T.P.S. REPORT

COVER SHEET

Prepared By: Carly Geehr, Peter Rubin and Ted Sweet

Date: 2007-12-07

Device/Program Type: Stapler Game for ME 218A

Vendor: Stanford University

Project Start Date: 11-05-2007

Project Completion Date: 12-07-2007

CONFIDENTIAL
PURPOSE OF REPORT

This report provides detailed information on the mechanical, electronic and software related elements of the Stapler Game. The information should be enough to allow someone with knowledge of mechatronics to be able to replicate the game. However, this game is quite complicated so beware that replicating it will be a daunting task.

The pages that follow are primarily concerned with the technical information of the game. For a description of the game and how one plays it, please see the website.
CHASIS

Every good penny arcade game starts with a solid, well built chassis as a foundation. The Stapler Game features such a chassis. This chassis is not only strong but is also aesthetically interesting. The chassis is intended to look like an office cubical with exaggerated perspective. When one approaches the Stapler Game, they feel as though they have just sat down at a stereotypical desk in a stereotypical cubical.

The chassis was constructed primarily our of masonite fiberboard and plywood, which was cut both using a laser cutter and hand tools. The technical drawings below show the rough size of the chassis and the layout of the various components on the chassis.
STAPLERS

The primary user input in the game is the staplers. The staplers are unmodified Swingline staplers. A limit switch is mounted on the underside of each stapler to sense when the staplers are pressed. There are also tape sensors mounted behind the staplers which can sense when the stapler has been opened. A single LED is mounted below each stapler to alert the player they should be stapling during play. All of these components are mounted to a laser cut stack of acrylic sheets. These sheets have mounts for all of the various components and the staplers. They are also transparent, so the acrylic appears to glow when the LED is turned on. The drawing below shows how all of the components fit together.
PAPERSTACKS

The “stacks of paper” at the rear of the desk are the main indication to the player of how much progress they have made in the game. Although it may appear that there are actual stacks of paper, there are not. As shown in the drawings below, the “stacks of paper” are just cardstock facades mounted on to a rack and pinion mechanism that makes them rise and fall.
There is a linear cam mechanism which actuates a limit switch when the stack is at either its top or bottom limit.

Linear Cam Functionality

Rack and Pinion and Linear Cam
BUSINESS CARD DISPENSER

At the end of the game, the business card dispenses business cards to the players as a prize. The dispenser uses two DC motors to deliver the cards. One of the motors powers a foam wheel which feeds the cards from a caddy to the next stage of the dispenser. The other motor powers an acrylic wheel which functions as a flywheel to launch the cards up into the air.
POPUPS (BOSS AND ERGONOMIST)

During game play, the players are visited by both their boss and their ergonomist. These "visits" are facilitated by a servo motor which rotates an arm that has an image of a boss on one side and an image of an ergonomist on the other. The servo only has a rotational range of 90 degrees, yet 180 degrees are needed to account for the three possible displays: the boss, the ergonomist or nothing. There are laser cut gears to gear up the servo so that there is a full 180 degrees of rotation.
OVERVIEW

The Stapler Game uses all of the I/O pins on the C32 microcontroller. The diagram below shows what each pin on the C32 is used for. As the diagram shows, the pins feed into a variety of different circuits which have a variety of functions.
STAPLERS

Each stapler is equipped with a limit switch (for sensing when the stapler has been pressed) and a tape sensor (for sensing when the stapler has been opened). The tape sensor is a packaged device that contains an infrared LED and a phototransistor. On the detection side, the circuit features a comparator with hysteresis implemented to get a clean digital signal. The hysteresis was adjusted so that the output to the C32 changed states when the stapler was within 1/8 inch of the tape sensor.

The limit switch circuit has a pull down resistor so that the output to the C32 is pulled down to 0 V when the stapler is no depressed. However, when the stapler is depressed the limit switch is closed and the output changes state.
The paperstacks are raised and lowered using a geared DC motor. The motor is controlled using a motor driver chip (LM293B) in an H-Bridge configuration with protection diodes. One output from the C32 is used as a PWM signal to the driver and another output is used to control the direction of the motor. A hex inverter is used to invert the direction signal so that a single output from the C32 can be used to control the two inputs to the motor driver.

A limit switch is used to detect when a stack has reached either the top or bottom of its travel. A debouncing circuit (consisting of low-pass filter and hex inverter) is used to eliminate noise when the switch is actuated.
PRIZE MOTORS

The motors used to administer the business cards are controlled using a motor driver (LM293B) and a half H-bridge circuit with diode protection. A half H-bridge was used because the motors only spin one direction.

BOSS/ERGO SERVO

The servo which raises and lowers the boss and ergonomist is controlled by a very simple circuit. The servo is connected to 5 V and ground and is given a PWM signal directly from the C32 which sets the position of the servo.
LED’s

The game contains many LED’s to alert the players of what is going on in the game. Each of the LED circuits are controlled by an n-channel mosfet (2N7000) with a signal from the C32 connected directly to the gate of the mosfet. Each circuit has a different number of LED’s so each circuit has a different size resistor to limit the current through the LED’s.

LEVEL SELECT

A potentiometer is used to allow the players to select the difficulty of the game. The signal from the potentiometer is connected to an analog-to-digital input on the C32, so the position of the potentiometer can be sensed.
SOFTWARE

Below is the pseudo code for the software that runs the game. The actual code is written in C and is attached as Appendix A.

**Play.c – Main Event Checker Module**

Main() {
    Perform main initialization sequence
    Initialize reset once
    Run main event checker in an endless loop
}

Main Event Checker (runs continuously throughout the game, handles all game states) {
    Check for interrupts and refresh continuous state variables
    Check which game stage we’re in:
    { 
        If we’re in RESET mode:
            If mechanical elements are in place and no longer moving, you're done resetting. Wait for coin!
                Initialize waitcoin
                Turn off all LEDs
                Go to waitcoin stage
        If we’re in WAITCOIN mode:
            When a coin is sensed, initialize waitstart
            Go to waitstart stage
        If we’re in WAITSTART mode:
            check the level
            When the start button is pressed, initialize pregame
            Go to the pregame stage
        If we’re in PREGAME mode:
            If all mechanical elements are done moving
                Initialize the game, go to game stage
        If we’re in GAME mode:
            Check for winner. If there is one, end the game.
                Check which level we’re on:
                if EASY:
                    If player 1 has stapled
                        Increment score
                        Update paper stack 1 position
                    If player 2 has stapled
                        Increment score
                        Update position of paper stack
                if MEDIUM:
                    If player 1 has staples left
                        if player 1 has stapled
                            Increment score
                            Update position of paper stack
                    If they are out of staples, do nothing until they reload
                        Illuminate the reload light only
                        if player reloads
                            Illuminate the stapler lights only
    }
}

If player 2 has staples left
    if player 2 has stapled
        Increment score
        Update position of paper stack
    If they are out of staples, do nothing until they reload
        Illuminate the reload light only
        if player reloads
            Illuminate the stapler lights only
If HARD:
    One of three scenarios:

1) If it's past time for the ergonomist to show up and she hasn't already come
    Set ergo flag high
    Start timer so we know when to put her away
    Pop her up!
    Set “gotcha” flags to zero for both players

2) If ergo has already wreaked her havoc OR if she hasn’t shown up yet, we treat it like it’s MEDIUM
    If player 1 has staples left
        if player 1 has stapled
            Increment score
            Update position of paper stack
        If they are out of staples, do nothing until they reload
            Illuminate the reload light only
            if player reloads
                Illuminate the stapler lights only
    If player 2 has staples left
        if player 2 has stapled
            Increment score
            Update position of paper stack
        If they are out of staples, do nothing until they reload
            Illuminate the reload light only
            if player reloads
                Illuminate the stapler lights only

3) If ergo is present and is in the process of wreaking havoc
    If the grace period has expired, we can punish the players
        If player 1 has stapled
            If gotcha flag is low, they haven’t been punished yet, so punish them! ha!
            Decrement score
            Move paper stack up
            Blink their stapler!
            Gotcha flag goes high
        If player 2 has stapled
            If gotcha flag is low, they haven’t been punished yet, so punish them! ha!
            Decrement score
            Move paper stack up
            Blink their stapler!
            Gotcha flag goes high
        If ergo time has expired, is done wreaking havoc
            Put the management away
            Relight both staplers

if we’re in POSTGAME mode:
    dispense the prize
    When the prize has been successfully dispensed, reset the game!
    Go to reset stage
RunChecks (handles checking of all continuously monitored states) {
    If reset button has been tripped
        Reset the game
        Game stage is now RESET
        Blink any LEDs that need blinking
        Stop any paper stacks that have hit their limit switches
}

Initialize main game (only run once at beginning of game) {
    Initialize timer
    Initialize PWM, set period to 20ms
    Initialize AD ports
    Set all M ports to inputs
    Set all T ports to outputs
    Start with all bits off
}

Initialize reset game stage {
    Set paper stacks to bottom position
    Set coin inserted flag to zero
    Turn off all LEDs
    Put boss + ergo away
}

Initialize waitcoin game stage {
    Illuminate paycheck slot only
}

Check for coin {
    Get current state of coin sensor
    Return a 1 if coin has been detected
}

Initialize waitstart stage {
    Illuminate start light only
}

Initialize pregame stage {
    Set both paper stacks to top position
    Set scores to zero
    Refill staplers
    Turn all lights off
}

Initialize game stage {
    Illuminate staplers only
    If difficulty is hard
        set ergo surprise timer
        set ergo presence flag low
}

Initialize postgame sequence {
    Set player with highest score to “winner”
    Flash their stapler
    Boss emerges
    Initialize prize dispenser
}

