TRS-80 Model 100

2021-11-27

Overview

TODO

Figure 1: TRS-80 Model 100

TODO

Screen

LCD

The display has a 1920×480 IPS panel. I think I read somewhere that these were designed as rear-view mirrors. At 218mm, it's actually a little wider than the original LCD.

Bracket

The LCD is mounted to a custom 3D printed bracket that's attached to the enclosure using the original screw bosses. The LCD fits snugly between bracket and the front plastic, with three pieces of double sided tape between the display and the bracket for good measure.

Configuration

```
hdmi_force_hotplug=1
hdmi_group=2
hdmi_mode=87
hdmi_force_mode=1
hdmi_timings=480 0 30 30 30 1920 0 6 6 6 0 0 0 60 0 66000000 7
max_framebuffer_width=480
max_framebuffer_height=1920
display_hdmi_rotate=1
```

Keyboard

Matrix

The TRS-80 keyboard matrix is 8 rows by 9 columns. The keyboard connects to the main board using two 10-pin connectors.

The basic process for scanning the keyboard matrix is:

```
INITIALIZE all columns as OUTPUT and set HIGH
INITIALIZE all rows as INPUT with internal pull-up resistors

FOR each column FROM 0 TO num_cols - 1 DO
    SET the current column TO LOW

    FOR each row FROM 0 TO num_rows - 1 DO
        IF the value at the current row IS LOW THEN
            key at the current position (row, column) is being pressed
        END IF
    ENDFOR

    SET the current column TO HIGH
ENDFOR
```

Firmware v1

For this first experiment, I connected the rows and columns directly to the digital I/O pins on an Arduino Pro Micro. The firmware scans the rows and columns every 5ms, and uses the `Keyboard` library to send key presses and releases to the host.

Wiring

Keyboard Pin	Description	Arduino Pro Micro Pin
1	Col 0	0
2	Col 1	1
3	Col 2	2

Keyboard Pin	Description	Arduino Pro Micro Pin
4	Col 3	3
5	Col 4	4
6	Col 5	5
7	Col 6	6
8	Col 7	7
9	Col 8	8
12	Row 0	21
13	Row 1	20
14	Row 2	19
15	Row 3	18
16	Row 4	15
17	Row 5	14
18	Row 6	16
19	Row 7	10
20	GND	GND

