Roshan Rajan, Katherine Yun, Penny Xu
ECE 110 Honors Lab Report
14 December 2015

Project: Triple C - Cute Chess Clock

Introduction:

Motivation

We realized that common game clocks often were external part of the game. In particular, a chess clock only tells the player the remaining time. Also, chess clocks' buttons are more prone to breaking and are not very accurate when a player hits it with a piece rather than making contact with the human hand. So, our team sought to make a cute chess clock that have feedback or function which involves the player and the game.

Solution Overview

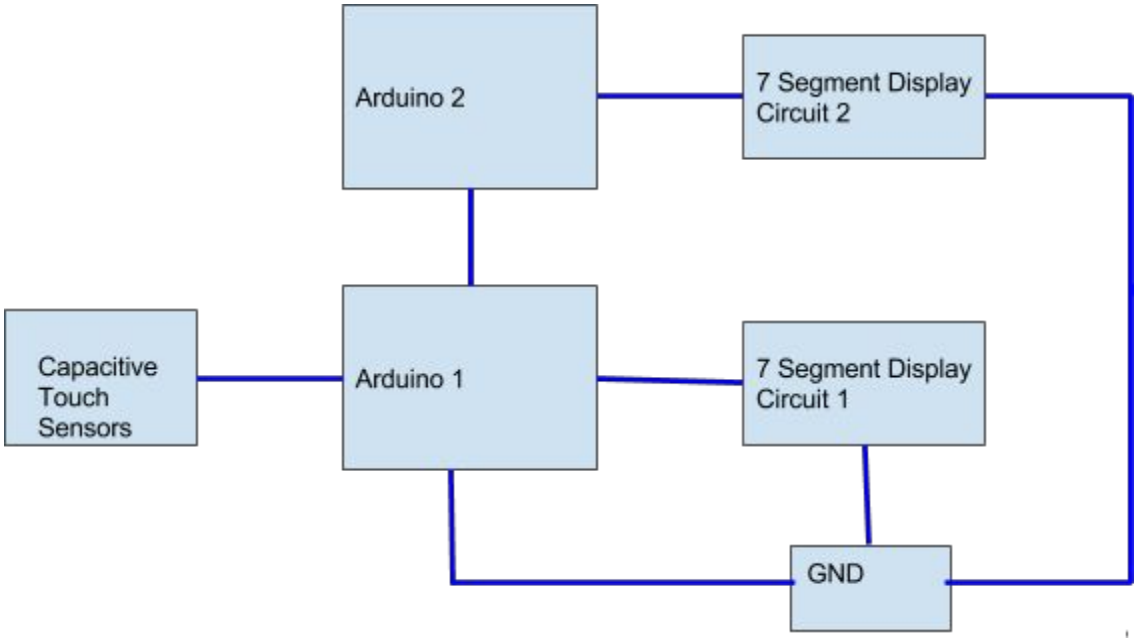
Our solution to the above problem is to make a chess clock that is integrated with sensors which provides useful information. We would use touch sensors as opposed to the traditional buttons to provide a more accurate detection. We will use Arduino to code the chess clock so that it will automatically store the change in time so that when the players go over the game, the player will now how long he or she spent on each move, which will help prepare the player on future games.

Design:

Design Overview

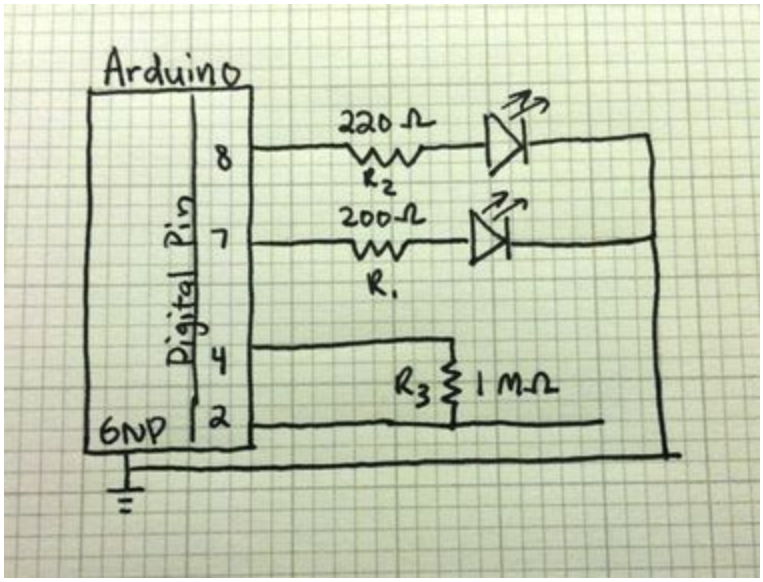
We designed our chess clock to be operated by capacitive touch sensors. This way, the clock will register only when a person's hand touches the plates connected to the extended wires. Using the digital send and receive pins, both player's wires will connect together to the receive pin which would toggle two states. This sensor will act as a switch that will turn on or pause the descending time displayed on the two 4-digit 7 segment display. We have two LEDs in circuit with an Arduino which will light up depending on the corresponding player's turn. We used three Arduinos to power and control the clock to reduce lag.