IsWinner (returns true if a player has won) {
    Determine if any player’s score exceeds that of the winning score
    for the level they’re playing
    Return true if a player has won
**Paper.c – Module to control paper stack movement + positioning**

SetPopup (given a paper stack ID and position, moves the stack to that position) {
    If it’s already moving, do nothing
    If it’s going to the minimum position:
        If it’s already there, don’t do anything
        Otherwise, plow down until you hit the limit switch
        Update the position array for that paper stack
    If it’s going to the maximum position:
        If it’s already there, don’t do anything
        Otherwise, plow down until you hit the limit switch
        Update the position array for that paper stack
    Otherwise, if it’s going somewhere in between the max and min
        If the final position is higher than where we are right now
            Set motor direction to UP
            Calculate how long to turn on the motor to reach destination
            If the final position is lower than where we are right now
                Set motor direction to down
                Calculate how long to turn on the motor to reach destination
                Turn on the motor for the calculated period of time
                Update paper position array
{

Check all popups for arrival (returns true if all paper stacks are done moving) {
    If stack one and stack two are not moving, return true
}

Check popup for arrival (given a motor ID, tells us whether it’s moving) {
    If paper stack is moving
        If it has hit the limit switch
            If direction is DOWN
                Update paper position array to MIN
            If direction is UP
                Update paper position array to MAX
            Stop the motor
        If the motor timer has expired
            Stop the motor
    Return true
}

Calculate move duration (given incremental change in position, motor ID, and direction, returns length of time motor should be turned on) {
    Calculate percentage of total range of motion motor needs to cover
    If direction is UP
        Return percentage * total number of ticks to get from bottom to top
    If direction is DOWN
        Return percentage * total number of ticks to get from top to bottom
}

Start motor {
    If direction is UP
        Set direction bit HI
        Turn motor on
    If direction is DOWN
        Set direction bit LO
Turn motor on
If direction is 0
Stop the motor
Update paper moving array

At limit (returns state of limit switch) {
If paper stack limit switch is HI, it’s in the middle of travel,
return FALSE
Else return TRUE
}

Just Hit Limit (returns true if there has been a state change in the limit switch since the last check) {
If the last limit hit isn’t the same as the current limit state we’re in, we know there’s been a change in state
Set the current limit state to the one we’re in (zero if we’re not at a limit)
If we’re at a limit now, return true
**Interface.c – UI functions**

Get Level (returns the level as set by the pot) {
    Read the pin the pot is connected to
    If it’s between the limits set for EASY, return EASY
    If it’s between the limits set for MEDIUM, return MEDIUM
    Else return HARD
}

Initialize prize dispensing {
    Start timer for flywheel warmup
    Turn on flywheel
}

Check for prize dispensing (returns true if dispensing is complete) {
    If flywheel isn’t ready yet
        If flywheel timer has expired
            Set dispenser timer
            Set flywheel to ready
            Turn on dispenser motor
            Return false
    If the prize dispensed state is low
        Check for expired prize dispenser timer
            If timer’s up
                Prize dispensed state goes high
                Turn off both flywheel and dispenser motor
                Return false
    If the timers have both expired
        Set both flags low
        Return true
}

Reset button tripped (returns true if start button has been held down for set length of time to reset game) {
    If the button is down and it wasn’t last time we checked
        Set timer for length of time we need for a reset
        Set button down flag high
        Return false
    If the bit is no longer high, but our flag hasn’t been updated
        update the flag, Return false
    If the button isn’t down
        Set flag low, Return false
    If the button is down and we’ve been holding it there
        Check if the timer has expired
        If it has, return true
}
LED.c – Module for controlling all LEDs

Control LED (this turns a given LED on, off, or to blinking) {
    If state is ON
        Set blinking flag to 0
        Turn that LED on
    If state is OFF
        Set blinking flag to 0
        Turn LED off
    Else, we must be blinking
        Set blinking flag high
        Blink that LED
}

Blink LED (uses an array of states to control blinking) {
    Check if blink timer needs initializing
    If it does, initialize it
    If the timer has expired, we need to change the state of the LEDs
        Cycle through each LED
            If its state is set to blink
                If it was on before, turn it off
                Otherwise, turn it on
        Reset the timer
}

Turn off LED (takes a light ID, sets that bit LO)

Turn on LED (takes a light ID, sets that bit HI)

Turn all off {
    Cycles through each light ID
    Turns off that LED
}
Servo.c – Module to control servo

Set Servo Angle (sets the angle of the servo to the specified angle) {
    Convert the angle to a duty cycle quantity
    If the angle is set to the ergo angle, set the ergo flag high
    Otherwise, set that flag low
    Set the duty cycle of the servo to the calculated value
}

Convert angle to duty (returns the duty cycle scaled between the min and max values according to the specified angle)

Management away (returns the servo to the neutral angle)
Stapler.c – Module for stapler I/O

Check staple press (given a player number, returns if they’ve stapled or not)

For player 1:
  If the stapler switch is down and this is a new state
    Update state
  If we’re above level EASY, decrement staples in stapler
    Return 1
  If the stapler is not down and the current state is high
    The stapler has been released! Update state variable

For player 2:
  If the stapler switch is down and this is a new state
    Update state
  If we’re above level EASY, decrement staples in stapler
    Return 1
  If the stapler is not down and the current state is high
    The stapler has been released! Update state variable

Has reloaded (returns true if given player has reloaded stapler) {
  For player 1:
    If the player needs to reload
      Check for stapler opening, set state variable
      Check for stapler closing, set state variable
      If both conditions satisfied
        Refill stapler
        Reset all flags
        Return 1
  For player 2:
    If the player needs to reload
      Check for stapler opening, set state variable
      Check for stapler closing, set state variable
      If both conditions satisfied
        Refill stapler
        Reset all flags
        Return 1

Refill Staplers {
  Set player 1 staples left to max
  Set player 2 staples left to max
  Reset reload flags

Check Stapler (returns true if the stapler has been opened or closed, helper for reload function) {
  If player 1:
    If OPEN:
      If the stapler is open and this is a new state
        Set open flag to high
        Return 1
    If CLOSED
      If the stapler is closed and this is a new state
        Set closed flag to high
        Return 1
  If player 2:
    If OPEN:
      If the stapler is open and this is a new state
        Set open flag to high
        Return 1
    If CLOSED
If the stapler is closed and this is a new state
   Set closed flag to high
   Return 1

}Used a staple (handles player decrementing staples count) {
   If player 1:
      If the number of staples left is less than 1,
         Set reload needed flag high
      Else, decrement staple count
   If player 2:
      If the number of staples left is less than 1,
         Set reload needed flag high
      Else, decrement staple count
Helpers.c - low level functions for use in any module

Wait (takes a given amount of ms and delays that amount) {
    Set delay timer
    Do nothing until it expires
}
# ifndef HELPERS
# define HELPERS

//DEFINITIONS
//Boolean definitions
# define TRUE 1
# define FALSE 0
# define SUCCESS 1
# define FAILURE 0
# define OPEN 1
# define CLOSED 0

//Levels of difficulty
# define EASY 1
# define MEDIUM 2
# define HARD 3

//Timer says hi
# define TIMER_TIME_BASE 1 //number of milliseconds equal to one tick

//FUNCTION PROTOTYPES
void Wait(int ticks);
#endif

//wait for a given number of milliseconds

//INCLUDES
#include "headers.h"

void Wait(int ticks)
{
    //uses timer 0, which is one of 8 possible timers
    TMRS12_InitTimer(0,ticks);
    while(TMRS12_IsTimerExpired(0) != TMRS12_EXPIRED);
}
```c
Interface.h

#ifndef INTERFACE
#define INTERFACE

//FUNCTION PROTOTYPES
char GetLevel(void);
void InitDispensePrize(void);
char CheckDispensePrize(void);
char LevelSubmitted(void);
char ResetButton(void);
#endif

Interface.c

//CONSTANT DEFINITIONS
#define LEVEL_TRANSITION_EM 220 //voltage in mV at which level transitions from easy to medium
#define LEVEL_TRANSITION_MH 600    //voltage in mV at which level transitions from medium to hard
#define FLYWHEEL_WARMUP_TIME 2000 //amount of time required to spin up the flywheel
#define DISPENSE_TIME 30      //time in ms that prize dispensing motor is on
#define RELAX_TIME 3000      //time to relax before the game is officially over
#define RESET_BUTTON_TIMEOUT 3000  //time in ms you have to hold down reset button for a reset

//MODULE VARIABLE DEFINITIONS
static char levelSubmitted = 0;    //state variable set to 1 if start button pressed
static char resetButtonDown = 0;   //state variable set to 1 if button is down
static char flywheelReady = 0; //state variable set to 1 when the flywheel is warmed up and ready to go
static char prizeDispensed = 0; //state variable set to 1 when the prize has been dispensed

//INCLUDES
#include "headers.h"

char GetLevel(void) {
    //AD0 - Analog level select, ranges between 0 and 1023
    short potVoltage = ADS12_ReadADPin(0);
    //printf("potV: %d\n",potVoltage);

    //return a level based on the pot voltage and transition points
    if(potVoltage < LEVEL_TRANSITION_EM) return EASY;
    else if(potVoltage < LEVEL_TRANSITION_MH) return MEDIUM;
```
else
    return HARD;
}

void LevelTest(void) {
    short potVoltage;
    char Level=GetLevel();
    while(TRUE){
        //while()
        potVoltage = ADS12_ReadADPin(0);
        printf("potV: %d\r\n",potVoltage);
        Wait(100);
    }
}

//Call this once to initialize prize dispensing
void InitDispensePrize(void) {
    TMRS12_InitTimer(3,FLYWHEEL_WARMUP_TIME); //start timer
    PTAD = (PTAD | BIT7HI); //turn on flywheel motor
}

