Physical Computing: Lab 2

By: Daniel Speiser, Yang Hu, Clément Bellec

September 30, 2015

Contents

1 Summary

The objective of this lab was to build a Morse Code transmitter that can read Morse input from a push button as dots and dashes, and display the corresponding digit on an LED matrix. Fig. 1 below gives the correspondence between digits and Morse code.

Figure 1: Morse Code for Digits

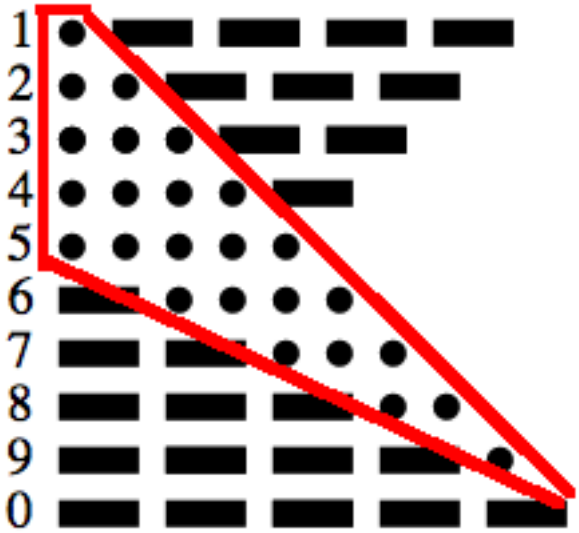

Figure 2: Cracking Morse Code

2 System-Level Block Diagram

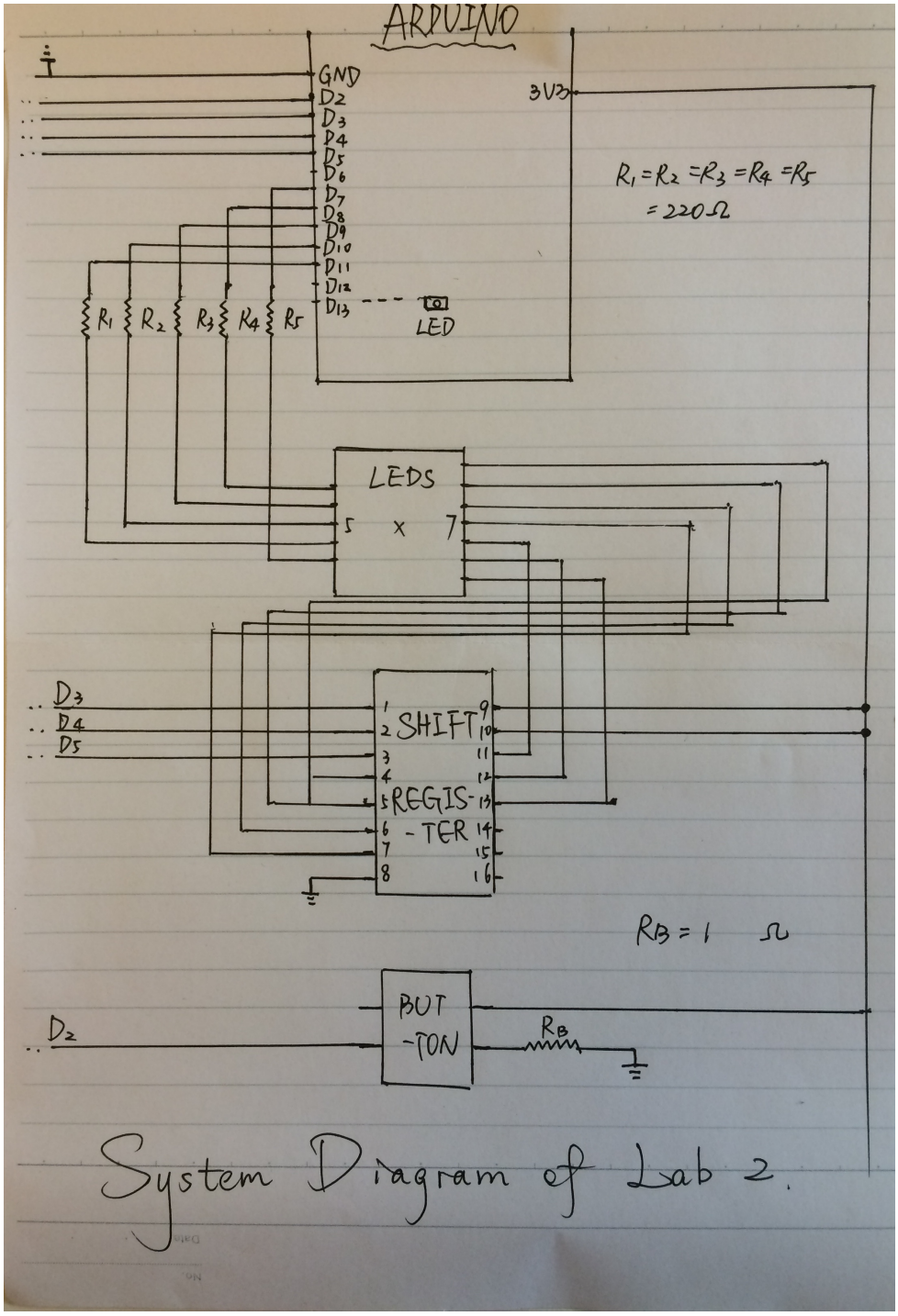

Figure 3: System Level Diagram

3 Software Description

3.1 Summary

Our software (as seen below) is designed to read voltage changes on the push button, separate dots (signal $<$ 300 ms) from dashes (signal $\geq$ 300ms), and check the corresponding digit (or identify an invalid entry). Once this is verified, it will turn on and off the LEDs on an LED matrix to display the 'translated' number. Due to the limited number of output on the Arduino FiO, we output directly from the board to the 5 columns of the matrix, while the output for the 7 rows is transferred through a Serial-In/Parallel-Out shift register.

3.2 Debouncing

We had to debounce the button to avoid mistaking noise for signal (analog input). We did this by separating two variables: "reading" and "buttonState", and using a press timestamp. The debouncing algorithm constantly pulls the voltage from the push button. Upon change in that voltage, it stores the new value in the "reading" variable, and starts a chronometer. If the voltage doesn't change during the DebounceDelay time (which we set to 50 ms) then the "reading" variable is passed to "buttonState" and becomes the new recorded state of the button.
For the purposes of this Lab, we used a polling method since our system wouldn't be powered on constantly. If it needed to remain on and be energy efficient, an interrupt method would be preferred.

