#define shingle_full PORTDbits.RD1          //Output Control Pin C2 - Move shingle arm full-on
#define electric_chair PORTAbits.RA5        //Output Control Pin C3 - Electric chair on
#define mp3_next_button PORTEbits.RE2       //Output MP3 Control
#define mp3_play_button PORTEbits.RE1       //Output MP3 Control
#define mp3_power_pin PORTEbits.RE0         //Output MP3 Control



//Output - Tower Motor Forward = PWM1
//Output - Tower Motor Reverse = PWM2



#define OSCCON_fast 0b11110010  //For Fast internal operation
#define OSCCON_slow 0b10010010  //For PWM Tower Motor
#define lamp_pins 19    //For serial connection
#define electric_chair_skips 3
#define tower_period 250
#define slow 350
#define fast 250
#define short_delay 50
#define mp3_button_delay 1500
#define mp3_hold_delay 5000
#define encoder_move 18
#define mp3_off_delay 5000  //delay for MP3 player OFF

//definition of serial processor outputs
#define LED_porch_center 9
#define LED_porch_tower 5
#define LED_porch_chase 6
#define LED_front_door 3
#define LED_tower_down 2
#define LED_E_up 4
#define LED_E_down 7
#define LED_fireplace 8
#define LED_stair_top 1
#define LED_tower_up 10
```

```
#define fan_pin 11
#define shingle_on 12
#define shingle_direction 14


double mp3_power_on_delay = 15000;
double tower_delay = 10000;
double chase_delay_max = 20000;
double servo_delay_max = 20000;
double delay_1a = 18000;    //From "Play" to porch light
double delay_1b = 20000;    //From porch light to door open
double delay_1c = 20000;    //From door open to room light
double delay_1d = 20000;    //From room light to door close
double delay_1e = 20000;    //From door close to next track (hanging)
double delay_2a = 9000;     //From next track to noose drop
double delay_2b = 20000;    //From noose drop to Red Light
double delay_2c = 20000;    //From Red Light to Next Track (chase)
double delay_3a = 20000;    //From Next Track to Door Open
double delay_3b = 20000;    //From Door Open to Blue LED's
double delay_3c = 1;    //From Blue LED's to Chase Start
double delay_3d = 44000;    //From Chase Start to Door Close
double delay_3e = 10000;    //From Door Close to Next Track (Fireplace)
double delay_4a = 7000;     //From Next Track to Lights Off
double delay_4b = 1;        //From lights off to Fireplace turn
double delay_4c = 60000;    //From Fireplace turn to Fireplace ON
double delay_4d = 5000; //From Fireplace ON to next track (Electric Chair)
double delay_5a = 25000;    //From Next Track to Lights conditioning
double delay_5b = 25000;    //From lights Conditioning to Electric Chair On
double delay_5c = 38;       //Electric Chair ON time (number of strobes)
double delay_5d = 20000;    //From electric chair off to next track (shingles)
double delay_6a = 10000;    //From Next track to Shingles UP
double delay_6b = 25000;    //From Shingles up to Light Conditioning
double delay_6c = 25000;    //From Light conditioning to next track (Thunder, Fan)
double delay_7a = 20000;    //From Next track to Fan ON
double delay_7b = 20000;    //From Fan ON to strobe
double delay_7c = 60;       //Strobe iterations

double index;
int encoder_count = 19;
int tower_delay_min = 500;
int strobe_delay = 5000;
double tower_delay_max = 1000;
int speed, i;
int electric_chair_cycles = 0;
char Command[lamp_pins+1];


/*
* For high interrupts, control is transferred to address 0x8.
*/
void encoder (void); /* prototype needed for 'goto' below */
#pragma code HIGH_INTERRUPT_VECTOR = 0x8

void high_ISR (void)
{
_asm
goto encoder
_endasm
}

#pragma code /* allow the linker to locate the remaining code */
#pragma interrupt encoder

//This function counts encoder pulses
void encoder (void)
{   //beginning of encoder function (interrupt)
    encoder_count++;

    winch_encoder = 0;  /* clear flag to avoid another interrupt */
    INTCON3bits.INT1IF = 0; //clear Interrupt flag
}   //end of encoder function (interrupt)
```

