Lab 6:
Digital Electronics –Shift Registers

At some point, as you are working on more and more complex Arduino projects (i.e. your final Project ☺), you may run out of output pins available on your Arduino. This can be a serious problem! For example, imagine that you want to run a lightshow with 64 LEDs, arranged in 8x8 matrix. In the best case scenario, you would require total of 16 wires to run this matrix. Well, this would pretty much tie-up all digital outputs of Arduino, and you would not be able to do anything else with it. Even worse, imagine that you want to run 64 RGB (color) LEDs – now you need a lot more wires! Or what about this https://www.youtube.com/watch?v=kfaW0sxBQzI (a very popular ES 50 final project, btw! ☺). You would need A LOT of outputs. How can we achieve this?

1. Objectives

- Introduce digital circuits with memory – so called *sequential circuits* (in contrast to combinatory circuits we studied in lectures)
- Demonstrate how shift register work, and how they can be used to store the information coming from Arduino
- Teach you how to control 8 (even 16, 24, etc) different devices (e.g. LEDs) using only one output data pin on Arduino board!

2. Background

In lectures, you were introduced to logic gates (AND, OR, NOT, NOR, etc.) and you learned how they could be used to realize more complex combinatory circuits (e.g. adder). *One common feature for all combinatory circuits is that they do not have memory* – outputs of these circuits depend on the current inputs, only. In other words, previous inputs (and outputs) play no role! This can be serious limitation for building complex digital circuits, including computers. Fortunately, electrical engineers figured out ways to realize memory using combinatory logic gates and “a little bit” of feedback - we will talk about this in lectures in great detail. For now, it is enough to say that this “trick” allows us to remember (latch) the state of the signal, that is memorize the signal history. This mechanism is used to realize registrars – basic memory elements in computers. Digital circuits with memory are called *sequential circuits*.

In this lab you will work with so called ***serial-in/ parallel-out shift registrar*** (74HC595 chip, Figure 1) – a very useful type of *sequential digital circuit*. Furthermore, this 74HC595 is synchronous system, that responds only when clock signal is present at its Pin 11 (for more on synchronous and asynchronous systems, you will have to wait for a lecture or two). Simply put, a shift register will let you expand on the digital outputs you have on your Arduino. Each one of these 74HC595s can act like 8 more digital outputs, and you can daisy chain them. So you could hook 8 of them up next to each other and have control of 64 outputs.

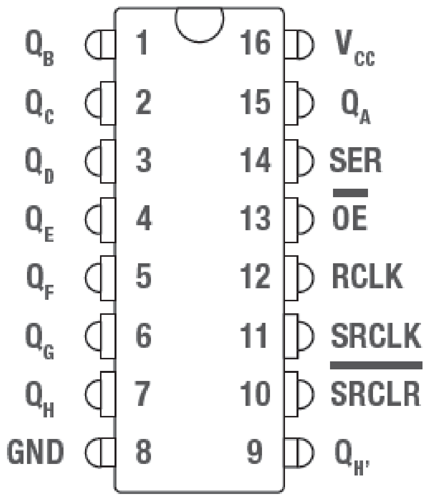


Figure 1. 74HC595 shift register. Output is in the form $Q_H Q_G Q_F Q_E Q_D Q_C Q_B Q_A$. Q_H is the most significant bit (MSB), and Q_A the least significant bit (LSB).

You can imagine a 74HC595 shift register as a row of 8 chairs. Each chair is either empty (0), or someone is sitting it (1). Now, every 10 seconds or so, someone rings a bell, and everyone has to get up and move one chair to the right. If there was someone in that rightmost chair, well they just go away. In that leftmost chair, you can either tell someone to sit in it, or just leave it empty. Now bringing this idea back to the 74HC595 (Figure 2): This shift register consists of 8 memory spots that can store one bit each (1 or 0). When signal on SRCLK (Serial Clock) pin is high (analogy to ringing the bell), bits from each spot shift to the right by one spot. The last pin (right most in this case) drops out, or it can be sent to next shift register, if needed. This process is illustrated in Figure 2. Importantly, 74HC595 has pin called RCLK (register clock, latch) that controls what information can be accessed. We can hold this pin LOW while loading information into the register and nothing on the display pins will change. Then when we are done loading info, and everything is how we want, we can pull the RCLK HIGH and the 74HC595 will display the information stored inside on its outputs. So even though we are changing values in the register in 8 steps, it looks like it was just one step (Figure 2).