/* PUBLIC FUNCTION: char CheckDispensePrize(void)
   description: returns true if prize dispensing is complete and the postgame is over
   pre: -
   post: -
   returns true or false */
char CheckDispensePrize(void) {
    if(flywheelReady == 0){ //haven't yet warmed up, so do it
        if(TMRS12_IsTimerExpired(3) == TMRS12_EXPIRED) { //check to see if our flywheel is ready to rock
            TMRS12_InitTimer(3,DISPENSE_TIME); //now this timer will control the length of time dispensing will take place
            flywheelReady = 1;
            PTT = (PTT | BIT5HI); //turn on dispensing motor
        }
        return FALSE;
    } else if(prizeDispensed == 0) { //ting-a-ling! we're ready to shoot business cards.
        //check if we should be done dispensing
        if(TMRS12_IsTimerExpired(3) == TMRS12_EXPIRED){
            TMRS12_InitTimer(3,RELAX_TIME); //now this timer will control the length of time dispensing
will take place

prizeDispensed = 1;
PTT = (PTT & BIT5LO); //after time's up, turn off both motors
PTAD = (PTAD & BIT7LO); //turn off flywheel

return FALSE;
}
else { //all done dispensing the prize. Let's relax for a bit, then return true
if(TMRS12_IsTimerExpired(3) == TMRS12_EXPIRED) {
    flywheelReady = 0;
    prizeDispensed = 0;
    return TRUE;
}
    return FALSE;
}

return FALSE;

char LevelSubmitted(void){
    // get current state of start button
    levelSubmitted = BIT2HI & PTAD;

    // returns 1 if start button pressed
    return levelSubmitted;
}

char ResetButton(void){
    //if button is down and it's a new thang
    if ((PTAD & BIT2HI) && !resetButtonDown) {
        //set the timer and set state of button to down
        TMRS12_InitTimer(5, RESET_BUTTON_TIMEOUT);
        resetButtonDown = 1;
        return 0;
    }

    //if the bit isn't high but our state doesn't reflect that, update the state
    if (!((PTAD & BIT2HI) && resetButtonDown)) {
        resetButtonDown = 0;
        return 0;
    }
// if button isn't down
if (!(PTAD & BIT2HI)) {
    resetButtonDown = 0;
    return 0;
}

// if button is down and we're holding it there
if ((PTAD & BIT2HI) && resetButtonDown){
    // check if the timer has expired
    // if timer 5 has expired, we set the reset flag
    if (TMRS12_IsTimerExpired(5) == TMRS12_EXPIRED) {
        printf("YOU JUST RESET THE GAME!!!\r\n");
        return 1;
    } else return 0;
}
LED.h

/*
LED.h

Description: This is the header file the module which provides all of the functions necessary to control the LED's used in the game */

#ifndef LED
#define LED

#define OFF 0
#define ON 1
#define BLINK 2
#define RELOAD_LIGHT_1 0
#define RELOAD_LIGHT_2 1
#define STAPLER_LIGHT_1 2
#define STAPLER_LIGHT_2 3
#define PAYCHECK_LIGHT 4
#define START_LIGHT 5
#define ERGO_INSTRUCT_LIGHT 6

/* Function Prototypes */

// state is OFF, ON, or BLINK;
void ControlLED(char state, int lightID);
void LEDTest(void);
void TurnAllOff(void);

// used as accessory function to ControlLED
static void TurnOffLED(int lightID);
static void TurnOnLED(int lightID);
void BlinkLED(void);

#endif

LED.c

/*
LED.c
Description: This is the code file the module which provides all of the functions necessary to control the LED's used in the game

```c
#include "headers.h"
#include "led.h"

/*
 Constants
 */
#define BLINK_RATE 300
#define NUM_LEDS 6   //the total number of LEDs we are controlling
#define TIMER_NUMBER 4

/* Module Variables */
static char LED_State [NUM_LEDS + 1];
static char IsBlinking [NUM_LEDS + 1];

void LEDTest()
{
    int i;
    int time;

    // Cycle through all LEDs and turn on then off - then blink them for awhile
    for(i=0; i<NUM_LEDS; i++)
    {
        time = TMRS12_GetTime();
        ControlLED(ON, i);
        while((TMRS12_GetTime() - time) < BLINK_RATE);
        ControlLED(OFF, i);
        ControlLED(BLINK, i);
    }

    time = TMRS12_GetTime();
    while((TMRS12_GetTime() - time) < 5000) BlinkLED();

    for(i=0; i<NUM_LEDS; i++) TurnOffLED(i);
```
void ControlLED(char state, int lightID)
{
    if(state == ON)
    {
        IsBlinking[lightID] = 0;
        TurnOnLED(lightID);
    }
    else if (state == OFF)
    {
        IsBlinking[lightID] = 0;
        TurnOffLED(lightID);
    }
    else
    {
        IsBlinking[lightID] = 1;
        BlinkLED();
    }
}

void BlinkLED(void)
{
    int i;

    // Check to see if blink timer needs first initialization
    if(TMRS12_IsTimerActive(TIMER_NUMBER) == TMRS12_NOT_ACTIVE && TMRS12_IsTimerExpired(TIMER_NUMBER) != TMRS12_EXPIRED)
    {
        TMRS12_InitTimer(TIMER_NUMBER, BLINK_RATE);
        //printf("\r\ninitialized blink timer\r\n\r\n");
    }

    // If the blink timer has expired, we need to change the state of the LEDs (otherwise do nothing)
    if(TMRS12_IsTimerExpired(TIMER_NUMBER) == TMRS12_EXPIRED)
    {
        //printf("blink timer expiration detected\r\n");
        for (i=0;i<NUM_LEDS;i++)  //cycle through each LED
        {
        
    
}
//check if each LED is supposed to be blinking
if(IsBlinking[i])
{
    //if it was on, turn it off
    if(LED_State[i] == ON) TurnOffLED(i);
    //if it was off, turn it on!
    else TurnOnLED(i);
}
// Reset the timer
TMRS12_InitTimer(TIMER_NUMBER, BLINK_RATE);
//printf("blink timer just reset\r\n");
}

static void TurnOffLED(int lightID)
{
    switch(lightID)
    {
    case RELOAD_LIGHT_1:
        PTT = PTT & BIT6LO;
        LED_State[RELOAD_LIGHT_1] = 0;
        break;
    case RELOAD_LIGHT_2:
        PTT = PTT & BIT7LO;
        LED_State[RELOAD_LIGHT_2] = 0;
        break;
    case STAPLER_LIGHT_1:
        PTAD = PTAD & BIT3LO;
        LED_State[STAPLER_LIGHT_1] = 0;
        break;
    case STAPLER_LIGHT_2:
        PTAD = PTAD & BIT4LO;
        LED_State[STAPLER_LIGHT_2] = 0;
        break;
    case PAYCHECK_LIGHT:
        PTAD = PTAD & BIT5LO;
        LED_State[PAYCHECK_LIGHT] = 0;
        break;
    case START_LIGHT:
PTAD = PTAD & BIT6LO;
LED_State[START_LIGHT] = 0;
break;
default:
    break;
}

static void TurnOnLED(int lightID)
{
    switch(lightID)
    {
    case RELOAD_LIGHT_1:
        PTT = PTT | BIT6HI;
        LED_State[RELOAD_LIGHT_1] = 1;
        break;
    case RELOAD_LIGHT_2:
        PTT = PTT | BIT7HI;
        LED_State[RELOAD_LIGHT_2] = 1;
        break;
    case STAPLER_LIGHT_1:
        PTAD = PTAD | BIT3HI;
        LED_State[STAPLER_LIGHT_1] = 1;
        break;
    case STAPLER_LIGHT_2:
        PTAD = PTAD | BIT4HI;
        LED_State[STAPLER_LIGHT_2] = 1;
        break;
    case PAYCHECK_LIGHT:
        PTAD = PTAD | BIT5HI;
        LED_State[PAYCHECK_LIGHT] = 1;
        break;
    case START_LIGHT:
        PTAD = PTAD | BIT6HI;
        LED_State[START_LIGHT] = 1;
        break;
    default:
        break;
    }
void TurnAllOff(void)
{
    int i;
    for (i=0;i<NUM_LEDS;i++) ControlLED(OFF,i);
}
Paper.h

#ifndef PAPER
#define PAPER

//motorID constants
#define PAPER1 0 //motorIDs, which correspond the the array indexes of paper stack state variables
#define PAPER2 1

//function prototypes
void PopupTest(char motorID);
char CalibratePopup(char motorID);
char GetPopupPos(char motorID);
void SetPopup(char motorID, char finalPos);
char CheckAllPopupsForArrival(void);
static char CheckPopupArrival(char motorID);
static int CalculateMoveDuration(int diffPos, char direction, char motorID);
static void StartMotor(char motorID, char direction, unsigned char dutyCycle);
static void StopMotor(char motorID);
static char JustHitLimit(char motorID);
static char AtLimit(char motorID);

#endif

Paper.c

///////////////////////////////////
//  paper.c  //
///////////////////////////////////

/*
OVRERVIEW: the paper.c module controls the paper stacks using three public functions.
1) CalibratePopup must be run before any other paper functions are used. As a precondition, the paper
stacks must not be at their limit switches. The paper stacks will end at the down position.
2) SetPopup is used to set the position of a chosen paper stack to an absolute position anywhere from 0
(bottom) to 100 (top)
3) CheckAllPopupsForArrival must be called continuously in an event-checking loop. It makes sure the paper
stacks stop when they hit their destinations or limits, and returns true if all the paper stacks are in
place
*/

//INCLUDES
#include "headers.h"  //includes all headers

//constant definitions
//direction constants
#define UP 1   //paper stack moving upwards
#define DOWN (-1) //paper stack moving downwards
#define TOP 1 //paper stack limit at the top of travel
#define BOTTOM (-1) //paper stack limit at the bottom of travel

//limits definitions and adjustment constants
#define MAX_PAPER_POSITION 100 //arbitrary max end of paper position range
#define MIN_PAPER_POSITION 0 //arbitrary min end of paper position range
#define DEF_PAPER_DUTY 60 //default duty cycle for paper movement
#define DUTY_UP_FACTOR 150 //multiplier for duty cycle when going up

//module variable definitions
//array variables are sized according to the number of paper stacks, which equals 2 for this game
static char paperMoving[2] = {0,0}; //are the paper stacks moving up, down, or 0?
static char paperLastLimit[2] = {0,0}; //1 if paper is at a limit, 0 if it isn't at a limit.