Code

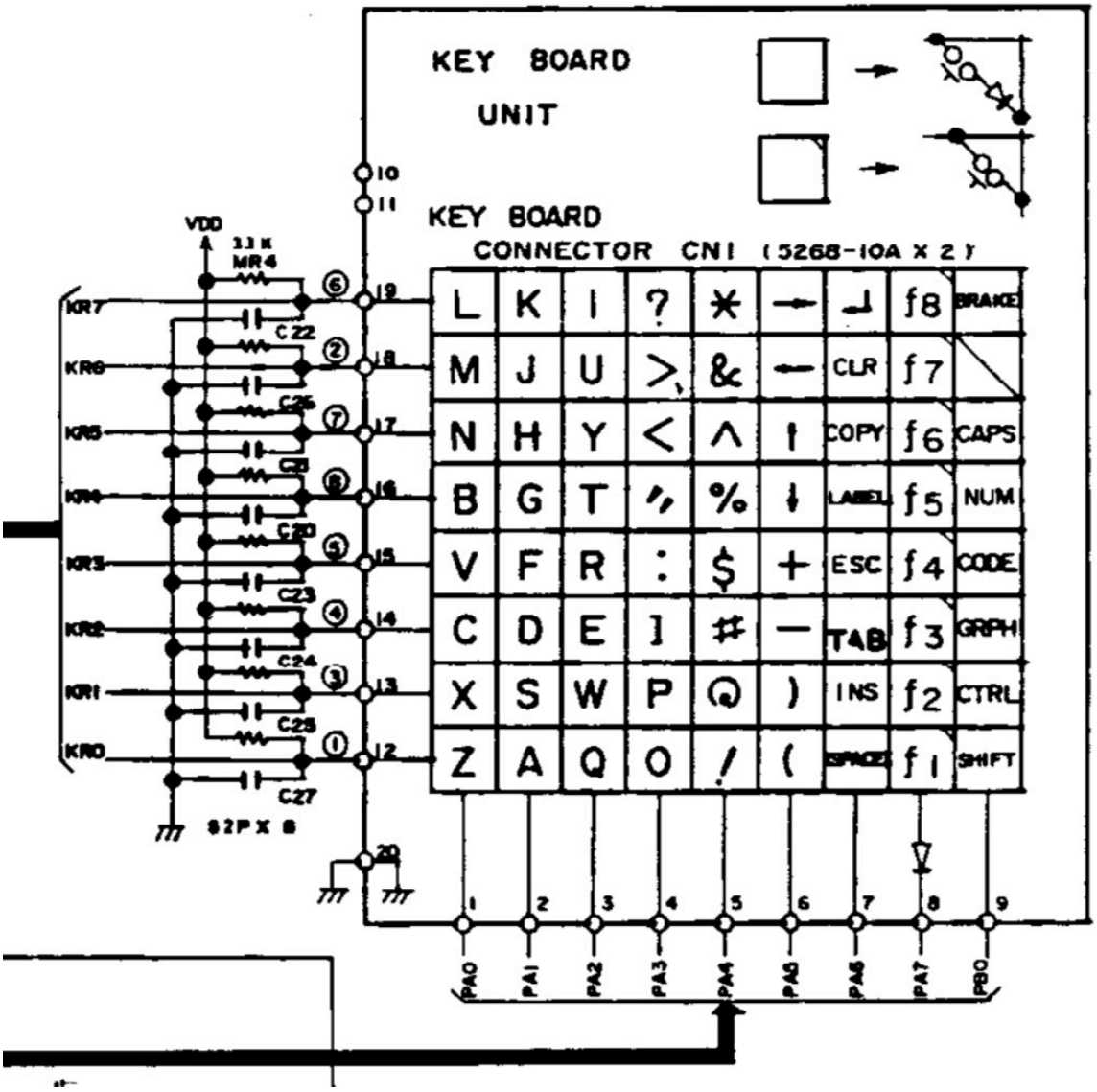

Figure 2: TRS-80 Model 100 Keyboard Matrix

```
#define USE_TIMER_3 true

#include <TimerInterrupt.h>
#include <ISR_Timer.h>

#include <Keyboard.h>

#define TIMER_INTERVAL_MS 5

#define num_cols 9
#define num_rows 8

int cols[num_cols] = { 0,  1,  2,  3,  4,  5,  6,  7, 8};
int rows[num_rows] = {21, 20, 19, 18, 15, 14, 16, 10   };

char charmap[72] = {
    /*      0    1    2    3     4    5                6              7       8 */
    /* 0 */ 'z', 'a', 'q', 'o',  '1', '9',             ' ',           KEY_F1, KEY_LEFT_SHIFT,
    /* 1 */ 'x', 's', 'w', 'p',  '2', '0',             KEY_BACKSPACE, KEY_F2, KEY_LEFT_CTRL,
    /* 2 */ 'c', 'd', 'e', ']',  '3', '-',             ' ',           KEY_F3, ' ',
    /* 3 */ 'v', 'f', 'r', ';',  '4', '=',             KEY_ESC,       KEY_F4, ' ',
    /* 4 */ 'b', 'g', 't', '\'', '5', KEY_DOWN_ARROW,  ' ',           KEY_F5, ' ',
    /* 5 */ 'n', 'h', 'y', ',',  '6', KEY_UP_ARROW,    ' ',           KEY_F6, ' ',
    /* 6 */ 'm', 'j', 'u', '.',  '7', KEY_LEFT_ARROW,  ' ',           KEY_F7, ' ',
    /* 7 */ 'l', 'k', 'i', '/',  '8', KEY_RIGHT_ARROW, KEY_RETURN,    KEY_F8, ' ',
};

void timerHandler(void)
{
    static int last_keys[num_rows * num_cols] = { 0 };
    static int next_keys[num_rows * num_cols] = { 0 };

    for (int c = 0; c < num_cols; c++)
    {
        digitalWrite(cols[c], LOW);

        for (int r = 0; r < num_rows; r++)
        {
            if (digitalRead(rows[r]) == LOW)
            {
                int idx = r * num_cols + c;
                next_keys[idx] = 1;
                Serial.print(r);
                Serial.print(",");
                Serial.print(c);
                Serial.print("\n");
            }
        }

        digitalWrite(cols[c], HIGH);
    }

    for (int i = 0; i < num_rows * num_cols; i++)
    {
        char key = charmap[i];

        // press if key is held down for two frames
        if (next_keys[i] && last_keys[i])
        {
            Keyboard.press(key);
```

Detour: Shift Register Experiments

Serial-In, Parallel-Out

The easiest way to reduce the number of digital I/O pins used by the keyboard firmware is to use a shift register. The 74HC595 is an 8-bit serial-in, serial or parallel-out shift register. It has an 8-bit storage register and an 8-bit shift register. Data is written to the shift register using serial input, and the contents of the storage register are shifted out in parallel.

This example displays binary number on eight LEDs using only three digital I/O pins.

```
int latchPin =  8; // pin connected to ST_CP of 74HC595
int clockPin = 12; // pin connected to SH_CP of 74HC595
int dataPin  = 11; // pin connected to DS of 74HC595

void setup()
{
    // set pins to output so you can control the shift register
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin,  OUTPUT);
}

void loop()
{
    // count from 0 to 255 and display the number on the LEDs
    for (int num = 0; num < 256; num++)
    {
        // take the latch pin low so the LEDs don't change while you're sending in bits
        digitalWrite(latchPin, LOW);

        // shift out the bits
        shiftOut(dataPin, clockPin, MSBFIRST, num);

        // take the latch pin high so the LEDs will light up
        digitalWrite(latchPin, HIGH);

        delay(500);
    }
}
```

Parallel-In, Serial-Out