External Registers (Jumping Game)

Intro

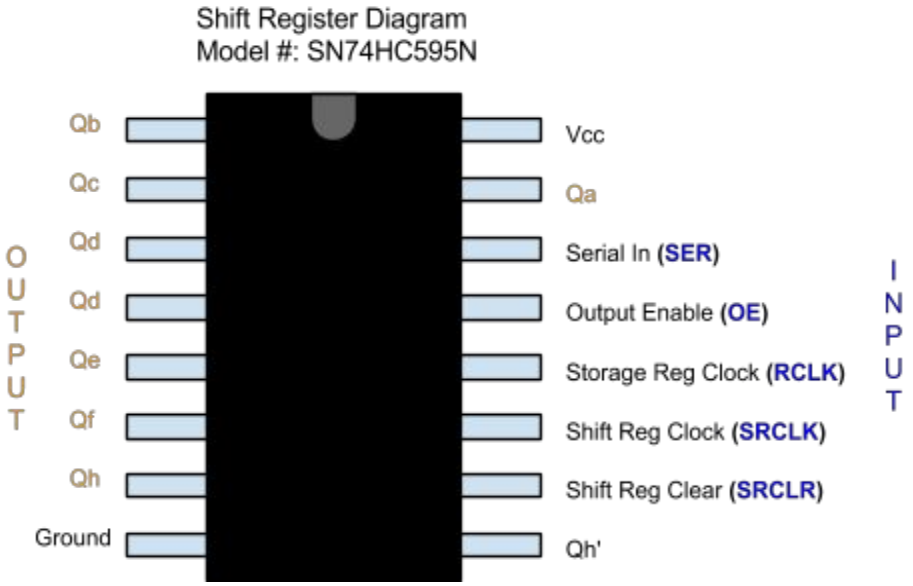
Why use an External Register

Using an external register, such as a shift register, can increase the number of microcontroller output pins available for other purposes. For example, suppose a project requires 8-bits to be displayed on 8 LEDs. The most straightforward approach would be to output the 8-bits using 8 microcontroller pins. Instead, by using a shift register, only 5 pins are needed, thus freeing 5 pins for other uses.

What the Shift Register Does

The shift register used in this lab contains two internal 8-bit registers named the **“Shift”** register and the **“Storage”** register. The “Shift” register is used to serially receive and store up to eight bits of data. The data stored in this register cannot be displayed on the output pins (Qa-Qh). However, the data stored in the “Shift” register can be copied, in parallel, to the “Storage” register at any time. From the “Storage” register, the data can be displayed on the output pins if output is enabled. Shifting data into the “Shift” register does not affect the contents of the “Storage” register.

A diagram of the shift register used in this lab is given below. The datasheet can be found in the GDrive folder under 'datasheets'.



How to Operate the Shift Register

1) Transmitting Data

- On the shift register, four pins are necessary for serially transmitting, and storing, up to a byte of data: **SRCLR, SER, SRCLK**, and **RCLK**. These can be connected to any four pins on the microcontroller -- we will outline below the preferred pins for this lab.
- When ready to begin transmitting, set **SRCLR** high and **RCLK** low.
- The following three steps should be repeated for every bit to be transferred.
    - 1) Set **SRCLK** low.
    - 2) Set **SER** equal to the bit value to be transmitted (0 or 1).
    - 3) After **SER** has been set, then set **SRCLK** high. The rising edge of **SRCLK** will shift in the contents of **SER** to the first spot of the“Shift" register and the remaining contents of the “Shift" register will be shifted over one spot. The bit stored in the most significant bit of of the “Shift" register will be 'bumped off' and lost.
- After all of the data has been transmitted, set **RCLK** high. This copies the contents of the “Shift" register to the “Storage" register. The copy takes place on the rising edge of **RCLK**.

  NOTE: To conserve a pin, **RCLK** and **SRCLK** can share the same clock line. Keep in mind that if they do, the contents of the “Shift” register will be a single clock cycle ahead of the contents of the “Storage” register. For example, shifting from right to left, if the “Shift” registers contains the value: 0000 1111, then the “Storage” register will contain: 0000 0111.