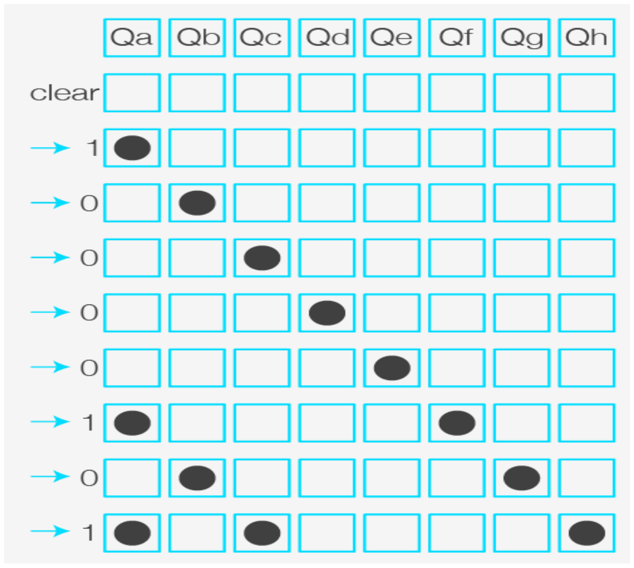


Figure 2. (source: http://bildr.org/2011/02/74hc595/) Time sequence of how byte 1000 0101 is written into the 74HC595: In first step 1 is written into Q_A. Then 1 from Q_A is moved to Q_B, and 0 is written into Q_A. Next, bits are moved one spot to the right, and new data (0 in this case) is written into Q_A. This process continues, and finally after 8 time steps, 1000 0101 is written into $Q_H Q_G Q_F Q_E Q_D Q_C Q_B Q_A$.

Summary:

- Whenever the signal on the SRCLK-pin (Figure 1) goes high, all the values get shifted to the right, and a new info (0 or 1) brought to pin 14 (SER for serial) is written in.
- After you shifted one byte in, you must set the RCLK pin HIGH in order to update the output-pins with the new data.
- You can 74HC595 to control 8 outputs at a time while only taking up a few pins on your Arduino. Furthermore, you can daisy-chain multiple shift registers together and extend your output even more.

3. 74HC595 Shift Register in Action

Now let's have some fun with 74HC595 and use it to control 8 LEDs.

a. **Wire up the 74HC595 chip** as shown in Figure 3. Make sure that Pins 8 and 13 are connected to GND (on Arduino), Pins 10 and 16 to +5V bias (on Arduino).

b. Connect the chip to Arduino: Arduino Pin 2 needs to be connected to 74HC595 Pin 14 (data in), Arduino Pin 3 to 74HC595 Pin 3 (clock signal), and Arduino Pin 4 to 74HC595 Pin 12 (latch, that is RCLK signal);

c. **Add 8 LEDs:** In this case you should connect the cathode (short pin) of each LED to a common ground, and the anode (long pin) of each LED to its respective shift register output pin. Using the shift register to supply power like this is called *sourcing current.* **Remember that Pin 15 on 74HC595 is LSB whereas Pin 7 is MSB!**

   Note: Some shift registers can't source current, they can only do what is called *sinking current.* If you end up working with these in future, you will have to flip the direction of the LEDs.

d. **Load code** *ShiftOut_Ex1.ino* **into your Arduino** and try it out. Study the code, and make sure that you understand every line.

   i. Modify the code to send a number <256 to shift register. What LED pattern do you get? Does that correspond to binary representation of your number? Try few different numbers.

   ii. Modify your code to write number 74 into the shift register and use oscilloscope to simultaneously display the clock signal (Pin 11 on 74HC595) and data signal (Pin 14). How many clock pulses do you see? What can you say about relationship between your data signal and LED status? What about relationship between clock and data signals? Feel free to try different numbers too.