```
void EnableHighInterrupts (void)
{   //initialization of Interrupt function
RCONbits.IPEN = 1; /* enable interrupt priority levels */
INTCONbits.GIEH = 1; /* enable all high priority interrupts */
INTCONbits.INT0IE = 1;  //enables INT0 pin interrupt for Clock
INTCON2bits.INTEDG0=0;  //falling edge for Clock
INTCON2bits.INTEDG1=1;  //rising edge for initiating READ
INTCON3bits.INT1IP=1;   //External Interrupt HIGH priority
INTCON3bits.INT2IE=0;   //Disable INT2 Interrupt
INTCON3bits.INT1IE=0;   //Enable INT1 Interrupt
}   //initialization of Interrupt function


//"bitsend" function outputs an individual serial bit
void bitsend (char value)
{   //bitsend function start
serialclock = 0;
value=value;
value=value;
value=value;
serialdata = value;
value=value;
value=value;
value=value;
value=value;
serialclock = 1;
value=value;
value=value;
value=value;
value=value;
value=value;
serialclock = 0;
}   //end of bitsend


//serialsend function that sends the entire array to be outputted to the serial processor
void serialsend (void)
{   //beginning of serialsend
int i;  //declare index variable
//sends initial values
serialcon=1;    //control pin resets bit count on slave
for(i=0;i<5;i++) {}
for(i=0;i<=lamp_pins;i++)
    bitsend(Command[i]);    //sends individual bit
serialcon=0;
for(i=0;i<5;i++) {}
}   //end of serialsend


//MP3 player POWER ON
void mp3_on (void)
{
mp3_next_button = 0;    //let go of other button
mp3_power_pin = 1;      //power on
for(index=0;index<mp3_button_delay;index++) {}
mp3_play_button = 1;    //"play" on
for(index=0;index<mp3_hold_delay;index++) {}
mp3_play_button = 0;    //let go of "play"
for(index=0;index<mp3_power_on_delay;index++) {}
}


//MP3 player PLAY
void mp3_play (void)
{
mp3_play_button = 1;
for(index=0;index<mp3_button_delay;index++) {}
```

```
mp3_play_button = 0;
}


//MP3 player NEXT
void mp3_next (void)
{
mp3_next_button = 1;
for(index=0;index<mp3_button_delay;index++) {}
mp3_next_button = 0;
}


//MP3 player OFF
void mp3_off (void)
{
mp3_power_pin = 0;
mp3_play_button = 0;
mp3_next_button = 0;
for(index=0;index<mp3_off_delay;index++) {}
}


//Tower Winch UP sequence
void winch_up (int speed)
{
OSCCON = OSCCON_slow;       //slow oscillator for lower motor frequency
OpenPWM2(tower_period);     //Apply "up" voltage to motor
SetDCPWM2(speed);           //for specified duty cycle
index = 0;
while(index<tower_delay_max & winch_home) //Delay for winch home or max delay
    {
    index++;
    }
ClosePWM2();            //stop motor
OSCCON = OSCCON_fast;   //resume high oscillator speed
}   //end of winch_up sequence


//Tower Winch Down sequence
void winch_down (int speed)
{
OSCCON = OSCCON_slow;       //slow oscillator for lower motor frequency
OpenPWM1(tower_period);     //Apply "down" voltage to motor
SetDCPWM1(speed);           //for specified duty cycle
index = 0;
encoder_count = 0;          //Reset encoder position
while(index<tower_delay_max & encoder_count<encoder_move} //Delay for set encoder turns or max delay
    {
    index++;
    }
ClosePWM1();            //stop motor
OSCCON = OSCCON_fast;   //resume high oscillator speed
}   //end of winch_up sequence


void lights_off (void)
{   //lights_off sequence turns off all lights on serial processor
for(i=0;i<lamp_pins;i++)
    {
    Command[i]=0;
    }
}   //end of lights_off sequence


void lights_on (void)
{   //lights_on sequence turns on all lights on serial processor
for(i=0;i<lamp_pins;i++)
    {
    Command[i]=1;
    }
```

```
    Command[shingle_on] = 0;
    Command[shingle_direction] = 0;
}   //end of lights_on sequence


//Homing Sequence
void home (void)
{
mp3_off();  //mp3 player off for reset
mp3_on();

