”Twisted” SIMON

Arduino Ignition Grant

Zaza Soriano
Brian Taylor

May 5, 2014

Contents

Listings

1 Goals

To learn about various sensors and how they interact with the Arduino by making a game based off of the classic game of SIMON.

2 References

http://www.instructables.com/id/Arduino-Simon-Says
http://www.instructables.com/id/Stickytape-Sensors/step5/Velostat

3 Hardware

- Arduino Board
- USB Cable
- Breadboard
- Wires
- 5 x Resistor 220Ω
- 4 x LED
- 4 x Button
- Speaker
- Rotary Knob (Potentiometer)
- Light Sensor (Photocell)
- Motion Sensor (Tilt Switch)
- Temperature Sensor
- Conductive Fabric
- Conductive Thread
- Velostat
- Tape
- Scissors

4 Discussion

4.1 SIMON

”Think fast... SIMON, ’Chase my flashing lights and sounds’!”

The classic SIMON is a computer-controlled game that consists of a base unit with 4 color

lenses and a control panel. The challenge is to repeat the ever-increasing random signals that SIMON generates. If the pattern you respond is correct SIMON will add another light to the pattern.

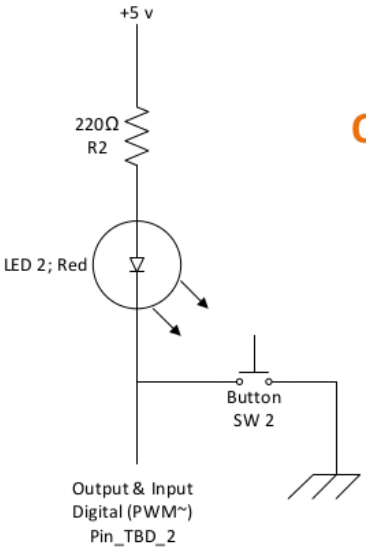
”Twisted” Simon is a game that is controlled by an Arduino. It will have 4 LEDs each with it’s own way to activate it. The LEDs could be activated by the press of a button or a flashlight, just to name a few examples (as the options are endless). The Arduino will start by lighting up a random LED, at witch point the Player will activate the appropriate LED in response (hint: the correct LED is the one the Arduino lit up). If correct, the Arduino will add a random LED to the sequence (which is now 2 long) and the process repeats itself. If the Player enters the wrong sequence, the game is lost and the Player must start over.

4.2 Wiring the Arduino

Each LED can be controlled by either a digital input (Figure 4.1a) or an analog input (Figure 4.1b). When wiring the sensors, make sure they match the GPIO pins used in the code. Figure 4.2 shows two examples of analog sensors that can be used for this lab.

Figure 4.1: Sensor Schematics

(a) Digital sensor Schematic



(b) Analog Sensor Schematic

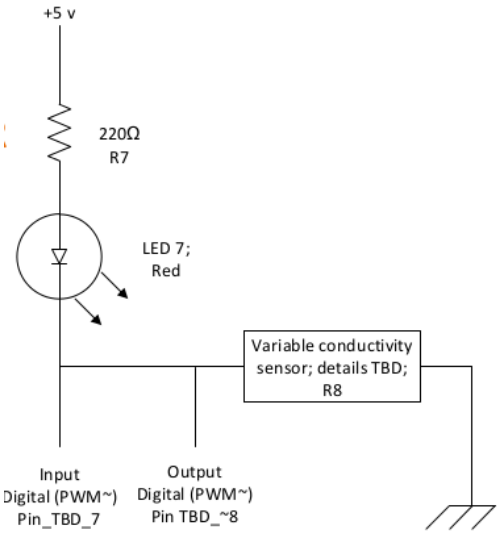
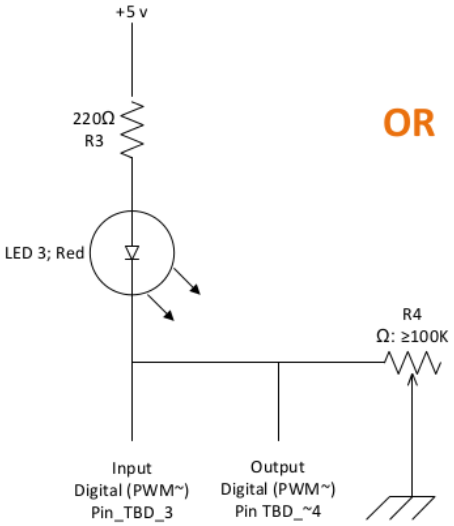