//RSVP inputs, outputs, and timers for the paper stacks
static int paperPosition[2] = {0, 0}; //destination position of paper (from 0 = max down to 100 = max up)
assume a start at the middle
static char limitPortsHI[2] = {BIT4HI, BIT5HI}; //which PTM ports are the paper stacks attached to?
static char motorDirectionPortsHI[2] = {BIT3HI, BIT4HI}; //PTM
static char motorEnablePorts[2] = {PWMS12_CHAN0, PWMS12_CHAN1}; //PTM
static char motorTimerNumbers[2] = {1, 2}; //reserve timers one and two

//the following variables need to be set during the calibration routine:
static int upTicks[2] = {900, 900}; //IF MOVING UP, how many ticks does it take to go all the way from
lower to upper limit? Needs to be calibrated.
static int downTicks[2] = {900, 900}; //IF MOVING DOWN, how many ticks does it take to go all the way from
upper to lower limit? Needs to be calibrated.

/* PUBLIC FUNCTION: void PopupsTest(char motorID)
   description: runs a routine to test the given popup
   pre: attach popup prototype with motor and limit switch
   post: -
   returns nothing
*/
void PopupTest(char motorID){

int i;

//Tell the world what's up
if(motorID == PAPER1){
    printf("Popup test for PAPER STACK 1: \r\n");
}
else if(motorID == PAPER2){
    printf("Popup test for PAPER STACK 2: \r\n");
}

//check limit switch
printf("Checking limit switch \r\n");
for(i=0; i<100; i++){
    Wait(400);
    if(AtLimit(motorID))
        printf("At limit \r\n");
    else
        printf("NOT at limit \r\n");
    if(JustHitLimit(motorID))
        printf("Just hit limit \r\n");
}

//Calibrate the first popup
//CalibratePopup(motorID);

/*
//Check SetPopup function
SetPopup(motorID, 60);
printf("Going to abs 60 \r\n");
while(!CheckAllPopupsForArrival());
printf("Arrived at abs 60 \r\n");
SetPopup(motorID, 20);
printf("Going to abs 20 \r\n");
while(!CheckAllPopupsForArrival());
printf("Arrived at abs 20 \r\n");
SetPopup(motorID, 100);
printf("Going to abs 100 \r\n");
while(!CheckAllPopupsForArrival());
printf("Arrived at abs 100 \r\n");
*/
SetPopup(motorID, 0);
printf("Going to abs 0 \r\n");
while(!CheckAllPopupsForArrival());
printf("Arrived at abs 0 \r\n");
*/

} /* PUBLIC FUNCTION: char CalibratePopup(char motorID)
description: automatically calibrates paper stack to know how long to pulse
pre: stacks are in the middle of travel, away from limits
post: stacks in the lower position
return char: 1 if calibration was successful, 0 if it failed */
char CalibratePopup(char motorID){
    int startTime;

    printf("\r\nBeginning popup calibration: \r\n");

    //return an error and do not calibrate if you start at a limit, since you don't know which limit you're
    //at
    if(AtLimit(motorID)){
        printf("ERROR: Calibration must start with paper stack between limits \r\n");
        return 0;
    }

    //go down until the lower limit is hit
    SetPopup(motorID, MIN_PAPER_POSITION);
    printf("1) Travelling to lower limit \r\n");
    //MAY NEED TO DEBOUNCE HERE
    while(!CheckPopupArrival(motorID)); //kill time until the stack arrives where it should be
    printf(" Reached the lower limit \r\n");
    Wait(1000); //pause to let the motor chill out for a wee bit

    //start a timer
    startTime = TMRS12_GetTime();

    //start going up until the upper limit is hit
    SetPopup(motorID, MAX_PAPER_POSITION);
    printf("2) Travelling to upper limit \r\n");
    while(!CheckPopupArrival(motorID));
    printf(" Reached the upper limit \r\n");
//stop the timer and calculate the elapsed time
upTicks[motorID] = TMRS12_GetTime() - startTime;
printf("upTicks = %d \r\n", upTicks[motorID]);
Wait(1000); //pause to let the motor chill out for a wee bit

//start a timer
startTime = TMRS12_GetTime();

//now go down until the lower limit is hit
SetPopup(motorID, MIN_PAPER_POSITION);
printf("3) Travelling to lower limit \r\n");
while(!CheckPopupArrival(motorID));
printf(" Reached the lower limit \r\n");

//stop the timer and calculate the elapsed time
downTicks[motorID] = TMRS12_GetTime() - startTime;
printf("downTicks = %d \r\n", downTicks[motorID]);
Wait(1000); //pause to let the motor chill out for a wee bit

printf("Calibration complete!\r\n");
return 1;
}

//PUBLIC FUNCTION
//Given a motor, returns the current position of the motor
char GetPopupPos(char motorID){
  return paperPosition[motorID];
}

/* PUBLIC FUNCTION: void SetPopup(char motorID, char finalPos)
   description: given a final position, goes to it. It you set a max position as a final destination, it
goes until the limit is hit. Otherwise it sets the stop time for getting to an intermediate destination.
   pre: calibration must be performed before SetPopup can be called
   post: have to check the motor's arrival continuously until the motor has arrived, or suffer the collision
   consequences
   returns nothing */
void SetPopup(char motorID, char finalPos){
  char diffPos;
char direction = 0;
int numTicks;

printf("Setting motorID=%d popup to finalPos=%d \r\n", motorID, finalPos);

if(paperMoving[motorID] && (finalPos > MIN_PAPER_POSITION) && (finalPos < MAX_PAPER_POSITION)){ //if it's already moving, do nothing
    //printf("WARNING: Trying to move a motor that's already in motion \r\n");
    return ;
}

//If you're going to min or max, just plow forward until you hit a limit
if(finalPos <= MIN_PAPER_POSITION){
    if(paperPosition[motorID] == MIN_PAPER_POSITION) //we are already at min
        return;
    TMRS12_ClearTimerExpired(motorTimerNumbers[motorID]); //NOT NEEDED?
    StartMotor(motorID, DOWN, DEF_PAPER_DUTY); //go motor! No timer needed because we want it to go all the way to the limit
    paperPosition[motorID] = MIN_PAPER_POSITION; //will end up at the destination
    //******
    if(AtLimit(motorID)){ //if we're trying to go end to end, allow a grace period
        Wait(200);
    }
}

else if(finalPos >= MAX_PAPER_POSITION){
    if(paperPosition[motorID] == MAX_PAPER_POSITION) //we are already at max
        return;
    TMRS12_ClearTimerExpired(motorTimerNumbers[motorID]);
    StartMotor(motorID, UP, DEF_PAPER_DUTY);
    paperPosition[motorID] = MAX_PAPER_POSITION;

    if(AtLimit(motorID)){ //if we're trying to go end to end, allow a grace period
        Wait(200);
    }
}

else{ //you are actually in a happy place in between the limits
    if(finalPos > paperPosition[motorID]){ //going up
        direction = UP;
        diffPos = finalPos - paperPosition[motorID]; //should be a positive value
        //printf("going up a relative %d ticks \r\n", diffPos);
    }
}
else if(finalPos < paperPosition[motorID]){ //going down
direction = DOWN;
diffPos = paperPosition[motorID] - finalPos; //should be a positive value
    //printf("going down a relative %d ticks \r\n", diffPos);
}
    //update motor position and states
numTicks = CalculateMoveDuration(diffPos, direction, motorID);
TMRS12_InitTimer(motorTimerNumbers[motorID], numTicks);
StartMotor(motorID, direction, DEF_PAPER_DUTY);
paperPosition[motorID] = finalPos;
}

/* PUBLIC FUNCTION: char CheckAllPopupsForArrival(void)
   description: a checking and handling function that stops the paper stacks if they hit limits or run out of time. Use in an event-checking loop..
   pre: none
   post: motors stopped if they need to be
   return char: true if all systems are stopped, false if any paper stack is still in motion */
char CheckAllPopupsForArrival(void){
    char CPA1;
    char CPA2;
    CPA1 = CheckPopupArrival(PAPER1);
    CPA2 = CheckPopupArrival(PAPER2);
    //if nothing's moving, our job's done here
    if(CPA1 && CPA2){
        return TRUE; //all systems are stopped
    }
    return FALSE; //at least one of the stacks is still moving
}