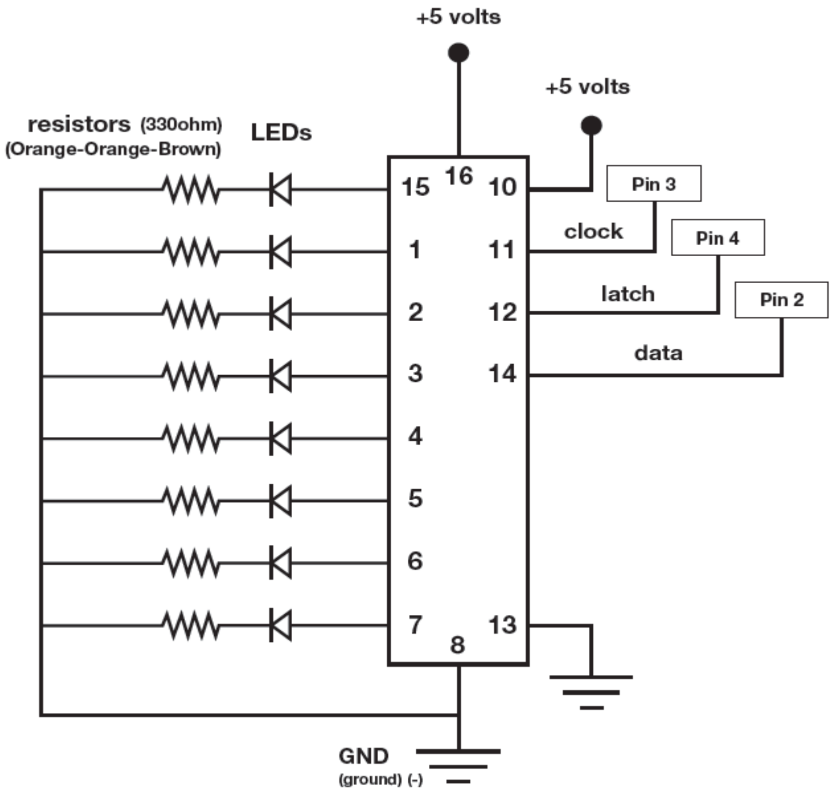


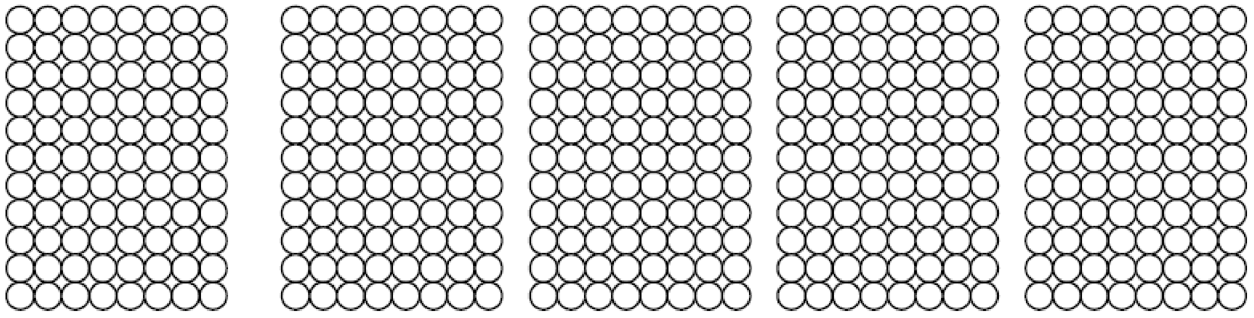
Figure 3. 74HC595 is "8-bit serial-in, parallel-out" shift register with output latches, that uses "synchronous serial communication" - i.e. you can pulse one pin up and down thereby communicating a data byte to the register bit by bit. Once the whole byte is transmitted to the register the information stored in each bit get parceled out to each of the individual output pins. This is the "parallel output" part, having all the pins do what you want them to do all at once.