Figure 4.3 shows the entire schematic needed to wire ”Twisted” SIMON to the Arduino. R5 can be substituted to get higher or lower volumes from the Piezo Buzzer. R12, R22, R32, R42 are each the various sensors that can act as the switch. **Make sure that the inputs/outputs match the values in the code!**

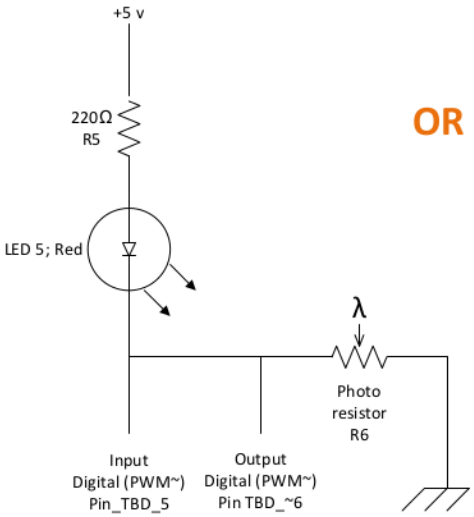
Figure 4.2: Analog Sensor Example Schematics

(a) Potentiometer Schematic



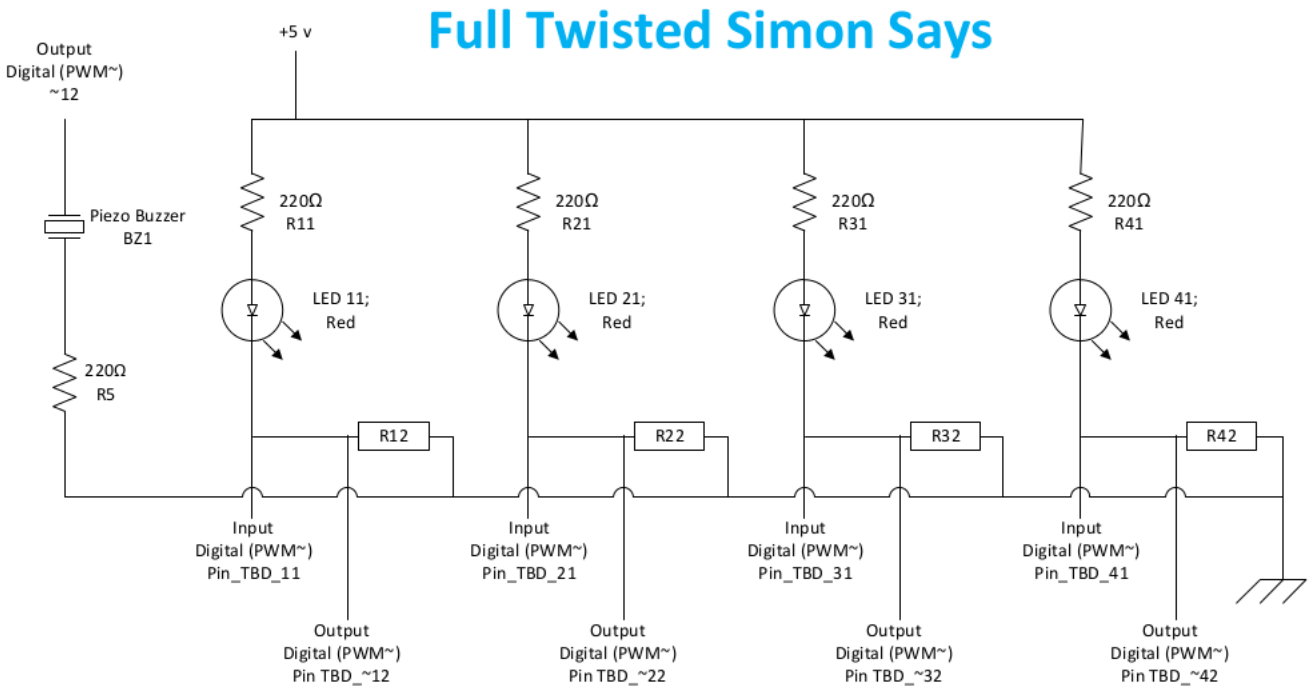
OR

(b) Photo Resistor Schematic



OR

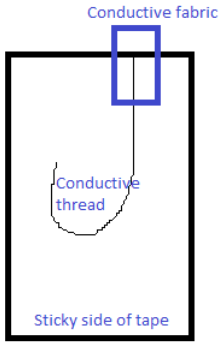
Figure 4.3: ”Twisted” SIMON Schematics



4.3 Making a Push Sensor

1. Gather your materials

Figure 4.4: Push sensor insides



(a) 3 pieces of velostat (same size)