//Homes the tower noose
if(winch_home)
    {
    winch_up(slow);
    }


fireplace_LED = 0;          //turns off fireplace
fireplace_LED = 0;          //turns off fireplace

fireplace_actuator = 0;     //Homes fireplace
front_door_actuator = 0;    //Homes front door
chase_on = 0;               //Homes Chase function

shingle_step = 0;           //closes shingle arm
shingle_full = 0;           //  ||
electric_chair = 0;         //turns off electric chair

lights_off();               //Turns off all lights on serial processor
serialsend();

index = 0;
while(servo_busy & index < servo_delay_max)
    {
    index++;
    }

index = 0;
while(chase_busy & index < chase_delay_max)
    {
    index++;
    }

mp3_on();

}   //end of home sequence


void main (void)
{   //beginning of main function

int i=0;    //index variable
char up = 1;
char data_length=20;
char clock_pulse;
char clock_state;
int delay = 1000;   //LED delay
int index1;
char light=0;


OSCCON=OSCCON_fast; //sets internal oscilator to maximum internal speed

//set direction of all pins to output
TRISA = 0b00000000;     //sets Control, Data, Clock pins to input
ADCON1 = 0b00001111;    //sets all PORTA bits to Digital
TRISB = 0xff;           //All PORTB is inputs
TRISC = 0;              //All PORTC is outputs
TRISD = 0;
```

```
TRISE = 0;

electric_chair = 0;

ClosePWM1();
ClosePWM2();

//sets intial values for output pins for serial commanded processor
lights_off();
serialsend();

EnableHighInterrupts(); //Configures interrupts for encoder counts

home();  //initial homing sequence


while(1)    //outer while loop
    {   //initialization of outer while loop

//Front door open & close Sequence
    mp3_play();

    for(i=0;i<delay_1a;i++) {}

    //light
    Command[LED_porch_center] = 1;  //porch light on
    serialsend();

    for(i=0;i<delay_1b;i++) {}

    //door open
    front_door_actuator = 1;
    for(i=0;i<short_delay;i++) {}
    index = 0;
    while(servo_busy & index < servo_delay_max)
        {
        index++;
        }

    for(i=0;i<delay_1c;i++) {}

    //light
    Command[LED_front_door] = 1;    //
    serialsend();

    for(index=0;index<delay_1d;index++) {}

    //door close
    front_door_actuator = 0;
    for(i=0;i<short_delay;i++) {}
    index = 0;
    while(servo_busy & index < servo_delay_max)
        {
        index++;
        }

    Command[LED_porch_center] = 0;
    serialsend();

    for(i=0;i<delay_1e;i++) {}



//Noose falls sequence
    mp3_next();

    for(i=0;i<delay_2a;i++) {}

    //noose drops
        //lights
    Command[LED_tower_down] = 1;
```

```
        Command[LED_front_door] = 0;
        serialsend();

            //action
        winch_down(fast);

        for(index=0;index<delay_2b;index++) {}

        //light
        Command[LED_front_door] = 1;
        Command[LED_tower_down] = 0;
        serialsend();

        for(index=0;index<delay_2c;index++) {}

    //Chase squence
        mp3_next();

        for(i=0;i<delay_3a;i++) {}

        Command[LED_porch_center] = 1;
        Command[LED_porch_chase] = 1;
        Command[LED_front_door] = 0;
        serialsend();

        //door opens
        front_door_actuator = 1;
        for(i=0;i<short_delay;i++) {}
        index = 0;
        while(servo_busy & index < servo_delay_max)
            {
            index++;
            }

        for(i=0;i<delay_3c;i++) {}

        //chase run
        chase_on = 1;
        for(i=0;i<short_delay;i++) {}
        chase_on = 0;
        index = 0;
        while(chase_busy & index < chase_delay_max)
            {
            index++;
            }

        for(index=0;index<delay_3d;index++) {}

        //door closes
        front_door_actuator = 0;
        for(i=0;i<short_delay;i++) {}
        index = 0;
        while(servo_busy & index < servo_delay_max)
            {
            index++;
            }

        Command[LED_porch_center] = 0;
        Command[LED_porch_chase] = 0;
        Command[LED_fireplace] = 1;
        serialsend();

        for(index=0;index<delay_3e;index++) {}


    //Fireplace open & flicker Sequence
        mp3_next();

        for(i=0;i<delay_4a;i++) {}

        Command[LED_fireplace] = 0;
```

```
        serialsend();

        for(i=0;i<delay_4b;i++) {}

        fireplace_LED = 1;


        for(index=0;index<delay_4c;index++) {}

        //fireplace rotates
        fireplace_actuator = 1;
        for(i=0;i<short_delay;i++) {}
        index = 0;
        while(servo_busy & index < servo_delay_max)
            {
            index++;
            }

        for(index=0;index<delay_4d;index++) {}

        Command[LED_fireplace] = 1;
        serialsend();

        fireplace_LED = 0;

