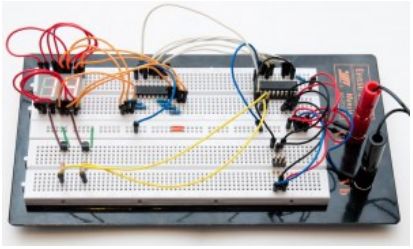
Feb 24 2012

7 Segment Multiplexing With ULN2003 & PNP Transistors

Electronics Add comments

The reason a started my electronics hobby was that I wanted to build a chess clock. Lacking a proper LCD display, I chose to multiplex several 7-segment displays. Most sources in the net did not specify hardware at all, and those that did were driving the segments with a 74HC595 shift register and using NPN transistors to enable one common cathode display at a time. However, if you look at 74HC595 specs you'll notice that it's not designed to source the amount of current that is required to drive several multiplexed 7-segment displays. It might work, but no one can say for how long!

It took me a while to find a good, inexpensive and readily available alternative. I finally found it in ULN2003, which is inexpensive darlington array that can drive 500 mA from each of its pins. So I decided to write a little tutorial on 7 segment multiplexing that walks through all the needed hardware and software in detail. Here's what we'll build (click for a larger image):



For this tutorial I assume you know how to connect ATtiny2313 to a programmer and flash it with custom software. You'll learn as much in IMakeProjects.com's AVR tutorial. You'll also need the following components:

- ATtiny2313 (actually, any AVR chip with 10 output pins will do)
- ULN2003, 7-channel darlington array (NPN, i.e. current sinks)
- 7 resistors, 330 ohm
- 2 seven segment displays, **common anode**
- 2 PNP transistors, e.g. BC558B, but almost any will do
- 2 resistors, 2.2 kohm
- 4.7 kohm pullup resistor(or similar value)
- 6-pin programming header
- A breadboard and jumper wires
- 5V DC voltage source (USB is one option)

Seven segment display basics

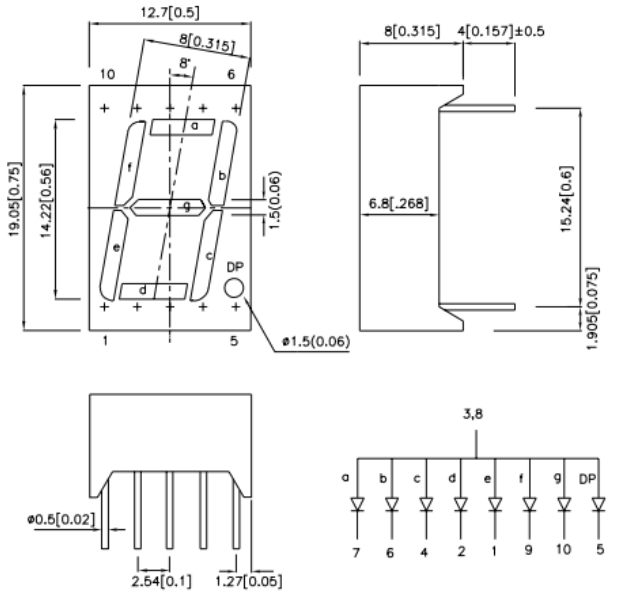
A seven segment display is basically just 8 LEDs (most include a decimal point) wired so that they share a common cathode or anode. Like normal LEDs, you need current-limiting resistors in series. Each segment needs its own resistor, otherwise LEDs get dimmer the more segments are lit. Number segments are labelled a-g and connected to "side pins", whereas common cathode/anode occupy the center pin on top and bottom of the display – they are internally connected so we only need to wire one of them.

Recent Writings

- 7 Segment Multiplexing With ULN2003 & PNP Transistors
- Default feed not disabled with Suffusion
- V-USB with ATtiny45 / ATtiny85 without a crystal
- Dissecting the Excalibur Game Time Chess Clock
- PicoScope 2204 USB Oscilloscope Review
- 8 bit and 4 bit LCD interfacing with ATtiny

Archives

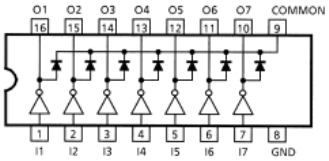
- February 2012 (9)
- January 2012 (3)
- December 2011 (1)
- March 2011 (1)
- July 2009 (1)
- May 2008 (1)
- April 2008 (1)
- February 2008 (1)
- January 2008 (7)



From the datasheet picture above, you can see how the pins are numbered (top left schematic), and which pin is connected to which segment (bottom right diagram).