(b) 2 pieces of tape (slightly larger than the velostat)

(c) 2 strands of conductive thread (about the length of the velostat)

(d) 2 pieces of conductive fabric (enough for the alligator clip to work with)

2. Place a conductive thread and fabric onto the sticky side of a piece of tape, show in Figure 4.4
3. Repeat previous step
4. Sandwich velostat between tape making sure to overlap the conductive thread

5 Procedure

5.1 ”Not so twisted” SIMON

1. Wire the Arduino with 4 LED’s, 4 push buttons, and 1 speaker (See Section 4.2)
2. Load the Twisted_Simon_Says.ino file onto the Arduino (See Appendix A.1)

5.2 ”Twisted” SIMON

1. Replace one of the push buttons with an analog sensor (See Section 4.2)
2. Change the appropriate sensor information in the code, as an example look at line 21 in Listing 5.1.

Listing 5.1: Twisted SIMON Sensors

```
/*
 * IO: If all inputs are digital then this is sufficient. The format is
 * { inputPort, outputPort, activeTone, threshold }. The output pins should be digital,
 * the input pins can be analog or digital. Analog pins should be written as A#.
 * The activeTone values can be found in the pitches.h file. threshold should be the value
 * you want the sensor to be below to activate the LED. It will be 0 for digital inputs.
 */
int P1[] = {
  2, 2, NOTE_C4, 0};
int P2[] = {
  3, 3, NOTE_G3, 0};
int P3[] = {
  4, 4, NOTE_GS3, 0};
int P4[] = {
  A0, 5, NOTE_B3, 400};
```

To get correct threshold value, uncomment the debug lines (shown in Listing 5.2), run the program, and use the serial monitor to pick an appropriate threshold for the analog sensor.

Listing 5.2: Twisted SIMON Debug

```
int readPort(int n) {
  if(!IOports[n][3])
    return !digitalRead(IOports[n][0]);
  else {
    //Serial.println(analogRead(IOports[n][0])); //Uncomment this line to test analog port
    return (analogRead(IOports[n][0]) < IOports[n][3]);
  }
}
*
void setup() {
  // put your setup code here, to run once:
  //Serial.begin(9600); //Uncomment this line to test analog port
  randomSeed(analogRead(5));
}
```

3. Load the modified Twisted_Simon_Says.ino file onto the Arduino

5.3 Future SIMON

What kinds of twists can you think of to make this came more complex? Some examples to get your mind rolling are:

- Make the pattern new each time it increases
- Set a time limit for the play to repeat the pattern
- Add more LEDs
- Add multiple sensors per LED