    //Electric chair Sequence
        mp3_next();

        for(i=0;i<delay_5a;i++) {}

        Command[LED_fireplace] = 0;
        Command[LED_E_down] = 1;
        serialsend();

        for(i=0;i<delay_5a;i++) {}

        Command[LED_E_up] = 1;
        Command[LED_E_down] = 0;
        serialsend();

        for(index=0;index<delay_5b;index++) {}

        //run electric chair
        electric_chair_cycles++;
        if(electric_chair_cycles > electric_chair_skips)
            {
            electric_chair = 1;
            }

        for(index=0;index<delay_5c;index++)
            {
            Command[LED_E_up] = 1;
            serialsend();
            for(i=0;i<strobe_delay;i++) {}
            Command[LED_E_up] = 0;
            serialsend();
            for(i=0;i<strobe_delay;i++) {}
            }
        electric_chair = 0;

        Command[LED_E_up] = 0;
        serialsend();

        for(index=0;index<delay_5d;index++) {}


    //Shingle trick Sequence
        mp3_next();

        for(index=0;index<delay_6a;index++) {}
```

```
        //face open & illuminate
        Command[shingle_on] = 1;
        Command[shingle_direction] = 0;
        serialsend();

        for(index=0;index<delay_6b;index++) {}

        Command[shingle_on] = 1;
        Command[shingle_direction] = 1;
        serialsend();

        for(index=0;index<delay_6c;index++) {}

        Command[shingle_on] = 0;
        Command[shingle_direction] = 0;
        serialsend();

    //Fan, thunder, reset Sequence
        mp3_next();

        Command[LED_stair_top] = 1;
        serialsend();

        for(index=0;index<delay_7a;index++) {}

        //Fan ON
        Command[fan_pin] = 1;
        serialsend();

        for(index=0;index<delay_7b;index++) {}

        for(index=0;index<delay_7c;index++)
            {
            lights_on();
            serialsend();
            for(i=0;i<strobe_delay;i++) {}
            lights_off();
            Command[fan_pin] = 1;
            serialsend();
            for(i=0;i<strobe_delay;i++) {}
            }

        Command[LED_stair_top] = 0;
        serialsend();

        //home all actions while fan runs (no lights)
        home();
        }   //end of outer while loop
    }   //end of main function
```

```
/*This is the program for the fireplace in the EE-490
Haunted House Project.  Power is applied for main effect;
special effect is controlled by one input pin.  The
processor used is a PIC18F4550.
Lance Bredthauer
Rev00   10/24/06
*/

#include <p18cxxx.h>

#pragma config WDT = OFF    //disable watchdog timer
#pragma config LVP = OFF    //disable low voltage programming

#define LEDA0 PORTAbits.RA0
#define LEDA1 PORTAbits.RA1
#define LEDA2 PORTAbits.RA2
#define LEDA3 PORTAbits.RA3
#define LEDA4 PORTAbits.RA4
#define LEDA5 PORTAbits.RA5
#define LEDA6 PORTAbits.RA6
#define LEDB0 PORTBbits.RB0
#define LEDB1 PORTBbits.RB1
#define LEDB2 PORTBbits.RB2
#define LEDB3 PORTBbits.RB3
#define LEDC0 PORTCbits.RC0
#define LEDC1 PORTCbits.RC1
#define LEDD0 PORTDbits.RD0
#define LEDD1 PORTDbits.RD1
#define LEDD2 PORTDbits.RD2
#define LEDD3 PORTDbits.RD3
#define LEDD4 PORTDbits.RD4
#define LEDD5 PORTDbits.RD5
#define LEDD6 PORTDbits.RD6
#define LEDD7 PORTDbits.RD7
#define LEDE0 PORTEbits.RE0
#define LEDE1 PORTEbits.RE1


void row1 (char state)
{   //beginning of row1 function
LEDA0 = state;
LEDB0 = state;
LEDD2 = state;
LEDB1 = state;
}   //end of row1 function


void row2 (char state)
{   //beginning of row2 function
LEDA1 = state;
LEDB2 = state;
LEDB3 = state;
LEDD4 = state;
LEDD5 = state;
}   //end of row2 function


void row3 (char state)
{   //beginning of row3 function
LEDA2 = state;
LEDD7 = state;
LEDD1 = state;
LEDD3 = state;
LEDD0 = state;
}   //end of row3 function


void row4 (char state)
{   //beginning of row4 function
LEDA3 = state;
LEDA5 = state;
```