Block Diagrams

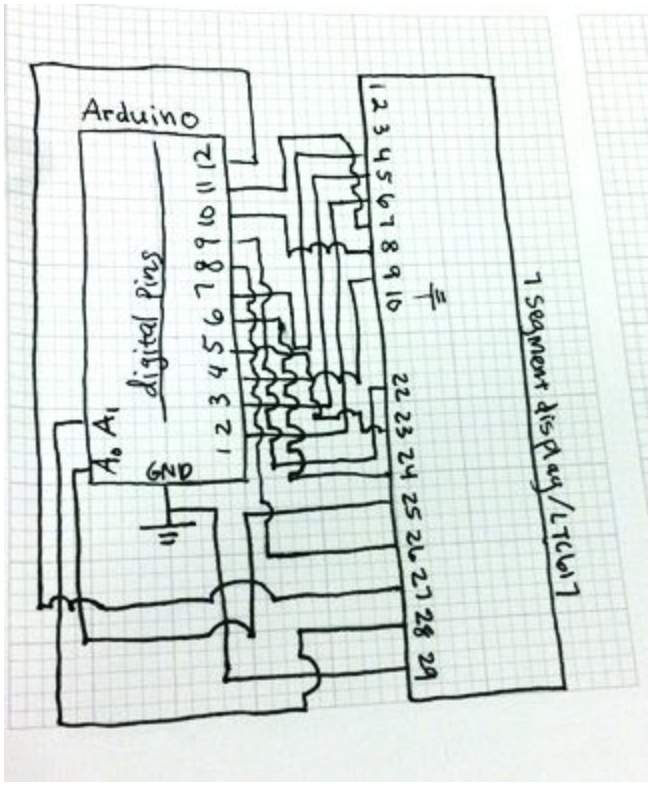


Schematics

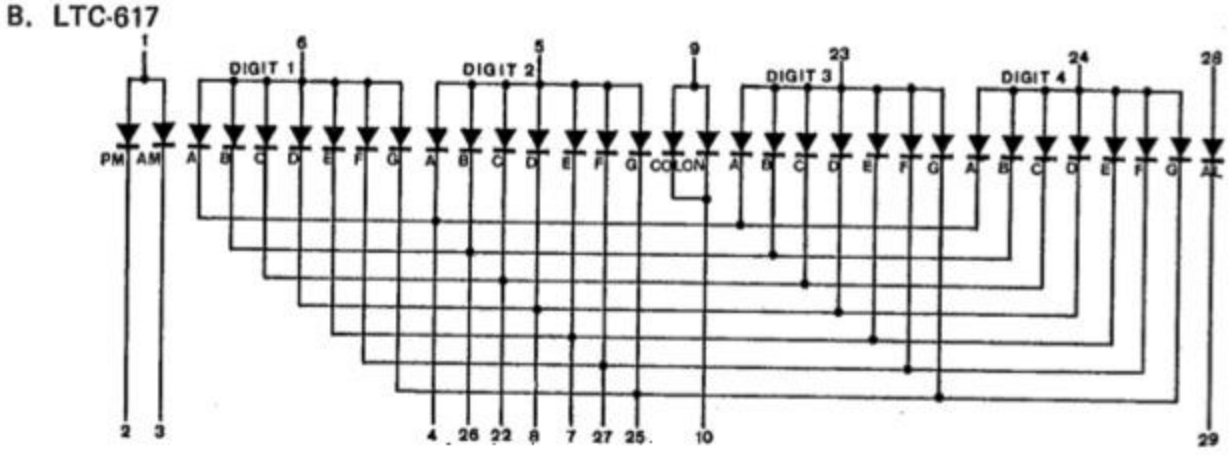
Capacitive Touch Sensor Circuit



Arduino to 7-Segment Display Circuit



7 Segment Display Schematics (Found in Data Sheet)



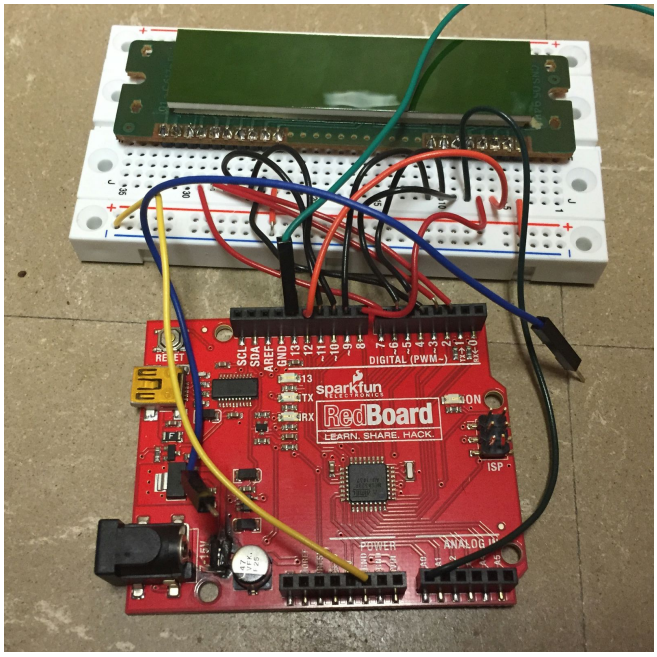
Results

Qualitative Analysis of results

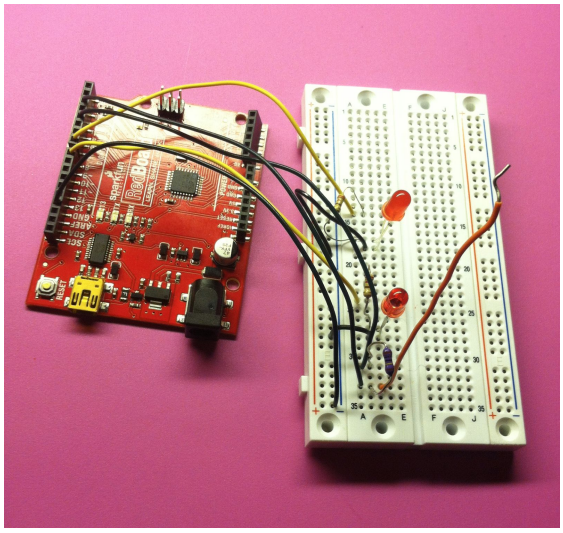
Our vision of how this project would work was supposed to be a clock that would time a chess game and would show the amount of time each move took to provide information to the player about how the player was doing and to evaluate the moves that were made. However, this vision proved to be extremely difficult. While we were able to get the clock to count down during each player's turn, we found it difficult to incorporate the capacitive touch sensor and allow the users to switch between players easily. This added effect also heightened the lag in the program causing the count down to not accurately count each second. Also the added functionality for evaluation was delayed to due to in ability for the clock to correctly count down. If we were to properly incorporate the RTC chip of the Arduino, then this lag could have been overcome and the functionality improved.