/* PRIVATE FUNCTION: static char CheckPopupArrival(char motorID)
   description: a checking and handling function that stops the given paper stack if it hits its limits or run out of time.
   pre: none
   post: motor stopped if it needs to be
   return char: true if the motor is stopped, false if it is still in motion */
static char CheckPopupArrival(char motorID){
//if stack is moving, stop if needed
if(paperMoving[motorID] != 0){  //if paper is moving
//printf("PAPER IS MOVING \r\n");
if(JustHitLimit(motorID)){
    //make sure if we hit a limit, we update our position to know where we are
    if(paperMoving[motorID] == DOWN) {
        printf("paperPosition[0]=%d \r\n", paperPosition[0]);
        printf("paperPosition[1]=%d \r\n", paperPosition[1]);
        printf("paperMoving[0]=%d \r\n", paperMoving[0]);
        printf("paperMoving[1]=%d \r\n", paperMoving[1]);
        printf("paperLastLimit[0]=%d \r\n", paperLastLimit[0]);
        printf("paperLastLimit[1]=%d \r\n", paperLastLimit[1]);

        paperPosition[motorID] = MIN_PAPER_POSITION;
        printf("Reinterpreted position of motor %d to MIN \r\n", motorID);
    }
    if(paperMoving[motorID] == UP){
        paperPosition[motorID] = MAX_PAPER_POSITION;
        printf("Reinterpreted position of motor %d to MAX \r\n", motorID);
    }
    TMRS12_ClearTimerExpired(motorTimerNumbers[motorID]);
    StopMotor(motorID);
}
if(TMRS12_IsTimerExpired(motorTimerNumbers[motorID]) == TMRS12_EXPIRED){
    TMRS12_ClearTimerExpired(motorTimerNumbers[motorID]);
    StopMotor(motorID);
}
return FALSE;  //one of the stacks is still moving
}
return TRUE;  //stacks aren't moving

/* PRIVATE FUNCTION: static int CalculateMoveDuration(int diffPos, char direction, char motorID)
description: given a difference in position, find how long it should move
pre: calibration completed
post: none
return int: the number of ticks the paper stack should move for */
static int CalculateMoveDuration(int diffPos, char direction, char motorID){
    int posPercent = (100 * diffPos) / (MAX_PAPER_POSITION - MIN_PAPER_POSITION); //ranges from 0 (down) to
```c
int ticksMove;
//printf("posPercent = %d \n", posPercent);

if (direction == UP){
    ticksMove = posPercent * (upTicks[motorID] / 100);
    printf("MoveDurationUp = %d \n", ticksMove);
    return (ticksMove);
}
else if (direction == DOWN){
    ticksMove = posPercent * (downTicks[motorID] / 100);
    printf("MoveDurationDown = %d \n", ticksMove);
    return (ticksMove);
}  
else{  
    printf("WARNING: Trying to calculate a move duration without a direction!");
    return 0;
}

/* PRIVATE FUNCTION: static void StartMotor(char motorID, char direction, unsigned char dutyCycle)   description: starts the selected motor going in a direction with a given duty cycle, taking into account the weight of the popup   pre: motor attached, voltage and duty high enough that the motor doesn't stall. See L239B data sheet for motor control details.   post: motor moving with given speeed and direction, paperMoving properly updated   returns nothing */
static void StartMotor(char motorID, char direction, unsigned char dutyCycle){
if(direction == UP){
    PTT = PTT | motorDirectionPortsHI[motorID]; //set direction bit high
    PWMS12_SetDuty((dutyCycle*DUTY_UP_FACTOR)/ 100, motorEnablePorts[motorID]);  //use PWM output to set the motor in motion
}  
else if(direction == DOWN){
    PTT = PTT & motorDirectionPortsLO[motorID]; //set direction bit low
    PWMS12_SetDuty(dutyCycle, motorEnablePorts[motorID]);
}  
else{  //if the direction is zero, stop the motor
    PWMS12_SetDuty(0, motorEnablePorts[motorID]);
}
```
void StopMotor(char motorID)
{
    printf("STOPPING MOTOR %d \r\n", motorID);
    StartMotor(motorID, 0, 0);
}

char JustHitLimit(char motorID){
    char curLimitState = AtLimit(motorID);
    if(paperLastLimit[motorID] != curLimitState){
        paperLastLimit[motorID] = curLimitState;
        if(curLimitState == TRUE) //return true if it just hit a limit, but not if it just came off one
            printf("JUST HIT LIMIT for motorID %d \r\n", motorID);
        return TRUE;
    }
    return FALSE;
}

char AtLimit(char motorID){
if(PTM & limitPortsHI[motorID]){  //if paper stack limit switch is high, then it is depressed and in the
    middle of travel (not at a limit)
    Wait(10);
    return FALSE;
} 
Wait(10); 
return TRUE; 
}
Play.h

#ifndef PLAY
#define PLAY

//FUNCTION PROTOTYPES
static void MainEventChecker(void);
static void RunChecks(void);
static void InitMAIN(void);
static void InitRESET(void);
static void InitWAITCOIN(void);
static void InitWAITSTART(void);
static void InitPREGAME(void);
static void InitGAME(void);
static void InitPOSTGAME(void);
static char IsWinner(void);
static char CheckCoin(void);

#endif

Play.c

//Uncomment the following line to run the full set of tests
//#define TEST

/*
THE STAPLER GAME DOCUMENTATION

OUTPUTS (8)
 T0 - Paper stack 1 motor control 1 (C) (PWM group 0)
 T1 - Paper stack 2 motor control 1 (C) (PWM group 0)
 T2 - Boss/ergo servo motor signal (PWM group 1)
 T3 - Paper stack 1 motor control 2 (D)
 T4 - Paper stack 2 motor control 2 (D)
 T5 - Prize motor signal
 T6 - Reload light 1
 T7 - Reload light 2
 AD3 - Stapler light 1
 AD4 - Stapler light 2
 AD5 - Paycheck light
 AD6 - Level select and start light
 AD7 - Prize flywheel motor

INPUTS (8)
*/
AD0 - Analog level select
AD1 - Coin sensor
AD2 - Start button
M0 - Stapler press 1
M1 - Stapler press 2
M2 - Stapler reload 1
M3 - Stapler reload 2
M4 - Paper stack 1 limit switches
M5 - Paper stack 2 limit switches

TIMERS (8)
0 - Wait helper function & staple test
1 - Paper stack 1
2 - Paper stack 2
3 - Prize dispenser
4 - LED blinking
5 - Reset button
6 - Ergo
7 - UNUSED!

*/

/*
PLAY MODULE
Description: This is the highest-level module. It calls all initializations necessary to play the game and then begins an infinite loop where it is constantly listens for events and then responds to the events in an appropriate way.
*/

//Management states
#define AWAY 0
#define ERGO 1
#define BOSS 2

//GameStage definitions
#define RESET 1 //Transition to start conditions
#define WAITCOIN 2 //The period when we are waiting for a player to insert a coin
#define WAITSTART 3 //The period when we are waiting for a player to select a level and hit start
#define PREGAME 4 //From coin insertion until the start of play begins
#define GAME 5 //The main game until a winner is declared
#define POSTGAME 6 //Winning sequence and prize delivery
//Game-winning scores!
#define EASY_WIN_SCORE 40   //number of staples needed to win the easy level
#define MEDIUM_WIN_SCORE 60 //number of staples needed to win the medium level
#define HARD_WIN_SCORE 60   //number of staples needed to win the hard level

//Game timeout definition
#define TIMEOUT 480000
#define PREGAME_LENGTH 2000 //how long the pregame sequence lasts (in ms)
#define POSTGAME_LENGTH 6000 //how long the postgame sequence lasts (in ms)
#define ERGO_TIME 5000 //how long you have till the ergo shows up
#define ERGO_GRACE_PERIOD 500 //number of milliseconds grace period before getting caught by ergo
#define ERGO_PUNISHMENT 10 //number of papers HR adds to the stack to punish you
#define ERGO_TIME_UP 3000 //how much time you have to spend with her watching you!
#define LEVEL_SPEED_MULTIPLIER_E 60
#define LEVEL_SPEED_MULTIPLIER_M 70

//MODULE VARIABLE DEFINITION
static char GameStage = RESET; //a variable that keeps track of what stage of the game we are in and is
                                   //used by the main event checker.
static char Level = 0; //Level keeps track of the difficulty level that the user chooses
static int P1score = 0; //cumulative player 1 score
static int P2score = 0; //cumulative player 2 score
static char PaperStack1Pos = 0; //position of paper stack 1 from 0 to 100
static char PaperStack2Pos = 0; //position of paper stack 2 from 0 to 100
static char Management = AWAY; //who's looking over the cubicle walls?
static char ErgoAlreadyCame = 0; //goes high if the ergonomist has already wreaked her havoc
static char P1gotcha = 0; //goes high if the ergonomist catches you. Prevents double jeopardy
static char P2gotcha = 0;
static char coinInserted;
static int StageStartTime = 0; //the clock tick time when the game started
static int ErgoStartTime = 0; //time when the ergonomist pops up (if lvl 3)

//INCLUDES
#include "headers.h"
//The REAL main function, which is disabled if we are in testing mode
#ifndef TEST
void main(void)
{
    //Perform main initialization sequence
    InitMAIN();
    printf("main initialized\r\n");

    //Initialize reset once
    InitRESET();
    printf("just reset\r\n");

    //Run main event checker in an endless loop
    while(TRUE)
    {
        MainEventChecker();
    }
    return;
}
#endif

//MainEventChecker
//This is our main event loop, which will always be running in a loop.
//Functions called within MainEventChecker must be non-blocking
static void MainEventChecker(void)
{
    //check for interrupts and refresh continuous state variables
    RunChecks();

    switch(GameStage)
    {
    case RESET:
        //Once mechanical elements are in place and no longer moving, you're done resetting. Wait for coin!
        if(CheckAllPopupsForArrival())
        {
            //Initialize waitcoin once
            printf("in reset phase\r\n");
            TurnAllOff();
        }
    }
case WAITCOIN:
    //When a coin is sensed, go to waitstart
    if(CheckCoin())
    {
        printf("just sensed coin\n");
        //Initialize waitstart once
        InitWAITSTART();
        GameStage=WAITSTART;
    }
    break;

case WAITSTART:
    //check the level
    Level = GetLevel();