3.3 Decoding Inputs

In order to decode inputs, we first had to 'crack' the Morse Code pattern (as can be found in fig. 2). A number between 1 and 5 begins with a dot, and a number between 6 and 9, or 0, begins with a dash. In order to decode the input, we start by determining whether the first signal is a dot or a dash. If the first signal is a dot, we set a count to 1, (since the digit must be between a 1 and a 5), otherwise, we set the count to 6 (again, since it must be between 6 and 9, or 0). We then traverse through the rest of the signals: if the following signal is the same as the first, we increment the count, otherwise we set a flag which informs us that the signal type has changed (i.e. from dot to dash, or dash to dot). We continue to traverse through the signals: if the signal changes again, we know that it is erroneous input and return -1, since Morse Digit signals (dots and dashes) do not change between each other more than once. Otherwise (if the signals remain the same), we return the count modulus 10 (to handle the case of 0) as our decoded digit.

```
/*
 Debounce
 Each time the input pin goes from LOW to HIGH (e.g. because of a push-button
 press), the output pin is toggled from LOW to HIGH or HIGH to LOW. There's
 a minimum delay between toggles to debounce the circuit (i.e. to ignore
 noise).
 Some variables are longs because the time, measured in miliseconds,
 will quickly become a bigger number than can't be stored in an int.
```

```
 */
// constants won't change. They're set here.
const int buttonPin = 2; // the number of the pushbutton pin
const int latchPin = 3; // the number of the pin commanding the latch on the shift register
const int dataPin = 4; // the number of the pin commanding the data (serial in) on the shift registe
const int clockPin = 5; // the number of the pin commanding the clock on the shift register
const int ledPin = 13; // the number of the LED pin
const int NUM_COLS = 5; // the number of columns in the LED matrix
const int COLS[] = {7, 8, 9 ,10, 11}; // array to run through the columns
// 0b11111110, 0b10000010, 0b10000010, 0b10000010, 0b11111110
const unsigned char ZERO[] = {0xFE, 0x82, 0x82, 0x82, 0xFE};
// 0b00000000, 0b00100010, 0b01111110, 0b10001010, 0b00000000
const unsigned char ONE[] = {0x0, 0x42, 0xFE, 0x2, 0x00};
// 0b00000000, 0b00100010, 0b01111110, 0b10001010, 0b00000000
const unsigned char TWO[] = {0x9E, 0x92, 0x92, 0x92, 0xF2};
// 0b10011110, 0b10010010, 0b10010010, 0b10010010, 0b11110010
const unsigned char THREE[] = {0x92, 0x92, 0x92, 0x92, 0xFE};
// 0b10010010, 0b10010010, 0b10010010, 0b10010010, 0b11111110
const unsigned char FOUR[] = {0xF0, 0x10, 0x10, 0xFE, 0x10};
// 0b11110000, 0b00010000, 0b00010000, 0b11111110, 0b00010000
const unsigned char FIVE[] = {0xF2, 0x92, 0x92, 0x92, 0x9E};
// 0b11110010, 0b10010010, 0b10010010, 0b10010010, 0b10011110
const unsigned char SIX[] = {0xFE, 0x92, 0x92, 0x92, 0x9E};
// 0b11111110, 0b10010010, 0b10010010, 0b10010010, 0b10011110
const unsigned char SEVEN[] = {0xE0, 0x80, 0x80, 0x80, 0xFE};
// 0b11100000, 0b10000000, 0b10000000, 0b10000000, 0b11111110
const unsigned char EIGHT[] = {0xFE, 0x92, 0x92, 0x92, 0xFE};
// 0b11111110, 0b10010010, 0b10010010, 0b10010010, 0b11111110
const unsigned char NINE[] = {0xF0, 0x90, 0x90, 0x90, 0xFE};
// 0b01100000, 0b10000000, 0b10011010, 0b10010000, 0b01100000
const unsigned char ERROR_MATRIX[] = {0x60, 0x80, 0x9A, 0x90, 0x60};
// the rows of the matrix will be commanded through the shift register
const unsigned char* DIGIT_MATRICES[] = { ZERO,
                                          ONE,
                                          TWO,
                                          THREE,
                                          FOUR,
                                          FIVE,
                                          SIX,
                                          SEVEN,
                                          EIGHT,
                                          NINE
                                        };
const int NUM_SIGNALS = 5; //num signals needed by button before it counts it as full input
// Defines the duration of a DOT. If the button is pressed for longer, assumes a DASH
const long DOT_DURATION = 300;

// variables that will change:
int buttonState = 0;                 // the current reading from the input pin
int lastButtonState = LOW;           // for debouncing
long lastDebounceTime = 0;           // the last time the output pin was toggled
long debounceDelay = 50;             // the debounce time; increase if the output flickers
long lastTime = 0;                   // Last recorded timestamp (used for mesuring duration)
boolean buttonWasPressed = false;    // Indicator of whether button was pressed in the last cycle
```

```
byte inputSignals[NUM_SIGNALS];      // Input signal buffer
int inputSignalIndex = 0;            // Index into the input signal buffer

void resetInputSignals() {           // Reset the input signal buffer and index
  memset(inputSignals, 0, NUM_SIGNALS); // reset each element in array to 0
  inputSignalIndex = 0;
}

int translateInput(byte input[]) {
  int num; bool flag = 0;             // create num and flag variabls
  if (input[0] == 0)                 // if first byte is 0 - i.e. a dot:
    num = 1;                          // num must be between 1 and 5. start at 1
