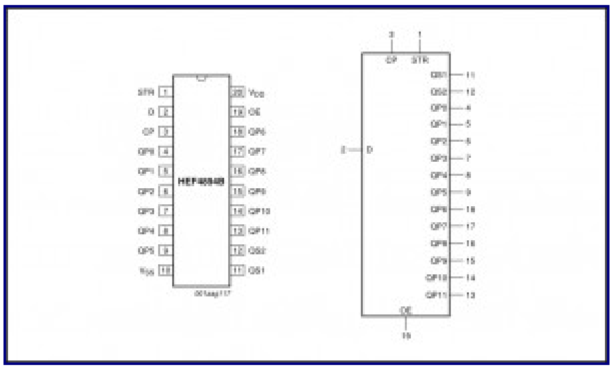
Shift Register and Seven Segment LEDs

2010-04-21 13:04:14 by Chris



The Door Widget requires controlling 10 seven-segment LED displays and one bi-color LED. My current design drives the LED anodes through eight 2N7000 MOSFETs and sinks each display anode (one at a time) through the NXP HEF4894B (Datasheet), a 12-stage shift register / LED driver (see Figure 1 for the pinout). I believe that this design is better suited for this project than Charlieplexing because:

- The firmware is easier to write and much-less taxing for the AVR
- The LEDs can be driven at a higher current that what the AVR I/O pins can provide (a necessity when each LED has a 1/10 duty cycle)

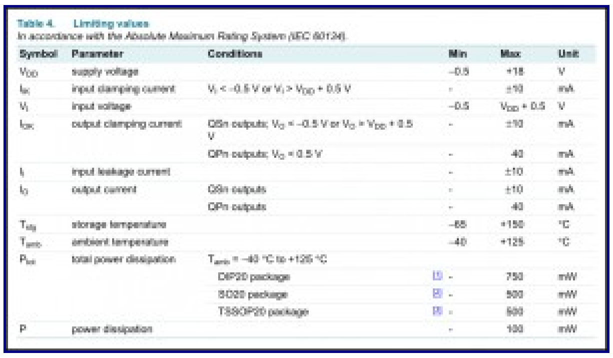


Table 4. Limiting values

In accordance with the Absolute Maximum Rating System (IEC 60134).

Symbol	Parameter	Conditions		Min	Max	Unit
V_{DD}	supply voltage			−0.5	+18	V
I_{IK}	input clamping current	$V_I < -0.5$ V or $V_I > V_{DD} + 0.5$ V		-	±10	mA
V_I	input voltage			−0.5	V_{DD} + 0.5	V
I_{OK}	output clamping current	QSn outputs; $V_O < -0.5$ V or $V_O > V_{DD} + 0.5$ V		-	±10	mA
		QPn outputs; $V_O < 0.5$ V		-	40	mA
I_I	input leakage current			-	±10	mA
I_O	output current	QSn outputs		-	±10	mA
		QPn outputs		-	40	mA
T_{stg}	storage temperature			−65	+150	°C
T_{amb}	ambient temperature			−40	+125	°C
P_{tot}	total power dissipation	T_{amb} = −40 °C to +125 °C				
		DIP20 package	[1]	-	750	mW
		SO20 package	[2]	-	500	mW
		TSSOP20 package	[2]	-	500	mW
P	power dissipation			-	100	mW

The HEF4894B has certain features that make it well-suited for driving the LED segments. Firstly, it is a 12-stage shift register which is *really* nice. Eight stages would have been too few, and 16 stages would have seemed wasteful. Secondly, it is capable of sinking 40 mA per channel with a total power dissipation limit of 750 mW (see Figure 2 for limiting values). I am only using ten of the output pins, and each one is only sinking 30 mA at the moment. It would be nice in the future to allow manual control of the LEDs' brightness, or to enable the door widget to adjust the brightness automatically by detecting the amount of ambient light. I will not have time to implement this feature by the end of the semester however.