    //When the start button is pressed, go to the pregame
    if(LevelSubmitted())
    {
        printf("level has been submitted: %d\n",Level);
        //Initialize pregame once
        InitPREGAME();
        GameStage=PREGAME;
    }
    break;

case PREGAME:
    if(CheckAllPopupsForArrival()){
        //Initialize game once
        InitGAME();
        GameStage=GAME;
    }
    break;

case GAME:
    //Check for winner. If there is one, end the game.
if(IsWinner())
{
    //Initialize postgame once
    printf("we have a winner!!!\n\n");
    InitPOSTGAME();
    GameStage=POSTGAME;
    break;
}

switch(Level)
{
case EASY:
    //Check if player 1 has stapled
    if(CheckStaplePress(1,Level))
    {
        //Increment score
        P1score++;
        //Update position of paper stack
        PaperStack1Pos = 100 - (100 * P1score) / (100 * EASY_WIN_SCORE /
LEVEL_SPEED_MULTIPLIER_E);
        if(PaperStack1Pos < GetPopupPos(PAPER1)) //only go down!
            SetPopup(PAPER1,PaperStack1Pos);
    }
    //Check if player 2 has stapled
    if(CheckStaplePress(2,Level))
    {
        //Increment score
        P2score++;
        //Update position of paper stack
        PaperStack2Pos = 100 - (100 * P2score) / (100 * EASY_WIN_SCORE /
LEVEL_SPEED_MULTIPLIER_E);
        if(PaperStack2Pos < GetPopupPos(PAPER2)) //only go down!
            SetPopup(PAPER2,PaperStack2Pos);
    }
    break;

case MEDIUM:
    //If player 1 has plenty of staples
    if(IsAmmoLeft(1))
    {

// Check if player 1 has stapled
if (CheckStaplePress(1, Level))
{
    // Increment score
    P1score++;
    // Update position of paper stack
    PaperStack1Pos = 100 - (100 * P1score) / (100 * EASY_WIN_SCORE / LEVEL_SPEED_MULTIPLIER_M);
    if (PaperStack1Pos < GetPopupPos(PAPER1)) // only go down!
        SetPopup(PAPER1, PaperStack1Pos);
}

// If they are out of staples, they need to reload
else
{
    // Illuminate the reload light only
    ControlLED(OFF, STAPLER_LIGHT_1);
    ControlLED(BLINK, RELOAD_LIGHT_1);

    // Check if player reloads
    if (HasReloaded(1))
    {
        // Illuminate the stapler lights only
        ControlLED(OFF, RELOAD_LIGHT_1);
        ControlLED(ON, STAPLER_LIGHT_1);
    }
}

// If player 2 has plenty of staples
if (IsAmmoLeft(2))
{
    // Check if player 1 has stapled
    if (CheckStaplePress(2, Level))
    {
        // Increment score
        P2score++;
        // Update position of paper stack
        PaperStack2Pos = 100 - (100 * P2score) / (100 * EASY_WIN_SCORE / LEVEL_SPEED_MULTIPLIER_M);
        if (PaperStack2Pos < GetPopupPos(PAPER2)) // only go down!
SetPopup(PAPER2,PaperStack2Pos);

// If they are out of staples, they need to reload
else
{
    // Illuminate the reload light only
    ControlLED(OFF,STAPLER_LIGHT_2);
    ControlLED(BLINK,RELOAD_LIGHT_2);

    // Check if player reloads
    if(HasReloaded(2))
    {
        // Illuminate the stapler lights only
        ControlLED(OFF,RELOAD_LIGHT_2);
        ControlLED(ON,STAPLER_LIGHT_2);
    }
}

break;

case HARD:

//first scenario//

// If it's past time for the ergonomist and she hasn't already come
if ((TMRS12_IsTimerExpired(6) == TMRS12_EXPIRED) && !ErgoAlreadyCame)
{
    ErgoAlreadyCame = 1;
    printf("ergo going up!\r\n");
    ErgoStartTime = TMRS12_GetTime();

    // Ergonomist attacks!
    SetServoAngle(ERGO_ANGLE);

    // Illuminate the ergonomist instruction
    ControlLED(ON,ERGO_INSTRUCT_LIGHT);
//Set gotcha flags to zero
P1gotcha = 0;
P2gotcha = 0;
}

////////////////////
//second scenario//
////////////////////

//If ergo has already wreaked her havoc, we treat it like it's MEDIUM
//Or if the ergo hasn't shown up yet at all
if (!ErgoIsUp())
{

    //The code for this case is identical to MEDIUM!
    //The code for this case is identical to MEDIUM!
    //The code for this case is identical to MEDIUM!
    //The code for this case is identical to MEDIUM!
    //The code for this case is identical to MEDIUM!

    //If player 1 has plenty of staples
    if(IsAmmoLeft(1))
    {
        //Check if player 1 has stapled
        if(CheckStaplePress(1,Level))
        {
            //Increment score
            P1score++;
            //Update position of paper stack
            PaperStack1Pos = 100 - (100 * P1score) / HARD_WIN_SCORE;
            if(PaperStack1Pos < GetPopupPos(PAPER1)) //only go down!
                SetPopup(PAPER1,PaperStack1Pos);
        }
    }

    //If they are out of staples, they need to reload
    else
    {
        //Illuminate the reload light only
        ControlLED(OFF,STAPLER_LIGHT_1);
        ControlLED(BLINK,RELOAD_LIGHT_1);

        //Check if player reloads
if(HasReloaded(1))
{
    //Illuminate the stapler lights only
    ControlLED(OFF,RELOAD_LIGHT_1);
    ControlLED(ON,STAPLER_LIGHT_1);
}

//If player 2 has plenty of staples
if(IsAmmoLeft(2))
{
    //Check if player 1 has stapled
    if(CheckStaplePress(2,Level))
    {
        //Increment score
        P2score++;
        //Update position of paper stack
        PaperStack2Pos = 100 - (100 * P2score) / HARD_WIN_SCORE;
        if(PaperStack2Pos < GetPopupPos(PAPER2)) //only go down!
            SetPopup(PAPER2,PaperStack2Pos);
    }
}

//If they are out of staples, they need to reload
else
{
    //Illuminate the reload light only
    ControlLED(OFF,STAPLER_LIGHT_2);
    ControlLED(BLINK,RELOAD_LIGHT_2);

    //Check if player reloads
    if(HasReloaded(2))
    {
        //Illuminate the stapler lights only
        ControlLED(OFF,RELOAD_LIGHT_2);
        ControlLED(ON,STAPLER_LIGHT_2);
    }
}
//third scenario/
////////////////////////////////////////////////////////////////////////////////////////////////

//If ergo is present and is in the process of wreaking havoc
if (ErgoAlreadyCame && ErgoIsUp())
{

    /*
    //If the grace period has expired, we can screw with them...
    if((TMRS12_GetTime()-ErgoStartTime) > ERGO_GRACE_PERIOD)
    {
    
    //Check if player has a staple event
    if(CheckStaplePress(1,Level))
    {
        //If gotcha flag is low, punish them! ha!
        if (!P1gotcha)
        {
            printf("gotcha, suckaaaaaa whose name is player 1!!!\r\n");

            //Decrement score
            P1score = P1score - ERGO_PUNISHMENT;
            if (P1score < 0)
                P1score = 5;
            //Normalize score to feed into setpopup
            PaperStack1Pos = 100 - (100 * P1score) / HARD_WIN_SCORE;
            //Move paper stack up
            SetPopup(PAPER1,PaperStack1Pos);
            //blink their stapler!
            ControlLED(BLINK,STAPLER_LIGHT_1);
            //Gotcha flag goes high
            P1gotcha = 1;
        }
    }