  else                                // otherwise:
    num = 6;                          // num must be between 6 and 9, or 0. start at 6
  // loop through the rest of the morse signals to determine the num
  for (int i = 1; i < NUM_SIGNALS; i++) {
    if (input[i] == input[0] && flag) // handles erroneous morse input
        return -1;                    // returns illegal morse code input
    else if (input[i] == input[0])    // if the first byte is equal to the current signal:
      num++;                          // increment the num
    else                              // otherwise:
      flag = 1; // a change of signal type (dot to dash, or dash to dot) occured, set flag
  }
  return num % 10;                    // returns num modulus 10 to handle the case of '0'
}

void displayOnLEDMatrix(int num) {
  const unsigned char* LED_MATRIX;    // matrix we will use to light LEDs
  if (num == -1)                      // error handling
    LED_MATRIX = ERROR_MATRIX;        // set matrix to error (display '?')
  else                                // otherwise
    LED_MATRIX = DIGIT_MATRICES[num]; // set matrix to correct num
  // loop through each column 1000 times, to display the digit for ~ 2 seconds
  // using a timer causes led flickering, since the calculation of millis() causes a slight delay
  for (int num_loops = 0 ; num_loops < 1000; num_loops++) {
    for(int i = 0; i < NUM_COLS; i++) { // loop through each col
      digitalWrite(COLS[i],LOW);      // set the current col to low, i.e. ON
      digitalWrite(latchPin, LOW);    // set the latch pin to low
      // shiftOut the current col in order to determine which rows to set HIGH or LOW
      shiftOut(dataPin, clockPin, LSBFIRST, LED_MATRIX[i]);
      digitalWrite(COLS[i],HIGH);     // turn rows back off
      digitalWrite(latchPin, HIGH);   // reset latch pin
    } // col loop
  } // display x times loop
}

void setup() {
  pinMode(buttonPin, INPUT);          // set button as input pin
  pinMode(latchPin, OUTPUT);          // set latch as output pin
  pinMode(clockPin, OUTPUT);          // set clock as output pin
  pinMode(dataPin, OUTPUT);           // set data as output pin
  pinMode(ledPin, OUTPUT);            // set led as output pin
  for(int i = 0; i < NUM_COLS; i++) { // loop through each col
    pinMode(COLS[i], OUTPUT);         // set col pin as output
```

```
    digitalWrite(COLS[i], HIGH);      // set col to HIGH, i.e. off
  }
  Serial.begin(9600);                 // init the serial port
  resetInputSignals();                // reset input signal buffer
}

void loop() {
  // read the state of the switch into a local variable:
  int reading = digitalRead(buttonPin); // get the current button state
  long currTime = millis();           // get the current timestamp
  long duration = currTime - lastTime; // get elapsed time
  // check to see if you just pressed the button
  // (i.e. the input went from LOW to HIGH), and you've waited
  // long enough since the last press to ignore any noise:
  if (reading != lastButtonState)     // if the switch changed, due to noise or pressing:
    lastDebounceTime = millis();      // reset the debouncing timer
  // whatever the reading is at, it's been there for longer
  // than the debounce delay, so take it as the actual current state:
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState)       // if the button state has changed:
      buttonState = reading;          // save the new button state
  }
  digitalWrite(ledPin, buttonState);  // turn on the built in led if the button was pressede
  // save the reading. next time through the loop, it'll be the lastButtonState
  lastButtonState = reading;

  if (buttonState == HIGH) {          // if the button is pressed (DEBOUNCED)
    if (!buttonWasPressed) {          // and the button was not pressed before
      buttonWasPressed = true;        // remember the button press
      lastTime = currTime;            // record the time of the button press
    }                                 // end of if (button was not pressed)
  } else {                            // the button is not pressed
    if (buttonWasPressed) {           // the button was just released
      if (duration < DOT_DURATION)    // Check if duration exceeds DOT cutoff
        inputSignals[inputSignalIndex] = 0;// remember the DOT
      else                            // otherwise
        inputSignals[inputSignalIndex] = 1;// remember the DASH
    inputSignalIndex++;               // advances to the next signal buffer
    buttonWasPressed = false;         // consume previous button press
    lastTime = currTime;              // record the time the button was released
    } else {                          // the button was not just released
      if (inputSignalIndex == NUM_SIGNALS) {   // if the buffer is full
        int currNum = translateInput(inputSignals); // get num from morse code
        displayOnLEDMatrix(currNum);  // call function to display digit on led matrix
        resetInputSignals();          // reset the input signal buffer
      }
    }
  }
}
```

4 References

- Button Debouncing: https://www.arduino.cc/en/Tutorial/Button
- Translating Button Input:
  https://newcircle.com/s/post/397/morse_code_transcoder_for_arduino
- Shift Out: https://www.arduino.cc/en/Reference/ShiftOut
- Morse Code: https://en.wikipedia.org/wiki/Morse_code