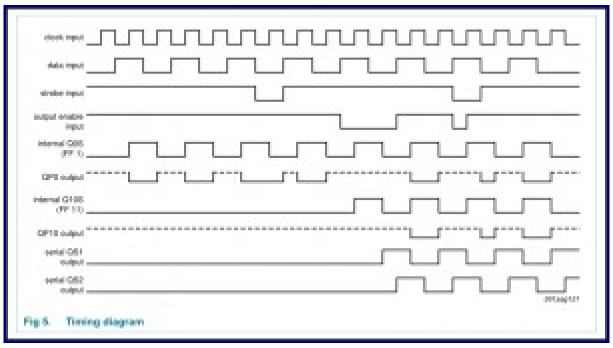


Fig 5. Timing diagram

The shift register takes commands through an SPI bus (timing diagram shown in Figure 3). Its pins are connected as follows:

- **STR** (Strobe) – This pin synchronizes the data currently being held in the internal shift register portion with the storage register portion. It is connected to pin B3 on the microcontroller.

- **D** (Data) – This pin takes in serial data from the microcontroller representing the state of the QP1 through QP11 output pins. Because there are 12 outputs, two bytes must be sent to fully set the outputs (MSB first). It is connected to pin B5 (MOSI) on the microcontroller.
- **CP** (Clock) – This is the serial clock and is connected to pin B7 (SCK) on the microcontroller.
- **QP1 - QP11** – These are the parallel output pins and are connected to the gate pins of the MOSFETs.
- **QS1 - QS2** (not used) – These are the serial out pins used for cascading multiple shift registers together.

Here is an excerpt from sevenseg.c that demonstrates sending data to the shiftregister through the SPI bus (light_up function) as well as the shift register initialization function (shiftreg_init) that appropriately sets up the SPI control register (SPI interrupt enable, Master select).

```
volatile unsigned char current_display;
volatile unsigned char current_char[] = {16,16,16,16,16,16,16,16,16,16};

void light_next(void)
{
    if (current_display++ > 9)    current_display = 0;

    light_up(current_display);

    return;
}

void light_up(unsigned char j)
{
    unsigned char lsb=0,msb=0;
    const prog_uint8_t* ch;
    ch = FONT_SEVEN_SEG + current_char[current_display];
    unsigned char b = pgm_read_byte(ch);

    switch (j)
    {
        case 0:  lsb = (1<<0);  break;
        case 1:  lsb = (1<<1);  break;
        case 2:  lsb = (1<<2);  break;
        case 3:  lsb = (1<<3);  break;
        case 4:  lsb = (1<<4);  break;
        case 5:  lsb = (1<<5);  break;
        case 6:  lsb = (1<<6);  break;
        case 7:  lsb = (1<<7);  break;
        case 8:  msb = (1<<0);  break;
        case 9:  msb = (1<<1);  break;
        default: lsb=0;  msb=0;
    }

    // Turn off shift register outputs
    SR_OUTPUT_OFF;
    // Display is now off!
    // Set up port pins with correct outputs to display current digit
    SEGMENTS_OUT = b;
    //SEGMENTS_OUT = FONT_SEVEN_SEG[17];
    // Transfer new values to shift register
    bit_clr(SPSR, 7); // Clear SPI interrupt flag
    SPDR = msb; // Write out MSB to shift register
    // Wait for shift register to receive byte
    while(!(bit_get(SPSR,7))) { }
    SPDR = lsb; // Write out LSB to shift register
    // Wait for shift register to receive byte
    while(!(bit_get(SPSR,7))) { }
    // Shift-in values
    SR_STROBE_ON;
    SR_STROBE_OFF;
    // Turn on shift register outputs
    SR_OUTPUT_ON;
    // Display is now on (with new digit)
    return;
}

void shiftreg_init(void)
{
    SR_STROBE_OFF;
    SR_OUTPUT_OFF;
    // Set SPI control register appropriately (SPI interrupt enable, Master select)
    //SPCR = 0b01010011;
    SPCR = 0b01010000;
}
```