- After copying the contents of the “Shift” register to the “Storage” register, drop **SRCLR** low to clear the “Shift” register and cease transmitting data to the shift register. If **SRCLR** is dropped low before setting **RCLK** high, the transmitted data will be cleared before it can be stored.

Here is a function used to transmit an 8-bit char into a shift register.
This example assumes RCLK and SRCLK do NOT share the same clock line.

For the following function. Use the following connections:

- PC3 connected to SRCLR
- PC2 connected to SRCLK
- PC1 connected to RCLK
- PC0 connected to SER

- OE connect to GND (causes the "Storage" register to always output)
- Be sure to set up V_{CC} and GND
- If you are using PORTC ensure you disable the JTAG fuse.

```
// Ensure DDRC is setup as output
void transmit_data(unsigned char data) {
    int i;
    for (i = 7; i >= 0 ; --i) {
        // Sets SRCLR to 1 allowing data to be set
        // Also clears SRCLK in preparation of sending data
        PORTC = 0x08;
        // set SER = next bit of data to be sent.
        PORTC |= ((data >> i) & 0x01);
        // set SRCLK = 1. Rising edge shifts next bit of data into the shift register
        PORTC |= 0x04;
    }
    // set RCLK = 1. Rising edge copies data from the “Shift” register to the
“Storage” register
    PORTC |= 0x02;
    // clears all lines in preparation of a new transmission
    PORTC = 0x00;
}
```

2) Displaying Data

- While **OE** is set low, the contents of the “Storage” register are displayed on the output pins (Qa - Qh). When **OE** is set high, the contents of the “Storage” register are not displayed on the output pins.

NOTE: Connecting **OE** directly to ground ensures that the contents of the “Storage” register are constantly displayed on the output pins.

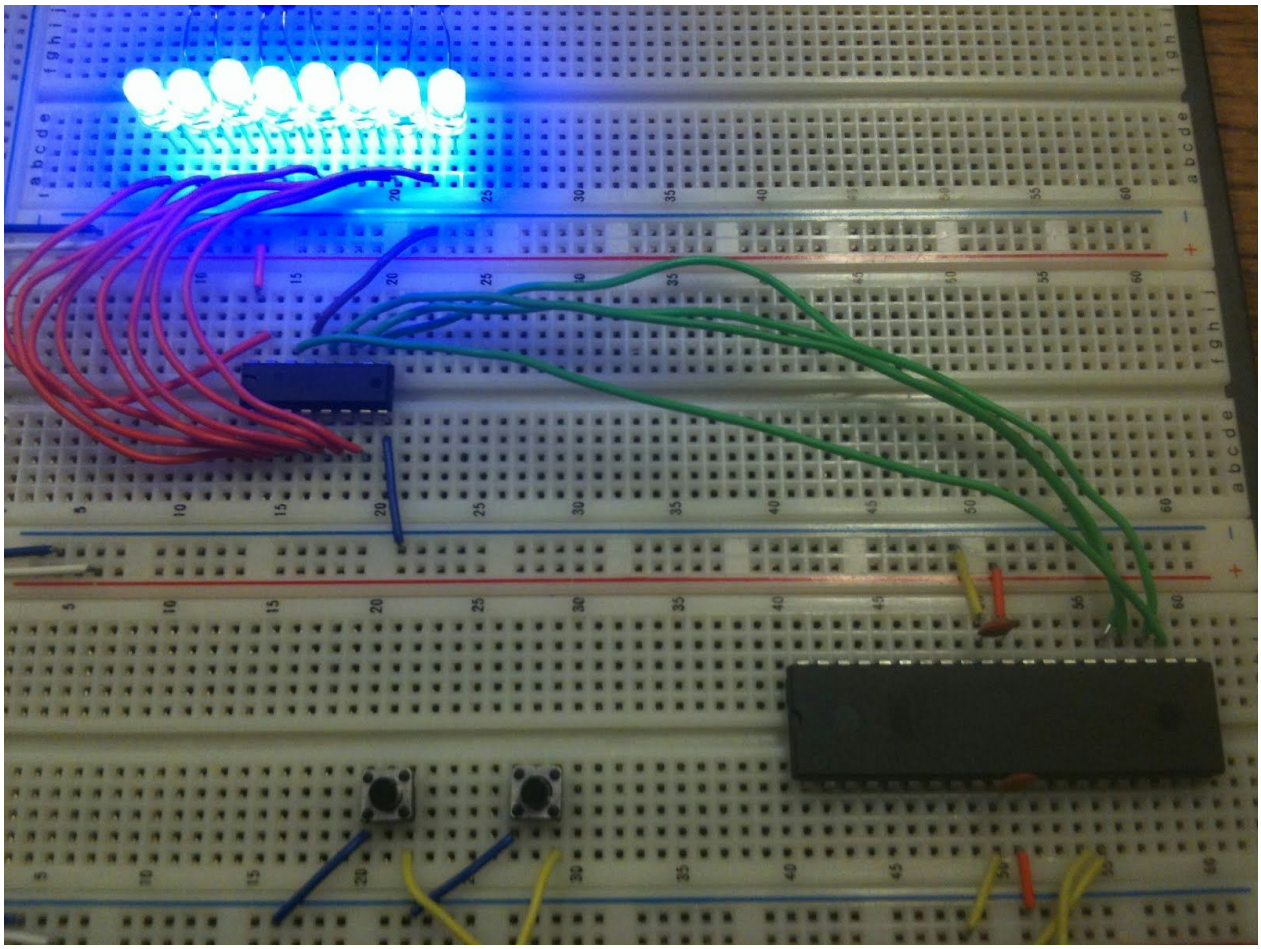
Pre-Lab:

NOTE: The transmit_data(data) function can work without the task scheduler in place. Try a simple program first to ensure your wiring is correct.

NOTE: We have included scheduler.h in the ".h files" folder on GDrive, this can simplify the process of setting up the task scheduler. You will need to include the file's directory as part of your project. Also, there is a main.c which shows an example of how to create tasks when using the scheduler.h. This code will work for both ATMega1284 and ATMega32.

With the “transmit_data” function provided in the intro, come prepared with a working

synchSM design that will display an 8-bit char value on the external shift register. A photo of an example breadboard setup is given below.



- Qa - Qh are connected to the LEDs
- SER, SRCLK, SRCLR, and RCLK are connected the the micro-controller
- OE is connected directly to ground

Part 1: Incrementing/Decrementing a Value on a Single Shift Register

Design a system where a ‘char’ variable can be incremented or decremented based on specific button presses. The value of the variable is then transmitted to the shift register and displayed on a bank of eight LEDs.

Criteria:

- Two buttons are required. One button increments the variable. The second button decrements the variable.
- The variable cannot be incremented past 0xFF or decremented below 0x00.
- If both buttons are pressed simultaneously, the variable is reset to 0x00.
- The variable is transmitted, and displayed, on the shift register whenever the variable changes value.

Hints:

- Use the “transmit_data” function, provided in the intro, to transmit the changing variable to the shift register.

Video Demonstration: http://youtu.be/kuS1MHpxJ1w

Part 2: Festive Light Display

Design a system where one of three festive light displays is displayed on the shift register’s LED bank. The choice of festive light displays is determined by button presses (See video link for examples of some festive light displays).

Criteria:

- Two buttons are used to cycle through the three available festive light displays
- One button cycles up, the other button cycles down.
- Pressing both buttons together toggles the system on or off
- Choose a default display for the system to start with. Whenever the system is turned on, the display shown on the LEDs should start with the default display.

Hints:

- Write a synchSM for each festive light display and use a shared “go” variable to determine which synchSM is allowed to write to the shift register.
- A fourth synchSM can be used to monitor button presses and adjust the value of the “go” variable.