```
}   //end of row3 function

//turns on left toung
void toung1 (char state)
{   //beginning of toung1 function
LEDA4 = state;
LEDA6 = state;
}   //end of toung1 function

//turns on lower-center toung
void toung2 (char state)
{   //beginning of toung2 function
LEDE1 = state;
}   //end of toung2 function

//turns on upper-center toung
void toung3 (char state)
{   //beginning of toung3 function
LEDC1 = state;
}   //end of toung3 function

//turns on right toung
void toung4 (char state)
{   //beginning of toung4 function
LEDD6 = state;
LEDE0 = state;
}   //end of toung4 function

//turns on flame tips
void toung5 (char state)
{   //beginning of toung5 function
LEDC0 = state;
}   //end of toung5 function

//turns off all output pins
void clear (void)
{   //beginning of clear function
PORTA = 0;
PORTB = 0;
PORTC = 0;
PORTD = 0;
PORTE = 0;
}   //end of clear function

//turns on all output pins
void set (void)
{   //beginning of set function
PORTA = 0xff;
PORTB = 0xff;
PORTC = 0xff;
PORTD = 0xff;
PORTE = 0xff;
}   //end of set function

void main (void)
{   //beginning of main function

int i=0;    //index variable
char up = 1;
int delay = 80; //LED delay

//set direction of all pins to output
TRISA = 0;
TRISB = 0;
TRISC = 0;
TRISD = 0;
TRISE = 0;

//Initialize all ports OFF
clear();
```

```
while(1)    //outer while loop
    {   //initialization of outer while loop
    if(up==1) delay = delay*(105/100);
    else delay = delay*(95/100);
    if(delay>=300) up=0;
    if(delay<=40) up=1;
    for (i=0;i<delay;i++) {}
    row1(1);
    for (i=0;i<delay*3;i++) {}
    row2(1);
    for (i=0;i<delay*2;i++) {}
    row3(1);
    for (i=0;i<delay*2;i++) {}
    row4(1);
    for (i=0;i<delay;i++) {}
    toung1(1);
    toung2(1);
    toung3(1);
    toung4(1);
    toung5(1);
    for (i=0;i<delay;i++) {}
    set();
    for (i=0;i<delay/5;i++) {}
    clear();
    row1(1);
    for (i=0;i<delay;i++) {}

    }   //end of outer while loop
}   //end of main function
```

```
/*This is the program for the basement processor in the EE490
Capstone Sr Design project Haunted House.  There is a 6-pin bi-directional
parallel signal to the main processor - 4 inputs and 2 outputs.  The actions
controlled are two servo motors and a discrete DC brush-type motor with
a homing limit switch.

This is for a PIC18F2520 processor.

Lance Bredthauer
Rev00   10/26/06
Rev01   12/01/06
*/

#include <p18cxxx.h>
#include <adc.h>
#include <pwm.h>

#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config PBADEN = OFF //set PORTB to Digital I/O
#pragma config OSC = INTIO67    //set oscilator internal w/RA6


#define door_servo PORTBbits.RB1            //INPUT Control B1 pin
#define fireplace_servo PORTBbits.RB2       //INPUT Control B2 pin
#define fireplace_flame PORTBbits.RB3       //INPUT Control B3 pin
#define chase PORTBbits.RB4                 //INPUT Control B4 pin
#define chase_busy PORTAbits.RA6            //OUTPUT CONTROL PIN B6
#define servo_busy PORTAbits.RA7            //OUTPUT CONTROL PIN B5
#define fireplace_servo_power PORTCbits.RC5 //OUTPUT Power for servo movement (RC5)
#define door_servo_power PORTCbits.RC3      //OUTPUT Power for servo movement (RC3)
#define fireplace_LED_output PORTAbits.RA3  //OUTPUT (not used)
#define chase_motor PORTAbits.RA4           //OUTPUT Chase operating
#define chase_home PORTBbits.RB0            //INPUT Chase object home