Quantitative Analysis of results

Delay of the program was our greatest difficulty. Whether it was adding more functionality to the program or correcting any of the existing code, there was a minute delay in how the Arduino utilized the program. As a result, much of our expected and wanted functionality was cut back in order to determine a way to reduce this loss. THe RTC module was supposed to fix this, however time and knowledge got in the way of properly implementing this. Given more time, this would definitely be the way to properly fix this issue. Our secondary issue was due to the multiple Arduinos where signals were affecting each other and not allowing for proper recognition of the capacitive touch sensor. While it is still unsure how to go about and fix this, implementing a system with less Arduino or creating a method where the signal interference would be minimized would be the next step.



Display Circuit



Touch Sensor Circuit

Future Work

Next steps for Project

There are many things we could have done to make the project function better. First of all, we could manage to simplify the design by using less Arduinos, breadboards and wires. For instance, using Arduino Nanos and bigger breadboards. That way, the design would be much easier to carry around and used during chess games. Secondly, we could also connect those jumper wires, which connect to the receive pin, to two separate pieces of foil and tape them on a flat surface. When chess players need to stop the clock, they can simply tap the foil on their side. Moreover, in the design of capacitive touch sensors, we could also add in a relay or transistor, both of which work as electrically operated switches, using a small voltage to switch on/off a high voltage or high current separately. Finally, to deal with the major problem of our design, we could try to improve the code we used for the Arduinos so that the time can be more accurate and able to record time between each switch of LEDs.

Conclusion

What Worked?

In our final project, the sub-circuit of capacitive touch sensor worked properly as we expected: when the wires connected to the receive pin were touched by hands, the two LEDs switched on/off and the two displays also started/paused.

What Didn't Work?

The major issue with our project was that the clock kept lagging and was inaccurate in counting down the time. We concluded that the sub-circuit of 7 segment displays was flawed by improper design or errors in circuit construction as well as delays due to the program cycle implementation. Also, the clock lacked the function of recording the time between player's moves.

What Did You Learn?

In this Honors lab, we were able to take our time and play with the hardware parts, which helped us have a better understanding of how capacitors, resistors, and sensors work. Even though it took longer to self-learn how we can put all the parts together, we were able to develop a big picture in terms of using and testing each parts. We also learned to figure out our mistakes through experimentation. Theoretically, the circuit and code that we constructed should have worked, but we learned that it is important to incorporate the differences and understand the connection between hardware and software. Technically, through this project, we learned how to construct a capacitive touch sensor and use a real time clock combined with 7 segment displays to count down the time. Also, we coded for the arduinos we used in this project and learned through trial and error.

Appendix:

Clock Component Code

```
#include <stdio.h>

#define DIGIT1 2
#define DIGIT2 3
#define DIGIT3 5
#define DIGIT4 6

#define SEGMENTA 7
#define SEGMENTB 8
#define SEGMENTC 9
#define SEGMENTD 10
#define SEGMENTE 11
#define SEGMENTF 12
#define SEGMENTG A0

#define COLON 4

#define BUTTON 13

#define ON  HIGH
#define OFF LOW

#define DELAYTIME 50

unsigned short minutes, seconds;
boolean pauseTime;
boolean keepPause;
int reading;
unsigned long lastTime; // keeps track of when the previous second happened

int buttonState;  // the current reading from the button pin
int lastButtonState = LOW;   // the previous reading from the button pin
unsigned long button_down_start = 0; // how long the button was held down
unsigned long lastDebounceTime = 0;  // the last time the output pin was toggled
unsigned long debounceDelay = 50;    // the debounce time

void setup() {
  // initialize all the required pins as output.
  pinMode(DIGIT1, OUTPUT);
  pinMode(DIGIT2, OUTPUT);
  pinMode(DIGIT3, OUTPUT);
  pinMode(DIGIT4, OUTPUT);

  pinMode(SEGMENTA, OUTPUT);
  pinMode(SEGMENTB, OUTPUT);
  pinMode(SEGMENTC, OUTPUT);
  pinMode(SEGMENTD, OUTPUT);
  pinMode(SEGMENTE, OUTPUT);
  pinMode(SEGMENTF, OUTPUT);
  pinMode(SEGMENTG, OUTPUT);

  pinMode(COLON, OUTPUT);
```

```
  // button is input
  pinMode(BUTTON, INPUT);

  // set the initial time
  minutes = 10;
  seconds = 0;
  pauseTime = false;
  keepPause = false;

  lastTime = millis();
  Serial.begin(9600);
}

void loop() {

  // Keep showing the display while waiting for timer to expire
  while (millis() - lastTime < 1000) {
    clock_show_time(minutes, seconds);

    digitalWrite(COLON, ON);

    // button presses increase minutes

    /*int max = 0;
    if (max < analogRead(BUTTON))
    {
      max = analogRead(BUTTON);
    }
     //Serial.print(analogRead(BUTTON));
     //Serial.print("    ");
     //Serial.println(max);
     // If the switch changed, due to noise or pressing:*/


    reading = digitalRead(BUTTON);
    Serial.println(reading);

     // If the switch changed, due to noise or pressing:
    if (reading != lastButtonState) {
      // reset the debouncing timer
      lastDebounceTime = millis();
    }

    if ((millis() - lastDebounceTime) > debounceDelay) {
      // whatever the reading is at, it's been there for longer
      // than the debounce delay, so take it as the actual current state:

      if (buttonState != reading) {
        button_down_start = millis(); // record the start of the current button state
      }

      buttonState = reading;

      // buttonState is now either on or off
      //if (buttonState == HIGH) {
        // if the button was held down more than 5 seconds, make it go faster
       // pauseTime = !pauseTime;
```

```
      //}

      // button has been pressed
     }

   lastButtonState = reading;
  }

  lastTime += 1000;

  incrementTime();
}

//
// a call to incrementTime decreases time by one second.
// also checks wheter Time should be paused or not
//
void incrementTime() {

  if(!pauseTime)
  {
  if (seconds == 0) {
    seconds = 59;
    if (minutes == 0) {
      minutes = 0;
      seconds = 0;
    }
    else {
      minutes--;
    }
  }
  else {
    seconds--;
  }
}
}

//
// resets the clock so the next game can be played
//
/*void resetTime(void){
  seconds = 0;
  minutes = 10;
}*/

//
// clock_show_time - displays the given time on the clock display
//   Note that instead of hr/min the user can also send min/sec
//   Maximum hr is 99, Maximum min is 59, and minimum is 0 for both (it's unsigned, heh).
//
void clock_show_time(unsigned short hours, unsigned short minutes) {
  unsigned short i;
  unsigned short delaytime;
  unsigned short num_leds[10] = { 6, 2, 5, 5, 4, 5, 6, 3, 7, 6 };