A
Arduino Code

Listing A.1: Twisted SIMON Code

```
#include "Twisted_Simon_pitches.h"

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define MAX_PATTERN 20
#define INITIAL_DELAY 1000

/*
 * IO: If all inputs are digital then this is sufficient. The format is
 * { inputPort, outputPort, activeTone, threshold }. The output pins should be digital,
 * the input pins can be analog or digital. Analog pins should be written as A#.
 * The activeTone values can be found in the pitches.h file. threshold should be the value
 * you want the sensor to be below to activate the LED. It will be 0 for digital inputs.
 */
int P1[] = {
  2, 2, NOTE_C4, 0};
int P2[] = {
  3, 3, NOTE_G3, 0};
int P3[] = {
  4, 4, NOTE_GS3, 0};
int P4[] = {
  5, 5, NOTE_B3, 0};

int* IOports[] = {
  P1, P2, P3, P4};

// port that outputs tone. Must be PWM capable
int tonePort = 6;

/*
 * PROGRAM STATE
 * These globals variables remember where we are in a pattern and if
 * we are reading or writing the pattern.
 */
enum {
  INIT_PATTERN,
  SHOW_PATTERN,
  READ_PATTERN
};

int state = INIT_PATTERN;
int pattern[MAX_PATTERN];
int patternLength = 0;
int patternIndex = 0;
int patternDelay = INITIAL_DELAY;

/*
 * Read from a virtual port.
 */
int readPort(int n) {
  if(!IOports[n][3])
    return !digitalRead(IOports[n][0]);
  else {
    //Serial.println(analogRead(IOports[n][0])); //Uncomment this line to test analog port
    return (analogRead(IOports[n][0]) < IOports[n][3]);
  }
}

/*
 * Output the tone associated with a virtual port
 */
```

```
61 void writeTone(int n) {
     tone(tonePort, IOports[n][2]);
63 }

65 /*
    * Enable=1, Disable=0 a virtual port. Also outputs
67  * the tone associated with that port
    */
69 void writePort(int n, int v) {
     /* The reference design will short to ground if we ever write 1 to
71    * disable the light while the switch is depressed. Instead we put
      * the pin in high impedance (input) if OFF is requested
73    */
     if(v) {
75     pinMode(IOports[n][1], OUTPUT);
       digitalWrite(IOports[n][1], 0);
77     writeTone(n);
     }
79   else {
       pinMode(IOports[n][1], INPUT);
81   }
   }
84
   /*
85  * Read the unique input available from any of the virtual ports.
    * Returns -1 if no input or if the input is not unique.
87  */
   int uniqueRead() {
89   int didRead = -1;
     int isUnique = 0;
91   for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
       if(readPort(i)) {
93       if(didRead == -1) {
           didRead = i;
95         isUnique = 1;
         }
97       else {
           isUnique = 0;
99       }
       }
101  }
     if(didRead != -1 && isUnique) {
103    return didRead;
     }
105  else {
       return -1;
107  }
   }
110
   /*
111 * Enable only the given virtual port. All others are disabled.
    */
113 void uniqueWrite(int n) {
     for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
115    writePort(i, i == n);
     }
117 }

119 /*
    * Disable all virtual ports and stop the tone.
121 */
   void clearOutput() {
123  uniqueWrite(-1);
     noTone(tonePort);
125 }

127 /*
    * Prepare all virtual ports for writing.
```

```
 */
void prepareWrite() {
  for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
    pinMode(IOports[i][0], INPUT);
    pinMode(IOports[i][1], OUTPUT);
  }
}

/*
 * Prepare all virtual ports for reading.
 */
void prepareRead() {
  for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
    pinMode(IOports[i][0], INPUT);
    pinMode(IOports[i][1], INPUT);
  }
}

void setup() {
  // put your setup code here, to run once:
  //Serial.begin(9600); //Uncomment this line to test analog port
  randomSeed(analogRead(5));
}

/*
void debugPrint() {
 Serial.print(uniqueRead());
 Serial.print(" ");

 for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
 Serial.print(readPort(i));
 Serial.print(" ");
 }

 Serial.println(state);
 }
 */

void loop() {
  //debugPrint();
  if(state == INIT_PATTERN) {
    // create the next pattern. advance the length or the speed to
    // make this pattern more challenging than the previous pattern
    if(patternLength > MAX_PATTERN) {
      patternLength = 1;
      patternDelay = patternDelay/2;
    }

    pattern[patternLength] = random(ARRAY_SIZE(IOports));

    patternLength = patternLength + 1;
    patternIndex = 0;
    prepareWrite();
    clearOutput();
    state = SHOW_PATTERN;
    delay(1000);
  }
  else if(state == SHOW_PATTERN) {
    // display the currently active pattern. If we've displayed it
    // completely then advance to the read state
    if(patternIndex >= patternLength) {
      // clear display and audio
      clearOutput();

      patternIndex = 0;
      state = READ_PATTERN;
    }
    else {
```

```
197         uniqueWrite(pattern[patternIndex]);
            delay(patternDelay);
200
            clearOutput();
201         delay(patternDelay / 2);
            patternIndex++;
203       }
        }
205     else if(state == READ_PATTERN) {
          // read the currently active pattern. when the input matches the
207       // current token in the pattern we display it and then advance
          // to the next token. When complete, move to INIT_PATTERN state
209       // if fail reset...
          if(patternIndex == patternLength) {
211         state = INIT_PATTERN;
            clearOutput();
213       }
          else {
215         prepareRead();
            int unique = uniqueRead();
217         if(unique == -1) {
              clearOutput();
219         }
            else {
221           prepareWrite();
              uniqueWrite(unique);
223           if(unique == pattern[patternIndex]) {
                delay(patternDelay);
225             patternIndex++;
              }
227           else{ //wrong LED!
                writeTone(4);
230
                for(int n = 0; n < ARRAY_SIZE(IOports); n++){
231               pinMode(IOports[n][1], OUTPUT);
                  digitalWrite(IOports[n][1], 0);
233             }

235             delay(patternDelay);
                state = INIT_PATTERN;
237             patternDelay = INITIAL_DELAY;
                patternLength = 0;
239           }
            }
241       }
        }
243     else {
          // reset
245       patternLength = 0;
          patternDelay = INITIAL_DELAY;
247       state = INIT_PATTERN;
        }
249 }
```