    if(CheckStaplePress(2,Level))
    {
        if (!P2gotcha)
        {
            //Decrement score
            P2score = P2score - ERGO_PUNISHMENT;
            if (P2score < 0)
                P2score = 5;
            //Normalize score to feed into setpopup
            PaperStack2Pos = 100 - (100 * P2score) / HARD_WIN_SCORE;
            //Move paper stack up
            SetPopup(PAPER2,PaperStack2Pos);
            //blink their stapler!
            ControlLED(BLINK,STAPLER_LIGHT_2);
            //Gotcha flag goes high
            P2gotcha = 1;
        }
    }

```
if (P2score < 0)
    P2score = 5;
//Normalize score to feed into setpopup
PaperStack2Pos = 100 - (100 * P2score) / HARD_WIN_SCORE;
//Move paper stack up
SetPopup(PAPER2, PaperStack2Pos);
//blink their stapler!
ControlLED(BLINK, STAPLER_LIGHT_2);
//Gotcha flag goes high
P2gotcha = 1;
}

*/
//If ergo is done wreaking havoc
if ((TMRS12_GetTime() - ErgoStartTime) > ERGO_TIME_UP)
{
    //put the management away
    ManagementAway();
    printf("ergo retreating\r\n");

    //relight staplers, turn off ergo light
    ControlLED(OFF, ERGO_INSTRUCT_LIGHT);
    ControlLED(ON, STAPLER_LIGHT_1);
    ControlLED(ON, STAPLER_LIGHT_2);
}

break;
}
break;
case POSTGAME:
    //When the prize has been successfully dispensed, reset the game!
    if (CheckDispensePrize()){
        GameStage=RESET;
        InitRESET();
    }
    break;
}
static void RunChecks(void)
{
    if (ResetButton() == 1)
    {
        //reset the game
        printf("resetting game now\r\n");
        GameStage = RESET;
    }
    //blink any LEDs that need blinkin'
    BlinkLED();
    //stop any paper stacks that have hit their limit switches
    CheckAllPopupsForArrival();
}

static void InitMAIN(void)
{
    printf("In MAIN initialization sequence\r\n");

    //Initialize a timer with a 1MS time base (each tick = 1ms)
    TMRS12_Init(TMRS12_RATE_1MS);

    //Initialize PWM
    PWMS12_Init();
    //20ms Period
    PWMS12_SetPeriod(0x5096, PWMS12_GRP1);

    //Initializes AD ports
    if(ADS12_Init("OOOOOIIA") != ADS12_OK)
        printf("ERROR: AD Initialization unsuccessful\r\n");
    // Initialize all PORT M Bits as inputs
    DDRM = 0x00;
    // Initialize all PORT T Bits as outputs
    DDRT = 0xFF;

    // Start with all bits off
    PTT = 0x00;
    PTM = 0x00;

    //Calibrate the paper stack motors (not necessary if values are hard-coded in)
static void InitRESET(void)
{
    printf("In RESET stage\r\n");
    //Set paper stacks to zero
    SetPopup(PAPER1,0);
    SetPopup(PAPER2,0);
    coinInserted = 0;
    //Illumination off
    TurnAllOff();
    //Put management away
    ManagementAway();
}

static void InitWAITCOIN(void)
{
    //printf("In WAITCOIN stage\r\n");
    //Illuminate paycheck slot only
    TurnAllOff();
    ControlLED(BLINK,PAYCHECK_LIGHT);
}

static char CheckCoin(void)
{
    // get current state of coin sensor
    coinInserted = BIT1HI & PTAD;
    // returns 1 if coin has been detected
    return coinInserted;
}
static void InitWAITSTART(void)
{
    printf("In WAITSTART stage\r\n");

    //Start game timeout timer (timer 7)
    // TMRS12_InitTimer(7,TIMEOUT);

    //Illuminate start light only
    TurnAllOff();
    ControlLED(BLINK,START_LIGHT);
}

static void InitPREGAME(void)
{
    printf("In PREGAME stage\r\n");

    //Paper stacks up
    SetPopup(PAPER1,100);
    SetPopup(PAPER2,100);

    //Set scores to zero
    P1score = 0;
    P2score = 0;

    //Refill staplers
    RefillStaplers();

    //Turn all lights off
    TurnAllOff();
}

static void InitGAME(void)
{
    printf("In GAME stage\r\n");

    //Illuminate staplers only
    TurnAllOff();
    ControlLED(ON,STAPLER_LIGHT_1);    ControlLED(ON,STAPLER_LIGHT_2);

    //If difficulty is hard, set ergo surprise time
if (Level == HARD)
{
    TMRS12_InitTimer(6, ERGO_TIME);
    ErgoAlreadyCame = 0;
}
}

static void InitPOSTGAME(void)
{
    char staplerlight;
    char winner;

    printf("In POSTGAME stage\r\n");

    if (P1score > P2score) {
        staplerlight = STAPLER_LIGHT_1;
        printf("PLAYER 1 WINS -> player 1 kicked your ASS, player 2!! \r\n");
        winner = PAPER1;
    } else {
        staplerlight = STAPLER_LIGHT_2;
        printf("PLAYER 2 WINS -> player 2 PWNED your lameness, player 1! \r\n");
        winner = PAPER2;
    }

    // Flash winner's stapler light
    TurnAllOff();
    ControlLED(BLINK,staplerlight);

    // Boss emerges
    SetServoAngle(BOSS_ANGLE);

    // Initialize the prize dispensers
    InitDispensePrize();
}

// Returns true if either player has exceeded the winning score
static char IsWinner(void)
{
    int winningScore;
// set the winning score based on the selected level
switch (Level)
{
    case EASY: winningScore = EASY_WIN_SCORE; break;
    case MEDIUM: winningScore = MEDIUM_WIN_SCORE; break;
    case HARD: winningScore = HARD_WIN_SCORE; break;
}

// return true if one of the players has won
if((P1score >= winningScore) || (P2score >= winningScore))
    return TRUE;

return FALSE;

// The main TESTING function
#ifdef TEST
void main(void)
{
    // Perform main initialization sequence
    InitMAIN();

    /*
    // Test coin sensor
    printf("Please insert a coin\n");
    while (!CheckCoin());
    printf("Coin check: CHECK!\n\n");

    // Test the level selection
    printf("Please select a level and hit start\n");
    LevelTest();
    printf("Level check: CHECK!\n\n");

    // Test the servo
    printf("Servo test beginning...\n");
    Wait(1000);
    printf("NOW!\n");
    ServoTest();
    printf("Servo check: CHECK!\n\n");
    */
}
#endif
//Test paper stacks
printf("Paper stack test beginning...");
Wait(1000);
printf("NOW!\n");
PopupTest(PAPER1);
PopupTest(PAPER2);
//Test stapler 1 for input
printf("\nStart stapling..." );
Wait(3000);
printf("NOW!!!!!\n");
StaplerTest();
printf("Staple check: CHECK!!\n");
//Test the reload functionality
printf("Please reload the stapler\n");
ReloadTest();
printf("Reload check: CHECK!!\n");
//Test all LEDs
printf("LED test beginning...");
Wait(1000);
printf("NOW!\n");
LEDTest();
printf("LED check: CHECK!!\n");
//Test prize dispensing ability
printf("Prize dispensing test beginning...");
InitDispensePrize();
while(CheckDispensePrize() == 0);
printf("Prize check: CHECK!!\n");
Wait(500);
printf("\nTesting complete! Have a nice life!\n");
return;

#endif
Servo.h

#ifndef SERVO
#define SERVO

//constant definitions
#define MAX_SERVO_ANGLE 90 //the max degrees of travel the servo can go
#define MIN_SERVO_DUTY 5 //the pwm that will put the servo at its min angle
#define MAX_SERVO_DUTY 10 //the pwm that will put the servo at its max angle
#define BOSS_ANGLE 90 //angle of servo that will pop up boss
#define ERGO_ANGLE 0 //angle of servo that will pop up ergonomist

//FUNCTION PROTOTYPES
void ServoTest(void);
void SetServoAngle(int angle);
void ManagementAway(void);
char ErgoIsUp(void);
static int ConvertAngleToDUTY(int angle);
#endif

Servo.c

//INCLUDES
#include "headers.h"
#include "servo.h"

static char ergoUp = 0;

void ServoTest(void)
{
  short i;
  for(i=0;i<=MAX_SERVO_ANGLE;i=i+5)
  {
    SetServoAngle(i);
    //kill some time
    Wait(100);
  }
  while(TRUE) {
    printf("BOSS \r\n");
    SetServoAngle(BOSS_ANGLE);
    //kill lots of time - it takes awhile for the motor to move, and we don't want to cut it off
    Wait(5000);
    SetServoAngle(ERGO_ANGLE);
printf("ERGO \r\n");
Wait(5000);
}

void SetServoAngle(int angle)
{
    //PWM grp1 period already set to 20ms in main initialization
    int duty;
    duty = ConvertAngleToDUTY(angle);
    //printf("duty cycle: %d\r\n",duty);
    if (angle == ERGO_ANGLE) ergoUp = 1;
    else ergoUp = 0;
    PWMS12_SetDuty((char) duty, PWMS12_CHAN2);
}
//Converts an angle to a duty cycle using info from the servo data sheet
static int ConvertAngleToDUTY(int angle)
{
    return (MIN_SERVO_DUTY*100 + (MAX_SERVO_DUTY-MIN_SERVO_DUTY) * (angle*100 / MAX_SERVO_ANGLE))/100;
}

char ErgoIsUp(void)
{
    return ergoUp;
}

void ManagementAway(void)
{
    int duty;
    ergoUp = 0;
    duty = (MIN_SERVO_DUTY + MAX_SERVO_DUTY)/2;
    PWMS12_SetDuty((char) duty, PWMS12_CHAN2);
}
Stapler.h

#ifndef STAPLER
#define STAPLER

//FUNCTION PROTOTYPES
void StaplerTest(void);
void ReloadTest(void);
char CheckStaplePress(int playerID, char level);
char HasReloaded(int playerID);
char IsAmmoLeft(int playerID);
static char CheckStapler(int playerID, char staplerState);
static void UsedAStaple(int playerID);
void RefillStaplers(void);
#endif

Stapler.c

/*OUTPUTS:
 T6 - Reload light 1
 T7 - Reload light 2
 AD3 - Stapler light 1
 AD4 - Stapler light 2
*/

#define STAPLER_CAPACITY 20 //number of staples that a stapler can hold

#include "headers.h"

static int P1staples = STAPLER_CAPACITY;  //number of staples in player 1's stapler
static int P2staples = STAPLER_CAPACITY;  //number of staples in player 2's stapler
static int stapler1down = 0;
static int stapler2down = 0;
static int player1needsreload = 0;
static int player2needsreload = 0;
static int player1hasopened = 0;
static int player2hasopened = 0;
static int player1hasclosed = 0;
static int player2hasclosed = 0;
void StaplerTest(void)
{
    int score = 0;