#define OSCCON_fast 0b11110010
#define OSCCON_slow 0b10010010
#define OSCTUNE_fast 0b11001111
#define OSCTUNE_center 0b11000000
#define OSCTUNE_slow 0b11010000


int servo_period = 200;
int fireplace_close_duty = 5;
int door_close_duty = 600;
int fireplace_open_duty = 795;
int door_open_duty = 200;

double index;
int fireplace_open_delay = 140;
int door_open_delay = 100;
int servo_power_on_delay = 3;
char fireplace_position;
char door_position;
int chase_delay_min = 2000;
double chase_delay_max = 100000;


void Chase_sequence (void)
{
chase_busy = 1;
chase_motor = 1;
for(index=0;index<chase_delay_min;index++) {}
index=0;
while(index<chase_delay_max & chase_home)
    {
    index++;
    }
chase_busy = 0;
chase_motor = 0;
}
```

```
void Open_fireplace (void)
{
OSCCON = OSCCON_slow;
servo_busy = 1;
fireplace_servo_power = 1;
for(index=0;index<servo_power_on_delay;index++){}
OpenPWM1(servo_period);
SetDCPWM1(fireplace_open_duty);
for(index=0;index<fireplace_open_delay;index++){}
//fireplace_servo_power = 0;
ClosePWM1();
OSCCON = OSCCON_fast;
servo_busy = 0;
fireplace_position=1;
}


void Close_fireplace (void)
{
OSCCON = OSCCON_slow;
servo_busy = 1;
fireplace_servo_power = 1;
for(index=0;index<servo_power_on_delay;index++){}
OpenPWM1(servo_period);
SetDCPWM1(fireplace_close_duty);
for(index=0;index<fireplace_open_delay;index++){}
//fireplace_servo_power = 0;
ClosePWM1();
OSCCON = OSCCON_fast;
servo_busy = 0;
fireplace_position=0;
}


void Open_door (void)
{
OSCCON = OSCCON_slow;
servo_busy = 1;
door_servo_power = 1;
for(index=0;index<servo_power_on_delay;index++){}
OpenPWM2(servo_period);
SetDCPWM2(door_open_duty);
for(index=0;index<door_open_delay;index++){}
//door_servo_power = 0;
ClosePWM2();
OSCCON = OSCCON_fast;
servo_busy = 0;
door_position=1;
}


void Close_door (void)
{
OSCCON = OSCCON_slow;
servo_busy = 1;
door_servo_power = 1;
for(index=0;index<servo_power_on_delay;index++){}
OpenPWM2(servo_period);
SetDCPWM2(door_close_duty);
for(index=0;index<door_open_delay;index++){}
//door_servo_power = 0;
ClosePWM2();
OSCCON = OSCCON_fast;
servo_busy = 0;
door_position=0;
}


void home (void)
{
```

```
OSCCON = OSCCON_slow;
servo_busy = 1;
fireplace_LED_output = 0;
fireplace_servo_power = 1;
for(index=0;index<servo_power_on_delay;index++){}
OpenPWM1(servo_period);
SetDCPWM1(fireplace_close_duty);

door_servo_power = 1;
for(index=0;index<servo_power_on_delay;index++){}
OpenPWM2(servo_period);
SetDCPWM2(door_close_duty);

for(index=0;index<fireplace_open_delay;index++){}
//fireplace_servo_power = 0;
ClosePWM1();

fireplace_position=0;

//door_servo_power = 0;
ClosePWM2();
OSCCON = OSCCON_fast;
door_position=0;
servo_busy = 0;
chase_motor = 0;
}

void main (void)
{

double delay=20000;
int i;

OSCTUNE = OSCTUNE_center;
OSCCON = OSCCON_fast;
TRISA=0;
TRISB=0xff;
PORTA=0;
TRISC=0;
PORTB=0;
ADCON1=0b00001111;

fireplace_servo_power = 1;
for(index=0;index<servo_power_on_delay*1000;index++){}

door_servo_power = 1;
for(index=0;index<servo_power_on_delay*1000;index++){}


home();

while(1)
{

    fireplace_LED_output = fireplace_flame;

    if(chase)
        {
        Chase_sequence();
        }

    if(!fireplace_position & fireplace_servo)
        {
        Open_fireplace();
        }

    if(fireplace_position & !fireplace_servo)
        {
        Close_fireplace();
        }
```

```
        if(!door_position & door_servo)
            {
            Open_door();
            }

        if(door_position & !door_servo)
            {
            Close_door();
            }


    }
    }
```