ULN2003

The chips I got from Partco had the exact model of ULN2003APG. The datasheet tells that the chip has 2.7 kohm resisors on input side, so I1-I7 can be directly wired to PD0-PD6. Here's how the pinout looks like:

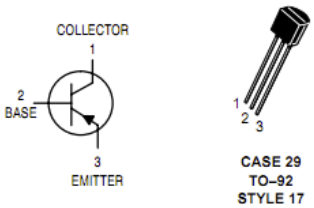


In this tutorial, we'll be wiring the "a" segments of both seven segment displays to a 330 ohm current-limiting resistor connected to O1 (only one display is on at a time so two displays can share a resistor even if two segments cannot). "b" segments are wired via another 330 ohm resistor to O2, and so on. Pretty simple!

Multiplexing common anode displays

Multiplexing means that you are actually switching only one display on at a time, but you do it so fast (at least 60 Hz) that the eye does not notice it. Easiest way to switch on a display is with a transistor, so the microcontroller does not need to sink/source all the current flowing through LEDs – even with only two displays it would most likely exceed MCU specs.

Most examples on the net use common cathode displays, in which case NPN transistor is called for, but the ULN2003 which we are using here for each segment is a current sink, which means individual segments need to be wired to it on "minus side" – so we need a common anode type of display. Because the "multiplex switch transistor" is now on VCC side instead of GND side, NPN transistor would not work. So we are using PNP transistor instead. See this article for more details on PNP switching (the previous page of that link shows NPN switching, by the way).

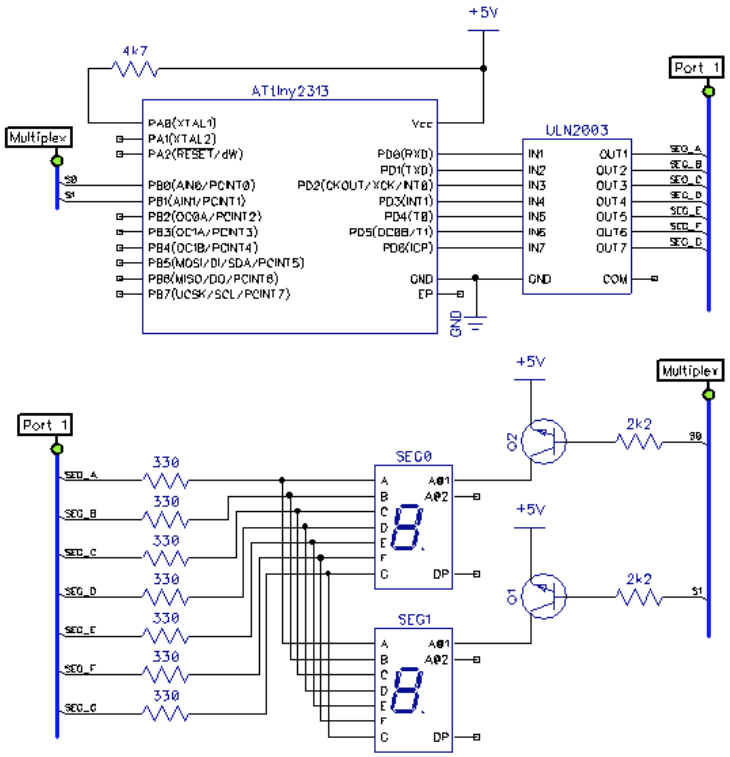


Note that for a PNP transistor, you wire the **emitter** to VCC (in this image it's pin 3) and **collector** to the common anode of the display.

Hardware setup and schematic

For simplicity, I chose to the 7-pin port D of ATtiny2313 for selecting the segments, and PB0+PB1 for first and

second digit display, respectively. Here's the full schematic – the wiring of the 6-pin programming is omitted for clarity:



Good order to wire these is the following:

1. Start with the ATtiny2313 and it's VCC, GND and RESET pullup resistor
2. Add the 6-pin header and wire it to ATtiny, VCC and GND
3. Add power and you can now test if you can program the ATtiny
4. Add ULN2003 and 330 ohm resistors, connect the ground pin and wire ATtiny PD0..PD6 to I1..I7
5. Add the first 7 segment display, and wire segments a-g to the 330 ohm resistors
6. Tip: Program ATtiny so it outputs VCC from all port D pins (DDRD=0x7F; PORTD=0x7F;) and temporarily wire seven segment display's common anode straight to VCC – now it's easy to start wiring segments to the correct resistors, as a segment lights up as soon as you plug a jumper wire in!
7. Add the second 7 segment display and connect the "a" segments of both displays, then "b" segments and so on
8. At this point you can test if everything except the transistor switching works by wiring the anodes straight to VCC and blinking the LEDs with ATtiny
9. Finally, add the transistors, connect their emitters to VCC, bases to PB0 & PB1 and collectors to display 1 & 2, respectively