    //goes for specific amount of time
    TMRS12_InitTimer(0,1000);
    while(TMRS12_IsTimerExpired(0) != TMRS12_EXPIRED)
    {
        if (!player1needsreload)
        {
            if (CheckStaplePress(1, MEDIUM)) score++;  //checks stapler for staple events
        }
    }
    //prints number of staples sensed
    printf("Number of staples sensed is %d\r\n",score);
}

void ReloadTest(void)
{
    player1needsreload = 1;
    while(!HasReloaded(1));
}

char CheckStaplePress(int playerID, char level)
{
    //switch for each player:
    switch (playerID)
    {
        case 1:
            //if stapler is down and current state is 0
            if ((PTM&BIT0HI) && !stapler1down)
            {
                stapler1down = 1;  //set current state to 1
                if (level > EASY) UsedAStaple(1);
                printf("stapler 1 just got pressed!\r\n");
                return 1;
            }
            //if stapler is not down and current state is 1
            else if (!(PTM&BIT0HI) && stapler1down)
            {
                stapler1down = 0;  //set current state to 0
            }
}

//...
//printf("stapler 1 just got released!
\n");
return 0;

case 2:
//if stapler is down and current state is 0
if ((PTM&BIT1HI) && !stapler2down)
{
stapler2down = 1;       //set current state to 1
if (level > EASY) UsedAStaple(2);
printf("stapler 2 just got pressed!
\n");
return 1;
}
//if stapler is not down and current state is 1
else if (!(PTM&BIT1HI) && stapler2down)
{
stapler2down = 0;       //set current state to 0
//printf("stapler 2 just got released!
\n");
}
return 0;

char HasReloaded(int playerId)
{
    //switch for each player:
    switch (playerId)
    {
    case 1:
        if (player1needsreload)
        {
            //check for stapler opening
            if (!player1hasopened) player1hasopened = CheckStapler(1,OPEN);
            //check for stapler closing
            if (!player1hasclosed && player1hasopened) player1hasclosed = CheckStapler(1,CLOSED);
            //if both conditions have been satisfied
            if (player1hasopened && player1hasclosed)
            {
                //refill the stapler
                P1staples = STAPLER_CAPACITY;
            }
        }
        break;
    case 2:
        if (player2needsreload)
        {
            //check for stapler opening
            if (!player2hasopened) player2hasopened = CheckStapler(2,OPEN);
            //check for stapler closing
            if (!player2hasclosed && player2hasopened) player2hasclosed = CheckStapler(2,CLOSED);
            //if both conditions have been satisfied
            if (player2hasopened && player2hasclosed)
            {
                //refill the stapler
                P2staples = STAPLER_CAPACITY;
            }
        }
        break;
    default:
        return 0;
    }
}
// reset all reload-related states
player1needsreload = 0;
player1hasopened = 0;
player1hasclosed = 0;
return 1;
} else return 0;
}

case 2:
  if (player2needsreload)
  {
  // check for stapler opening
  if (!player2hasopened) player2hasopened = CheckStapler(2,OPEN);
  // check for stapler closing
  if (!player2hasclosed && player2hasopened) player2hasclosed = CheckStapler(2,CLOSED);
  // if both conditions have been satisfied
  if (player2hasopened && player2hasclosed)
  {
  // refill the stapler
  P2staples = STAPLER_CAPACITY;
  // reset all reload-related states
  player2needsreload = 0;
  player2hasopened = 0;
  player2hasclosed = 0;
  return 1;
  } else return 0;
  }
  
// refill both staplers
void RefillStaplers(void){
  P2staples = STAPLER_CAPACITY;
  P1staples = STAPLER_CAPACITY;
  player1needsreload = 0;
  player2needsreload = 0;
}

static char CheckStapler(int playerId, char staplerState)
{
switch(playerID)
{
    case 1:
        switch(staplerState)
        {
            case OPEN:
                //if stapler is open and this is a new thang
                if (((PTM&BIT2HI) && !player1hasopened))
                {
                    player1hasopened = 1;       //player 1 has opened stapler
                    printf("stapler 1 just got opened!\r\n");
                    return 1;
                } else return 0;
            case CLOSED:
                //if stapler is closed and this is a new thang
                if (!(PTM&BIT2HI) && !player1hasclosed)
                {
                    player1hasclosed = 1;       //player 1 has closed stapler
                    printf("stapler 1 just got closed!\r\n");
                    return 1;
                } else return 0;
        }
    case 2:
        switch(staplerState)
        {
            case OPEN:
                //if stapler is open and this is a new thang
                if (((PTM&BIT3HI) && !player2hasopened))
                {
                    player2hasopened = 1;       //player 2 has opened stapler
                    printf("stapler 2 just got opened!\r\n");
                    return 1;
                } else return 0;
            case CLOSED:
                //if stapler is closed and this is a new thang
                if (!(PTM&BIT3HI) && !player2hasclosed)
                {
                    player2hasclosed = 1;       //player 2 has closed stapler
                    printf("stapler 2 just got closed!\r\n");
                    return 1;
                } else return 0;
        }
}
static void UsedAStaple(int playerID)
{
    switch(playerID)
    {
        case 1:
            if (P1staples <= 1)
            {
                player1needsreload = 1;
                break;
            } else
            {
                P1staples--;             
                break;
            }
        case 2:
            if (P2staples <= 1)
            {
                player2needsreload = 1;
                break;
            } else
            {
                P2staples--;             
                break;
            }
    }
}

char IsAmmoLeft(int playerID)
{
    if (playerID == 1) return !player1needsreload;
    if (playerID == 2) return !player2needsreload;
}
#ifndef HEADERS
#define HEADERS

//All of the headers needed for mystapler!

//Proven code
#include <stdio.h>
#include <ME218_C32.h>
#include <timers12.h>
#include <PWMS12.h>
#include <ADS12.h>

//Our code
#include "play.h"
#include "stapler.h"
#include "paper.h"
#include "servo.h"
#include "interface.h"
#include "LED.h"
#include "helpers.h"

#endif
# Appendix B

## BOM

<table>
<thead>
<tr>
<th>Category</th>
<th>Item</th>
<th>Description</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Electronic</strong></td>
<td>2N7000</td>
<td>N-channel mosfet</td>
<td>6</td>
<td>$0.50</td>
</tr>
<tr>
<td></td>
<td>LM239B</td>
<td>Motor driver chip</td>
<td>2</td>
<td>$2</td>
</tr>
<tr>
<td></td>
<td>74HC14</td>
<td>Hex inverter</td>
<td>1</td>
<td>$1</td>
</tr>
<tr>
<td></td>
<td>LM339</td>
<td>Comparetor</td>
<td>1</td>
<td>$1</td>
</tr>
<tr>
<td></td>
<td>Potentiometer</td>
<td>10K Potentionmeter</td>
<td>1</td>
<td>$3</td>
</tr>
<tr>
<td></td>
<td>DC Motor</td>
<td>Geared DC motor</td>
<td>4</td>
<td>$6</td>
</tr>
<tr>
<td></td>
<td>Tape sensor</td>
<td>Used for staplers</td>
<td>2</td>
<td>$2</td>
</tr>
<tr>
<td></td>
<td>Coin Sensor</td>
<td>Senses pennies</td>
<td>1</td>
<td>$2</td>
</tr>
<tr>
<td></td>
<td>Limit Switch</td>
<td>Limit switch with lever</td>
<td>4</td>
<td>$1.50</td>
</tr>
<tr>
<td></td>
<td>LED</td>
<td>Varying colors</td>
<td>50</td>
<td>$0.20</td>
</tr>
<tr>
<td></td>
<td>Button</td>
<td>Start button</td>
<td>1</td>
<td>$3</td>
</tr>
<tr>
<td></td>
<td>Servo</td>
<td>For boss/ergo</td>
<td>1</td>
<td>$15</td>
</tr>
<tr>
<td></td>
<td>Molex</td>
<td>Connectors of diff sizes</td>
<td>20</td>
<td>$0.30</td>
</tr>
<tr>
<td></td>
<td>Diodes</td>
<td>For preventing V spikes</td>
<td>10</td>
<td>$0.05</td>
</tr>
<tr>
<td></td>
<td>Resistors</td>
<td>Various values</td>
<td>100</td>
<td>$0.01</td>
</tr>
<tr>
<td></td>
<td>Capacitors</td>
<td>Various values</td>
<td>5</td>
<td>$0.10</td>
</tr>
<tr>
<td><strong>Mechanical</strong></td>
<td>Stapler</td>
<td>Swingline, red</td>
<td>2</td>
<td>$20</td>
</tr>
<tr>
<td></td>
<td>Acrylic</td>
<td>Multi colored</td>
<td>1</td>
<td>$20</td>
</tr>
<tr>
<td></td>
<td>Masonite</td>
<td>Brown</td>
<td>1</td>
<td>$15</td>
</tr>
<tr>
<td></td>
<td>Foam core</td>
<td>Red</td>
<td>1</td>
<td>$5</td>
</tr>
<tr>
<td></td>
<td>Fasteners</td>
<td>Various</td>
<td>30</td>
<td>$0.05</td>
</tr>
<tr>
<td></td>
<td>Foam wheels</td>
<td>Foamy</td>
<td>4</td>
<td>$2</td>
</tr>
<tr>
<td><strong>Decorative</strong></td>
<td>Inbox</td>
<td>from ikea</td>
<td>1</td>
<td>$4.50</td>
</tr>
<tr>
<td></td>
<td>Paper</td>
<td>for graphics</td>
<td>1</td>
<td>$10</td>
</tr>
<tr>
<td></td>
<td>Veneer</td>
<td>Wood, looks cheesey</td>
<td>1</td>
<td>$5</td>
</tr>
<tr>
<td></td>
<td>Mug</td>
<td>DFS</td>
<td>1</td>
<td>$3</td>
</tr>
<tr>
<td></td>
<td>Tape</td>
<td>Scotch</td>
<td>1</td>
<td>$3</td>
</tr>
<tr>
<td></td>
<td>Post-its</td>
<td>Multi-colored</td>
<td>1</td>
<td>$5</td>
</tr>
<tr>
<td></td>
<td>Business Cards</td>
<td>Customized</td>
<td>500</td>
<td>$0.01</td>
</tr>
<tr>
<td></td>
<td>Felt</td>
<td>Grey</td>
<td>1</td>
<td>$5</td>
</tr>
</tbody>
</table>

**TOTAL**   $214.00