Video Demonstration: http://youtu.be/lhBw5hU6Tz8

Part 3: Two Festive Light Displays

Expand upon part 2 of the lab by adding another shift register and two buttons. Design a system where two different festive light displays can be displayed on two separate shift

registers simultaneously. The selected festive light display for each shift register, is determined by the two buttons designated to that shift register.

Criteria:

- Two buttons control the festive light display shown on shift register 1. The other two buttons control the festive light display shown on shift register 2.
- For each shift register, if their designated buttons are pressed simultaneously, the LED display on that shift register is toggled on or off.
- The two shift registers should share the following lines: SER and SRCLK.
- RCLK and SRCLR should be independent for each shift register.
- When writing to either shift register, the same “transmit_data” function should be called.

Hints:

- Modify the “transmit_data” function so that different RCLK and SRCLR lines are used depending on which shift register number is passed as a parameter.
- If a new shared “go” variable is introduced, it is possible to complete this part of the lab without adding any additional synchSMs.

Video Demonstration: http://youtu.be/ybVjDfjq5TI

Extra Credit 1: Daisy Chaining Shift Registers

When shift registers are “Daisy Chained” together, this means that the shift registers are connected in series where the output of one shift register is connected to the input of another shift register. “Daisy Chaining” shift registers together allows for eight additional outputs for the cost of two microcontroller pins.

“Daisy Chain” two shift registers (Sreg1 and Sreg2) which displays a single illuminated LED that shifts back and forth across all 16 LEDs, bouncing the opposite direction when the illuminated LED hits the far left or far right LED.

Criteria:

- “Daisy Chain” Sreg1 with Sreg2 by connecting Qh of Sreg1 to SER of Sreg2.
- A ‘short’ variable should be shifted into Sreg1.
- Sreg1 and Sreg 2 should have independent clock lines
- For Sreg1 and Sreg2, have the local RCLK and SRCLK share the same clock

line.

Hints:

- Instead of pulsing Sreg1 and Sreg2's clock lines in parallel, pulse them sequentially. Doing this ensures that Sreg1's Qh is up-to-date before shifting it into Sreg2.
- Since RCLK and SRCLK share the same clock line, once the entire short value has been shifted into the "Daisy Chained" system, an additional clock pulse will be necessary to update the LEDs since the "Storage" register is a clock cycle behind the "Shift" register.
- Make changes to the provided "transmit_data" function that reflect the transmitting functionality of your design.

Video Demonstration: http://youtu.be/4ryYqDwJ3jw

Extra Credit 2: The Jumping Game

Using two "Daisy Chained" shift registers, design the following game. A player controller illuminated LED needs to avoid a collision with a game controlled (enemy) illuminated LED. The player can move their LED left or right, and can jump left or right. A collision can be avoided by jumping over the enemy LED.

Criteria:

- Three buttons control the player LED: Left, Right, and Jump
- Pressing Left or Right on their own will shift the player LED by one bit in the desired direction
- A jump occurs when the Jump button is pressed simultaneously with Left or Right button. The direction of the jump is determined by what directional button is pressed with the Jump button.
- A jump should shift by 4 bits.
- A jump should not shift past an extreme. For example, jumping right from 0x0004, should result in 0x0001 instead of 0x0000;
- The enemy LED shifts by one bit in one direction until it hits an extreme, then changes directions. For example, if the enemy is shifting left and equals 0x8000, the enemy should now change direction and begin shifting right until it equals 0x0001.
- The Enemy LED moves at a constant rate

Hints:

- Use three synchSMs to control the game functionality. One for the player LED,

one for the enemy LED, and one to display the player and enemy positions.

- Use shared short variables for player and enemy. If these variables are equivalent, then a collision has occurred.
- To display the player and enemy positions, OR the player and enemy variables together, and transmit the result using the transmit_data function from the previous Extra Credit.

Video Demonstration: http://youtu.be/6DOexqEb33A

Each person must submit their .c source files according to instructions in the Lab submission guidelines.