Software

First thing to tackle is the display of numbers 0 to 9 using PORTD. With a few defines it's easy to construct the necassary bitmasks:

```
#define SEG_ALL 0x7F
#define SEG_A 1
#define SEG_B 2
#define SEG_C 4
#define SEG_D 8
#define SEG_E 16
#define SEG_F 32
#define SEG_G 64

// display given digit in seven-segment display
void display(uint8_t n) {
    switch(n) {
        case 0: PORTD = SEG_ALL-SEG_G; break;
        case 1: PORTD = SEG_B+SEG_C; break;
        case 2: PORTD = SEG_ALL-SEG_F-SEG_C; break;
        case 3: PORTD = SEG_ALL-SEG_F-SEG_E; break;
        case 4: PORTD = SEG_F+SEG_G+SEG_B+SEG_C; break;
```

```
        case 5: PORTD = SEG_ALL-SEG_B-SEG_E; break;
        case 6: PORTD = SEG_ALL-SEG_B; break;
        case 7: PORTD = SEG_A+SEG_B+SEG_C; break;
        case 8: PORTD = SEG_ALL; break;
        case 9: PORTD = SEG_ALL-SEG_E; break;
        default:PORTD = 0; break;
    }
}
```

To have something useful to display, we'll make the 16-bit timer1 count seconds which we will then display. For multiplexing, we'll use 8-bit timer0 with prescaler of 8 and overflow interrupt. At 1 MHz (ATtiny2313 with default fuses), this results in 10^6 / 256 / 8 = ~500 calls per second. Each call, we turn off one display, reconfigure the segments to light up, and turn on the other display. This results in ~250 Hz refresh rate which should be plenty:

```
volatile uint8_t timer_seconds = 0;

ISR(TIMER1_COMPA_vect) { // 1 Hz counter
    timer_seconds++;
}

volatile uint8_t active_display = 0;

ISR(TIMER0_OVF_vect) { // multiplex code
    PORTB |= (1 << active_display); // display OFF (VCC at base)

    active_display ^= 1; // toggle between 0 and 1

    if(active_display == 0) // prepare next digit
        display(timer_seconds % 10);
    else
        display((timer_seconds/10) % 10);

        PORTB &= ~(1 << active_display); // display ON (GND at base)
}
```

Note that because we are using a PNP transistor, we actually need to "write 1" to a pin to switch the current off! Now all we need to do is initialize the ports and timers and let the interrupts do their magic:

```
int main(void) {
        // initialize ports
        DDRD = SEG_ALL; // PD0-PD6 as outputs
        DDRB = (1<<PB0) + (1<<PB1); // PB0+PB1 as segment selectors

        // init timer 0 (multiplex)
        TIMSK |= (1 << TOIE0); // timer 0 (8-bit) overflow interrupt

        // init timer 1 (100 Hz clock and logic)
        TCCR1B |= (1 << WGM12); // configure for CTC mode
        TIMSK |= (1 << OCIE1A); // CTC interrupt
        OCR1A = 15625; // with prescaler 64, results in 1 Hz @ 1 MHz

        sei(); //  enable global interrupts

        // start timers
        TCCR0B |= (1 << CS01); // timer 0 at clk/8
        TCCR1B |= (1 << CS11)+(1 << CS10); // timer 1 at clk/64

        while(1) {} // loop forever

        return 1;
}
```

This is actually all of the code needed for the project. Just add includes to `avr/io.h` and `avr/interrupt.h` and you are all set. For the lazy ones, here's the complete source file.

Once you have this one working, it is of course easy to add a shift register like the 74HC595 in front of ULN2003 to reduce the amount of pins needed on AVR side (3+2 required with a '595, and using I2C I/O expander it could even be reduced to 2+2 – and of course you can do the same on multiplex side to switch up to 8 displays with only 2-3 pins).

Thanks for reading! If you liked the tutorial, do subscribe to the RSS feed, there's more in the works. :)

Posted by jokkebk at 00:07 Tagged with: 7 segment, ATtiny, ATtiny2313, AVR, multiplexing, pnp, seven segment, uln2003

V-USB with ATtiny45 / ATtiny85 without a crystal