iii. Now, try sending number >255 and see what you get. What is going on? What binary number do LEDs represent? Why?

iv. Modify the code again to send following sequence to shift register (one after another, in the same loop): 1, 2, 4, 8, 16, 32, 64, 128. What happens? Describe your observations in few words.

v. Send number 3 to shift register and record the status of LEDs. Now replace ‘MSBFIRST’ with ‘LSBFIRST’ in your code, and send number 3 to the shift register again. What happens now?

e. Load code *ShiftOut_Ex2.ino* into Arduino and run it. What does the program do? If you cannot figure it out, try marking the LEDs status in each step in the space below (from top down, left to right).



f. **Scaling up….** Hopefully, by now you are convinced that shift registers are very useful: using only one output pin on Arduino (along with two additional control pins: clock and latch) you were able to control 8 different LEDs, independently. Now, let's try to build on this and scale the system up. Shift registers have a pretty clever option built in that allows them to be chained or cascaded together. Remember how we said hat the last register just dumps its value when it is shifted over? Well the Pin 9 is where the 74HC595 dumps that value. So we just take that and use it as the input (Pin 14) on a next shift register in the chain, and bam! They are chained together. Note that you will need to control both registers with the same Arduino, and bring *clock* and *latch* signals to appropriate pins on both registers. Lets try this in practice. You can do this part in two ways – you can get another (or more) 74HC595 along with more diodes and resisters and wire this up, or you can work with another group and combine your circuits (Figure 4). Notice that you will need only one Arduino for this: both 74HC595 chips will have their Pins 11 connected to Arduino Pin 3, and Pins 12 to Arduino Pin 4. One of the chips will continue having its data Pin 14 connected to Arduino Pin 2 (this will be your *lower byte*). However, second shift register will have its data Pin 14 connected to the serial out pin (Pin 9) of the first shift register.

i. What is the largest number that you can write into two shift registers? Try sending out some big numbers (>255) and see if your results match your expectations. Code *ShiftOut_Ex3.ino* will be helpful for this. Notice how in code we are shifting two bytes, one after another.

ii. Now let's try our counter code again, this time with 16 different LEDs. Load Arduino code *ShiftOut_Ex4.ino* and see what happens. Btw, how long do you think it would take for this two-byte counter to count all the way up if delay ('wait' variable) is increased to 100ms? Do you want to wait this long?

iii. If you are adventurous, you could scale this up further, and use 3, 4 or more shift registers, simply daisy-chaining them as described above. It is powerful way to further expand the Arduino outputs. But, what price do you pay for this?

iv. **Optional**: If you have time, you can try daisy-chaining 3, 4 or more shift registers and have fun with it.

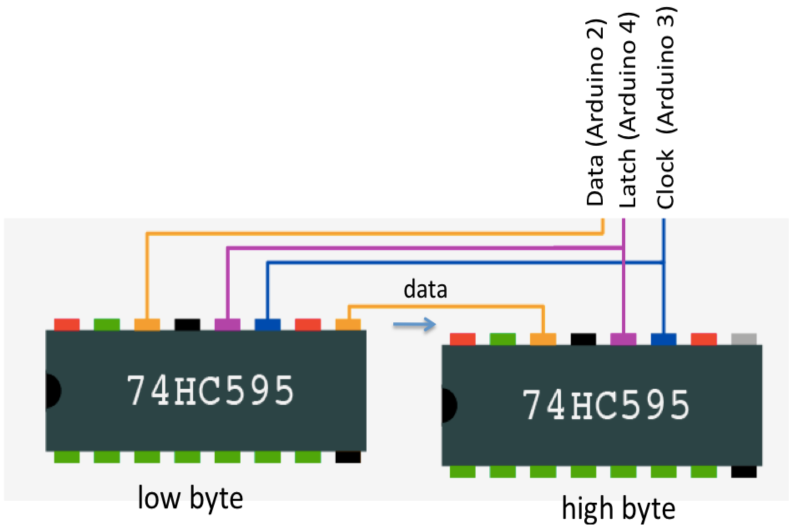


Figure 4. Daisy-chaining two shift-registers