Door Widget Seven-Segment Scrambling

2010-05-09 14:05:15 by Chris

One feature of the Door Widget is its rotating passcode entry system. This feature requires the display digits to scramble randomly each time a passcode will be entered.

The 10 digits are stored in an array and these values are spit out to the displays, one at a time, every timer interrupt. The logical way to rotate the digits was to randomly pick two displays, swap their digits, and repeat this process enough times to sufficiently jumble the digits.

I needed to generate random numbers to represent indices in the array of digits. There are several ways to generate random integers in C on the AVR. The easiest method that generates a char from 0 to N-1 is shown below. Note: The code examples below generate characters but could just as easily generate integers.

```
// Generating random characters between 0 and N-1
unsigned char r1 = rand()%N;  // Not so random
```

Below is a better method. Again it generates a random character from 0 to N-1. The first example uses floating point math (slow) and the next does not. RAND_MAX is an ANSI constant defined in stdlib.h and is equal to 0×7FFF. N must be much less than RAND_MAX (10 << 0x7FFF).

```
// Generating random characters between 0 and N-1

// Better method (floating point)
unsigned char r2 = (char)((double)rand()/((double)RAND_MAX+1)*N);

// Same method without floating point
unsigned char r3 = rand()/(RAND_MAX / N + 1);
```

I tested the r2 example (floating point method) on the Door Widget hardware and the time necessary to perform even 10 swaps created a noticeable lag in execution. I ended up going with the non-floating point, r3 example at the bottom. Fifty swaps are able to execute relatively quickly. The relevant C code is shown below.

```
unsigned char get_random(void)
{
    unsigned char N = 10;
    //return (char)((double)rand() / ((double)RAND_MAX + 1) * N);
    return rand() / (RAND_MAX / N + 1);
}

void scramble(void)
{
    unsigned char i,r1,r2,c1,c2;

    for (i=0; i<=50; i++)
    {
        r1 = get_random();
        r2 = get_random();

        c1 = current_char[r1];
        c2 = current_char[r2];

        current_char[r1] = c2;
        current_char[r2] = c1;
    }
    return;
}
```

I was surprised to find that this method had an interesting effect on the displays. The seven segment

displays all visibly shuffle every time scramble() is called. My original plan was to scramble a temporary copy of the digits and then copy back to current_char[]. I only stored the new values to current_char[] directly for debugging purposes. I like the effect, however, and will probably keep it like this.

Source:
C FAQ: Question 13.16 - http://c-faq.com/lib/randrange.html