  unsigned short digit[4];
  unsigned short hide_leading_hours_digit;
```

```
  // convert minutes and seconds into the individual digits
  // check the boundaries
  if (hours > 99) hours = 99;
  if (minutes > 59) minutes = 59;

  // convert hr
  if (hours < 10 && hours > 0) {
    hide_leading_hours_digit = 1;
  }
  else {
    hide_leading_hours_digit = 0;
  }

  digit[0] = hours / 10;
  digit[1] = hours % 10; // remainder
  digit[2] = minutes / 10;
  digit[3] = minutes % 10; // remainder

  for (i = hide_leading_hours_digit; i < 4; i++) {
    clock_all_off();
    clock_show_digit(i, digit[i]);

    // fewer leds = brighter display, so delay depends on number of leds lit.
    delaytime = num_leds[digit[i]] * DELAYTIME;
    delayMicroseconds(delaytime);
  }

  clock_all_off();
}

//
// clock_all_off - turns off all the LEDs on the clock to give a blank display
//
void clock_all_off(void) {

  // digits must be ON for any LEDs to be on
  digitalWrite(DIGIT1, OFF);
  digitalWrite(DIGIT2, OFF);
  digitalWrite(DIGIT3, OFF);
  digitalWrite(DIGIT4, OFF);

  // segments must be OFF for any LEDs to be on
  digitalWrite(SEGMENTA, ON);
  digitalWrite(SEGMENTB, ON);
  digitalWrite(SEGMENTC, ON);
  digitalWrite(SEGMENTD, ON);
  digitalWrite(SEGMENTE, ON);
  digitalWrite(SEGMENTF, ON);
  digitalWrite(SEGMENTG, ON);

  // turn off colon and alarm too
  digitalWrite(COLON, OFF);
}

//
// clock_show_digit - turns on the LEDs for the digit in the given position
//      position can be from 0 through 3: 0 and 1 being the minutes, 2 and 3 being the seconds
```

```
//      value can be from 0 through 9, ie, a valid single digit.
//
//      (if value is out of range, it displays a 9. if digit is out of range display remains blank)
//
void clock_show_digit(unsigned short position, unsigned short value) {
  byte a;
  byte b;
  byte c;
  byte d;
  byte e;
  byte f;
  byte g;

  switch (position) {
    case 0:
      digitalWrite(DIGIT1, ON);
      break;
    case 1:
      digitalWrite(DIGIT2, ON);
      break;
    case 2:
      digitalWrite(DIGIT3, ON);
      break;
    case 3:
      digitalWrite(DIGIT4, ON);
      break;
  }

  a = !(value == 1 || value == 4);
  b = !(value == 5 || value == 6);
  c = !(value == 2);
  d = !(value == 1 || value == 4 || value == 7);
  e =  (value == 0 || value == 2 || value == 6 || value == 8);
  f = !(value == 1 || value == 2 || value == 3 || value == 7);
  g = !(value == 0 || value == 1 || value == 7);

  if (a) digitalWrite(SEGMENTA, OFF);
  if (b) digitalWrite(SEGMENTB, OFF);
  if (c) digitalWrite(SEGMENTC, OFF);
  if (d) digitalWrite(SEGMENTD, OFF);
  if (e) digitalWrite(SEGMENTE, OFF);
  if (f) digitalWrite(SEGMENTF, OFF);
  if (g) digitalWrite(SEGMENTG, OFF);
}
```

Capacitive Touch Sensor Code

```
#include <CapacitiveSensor.h>

int led = A0; //change '42' to any desired pin...
int led2 = A5;
long time = 0;
int state = HIGH;
int state2 = LOW;

boolean yes;
boolean previous = false;

int debounce = 200;

CapacitiveSensor   cs_4_2 = CapacitiveSensor(4,2);

void setup()
{
  cs_4_2.set_CS_AutocaL_Millis(0xFFFFFFFF);  //Calibrate the sensor...
  pinMode(led, OUTPUT);

}
void loop()
{
   long total1 =  cs_4_2.capacitiveSensor(30);

   if (total1 > 60){yes = true;}
   else {yes = false;}

   // to toggle the state of state
   if(yes == true && previous  == false && millis() - time>debounce){

     if(state == LOW){
       state = HIGH;
     }
     else
       state = LOW;
       time = millis();
  }


    if (state == HIGH) {
      state2 = LOW;
    }

    else {
      state2 = HIGH;
    }

    digitalWrite(led, state);

    digitalWrite(led2, state2);
     previous = yes;

     delay(200);
}
```