Microcontroller Systems and Interfacing

week 12 experimental lab

1 Serial to parallel conversion

Using many LEDs (e.g., several seven-segment displays or bar graphs) is difficult, because only a few digital outputs are available. If each signal (LED) has one dedicated output pin, we are limited to at most 14 signals (digital pins 0 to 13). To connect $N > 14$ signals we need a different approach.

Instead of N *parallel* connections (giving each a dedicated output pin) we can use a single pin and send a series of N single-bit values. Each value represents one of the N different signals. This *serial* signal will be received by an external *serial to parallel converter* and converted back to the N parallel signals that we need. One simple device that can perform serial to parallel conversion is a *shift register*.

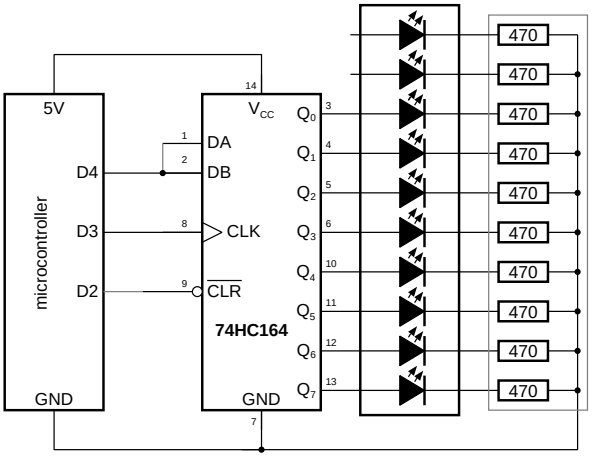
1.1 Build a bar-graph display interface using a shift register

► Disconnect your Arduino USB cable. **Do not connect it again until you have completed and then double-checked your circuit.** If your shift register is not properly connected, you will destroy it!

► Connect a 74HC164 shift register to the Arduino. The inputs can be connected directly to the microcontroller digital pins as follows:

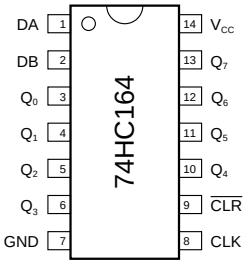
D2	→	$\overline{\text{CLR}}$	(pin 9 on 74HC164)
D3	→	CLK	(pin 8)
D4	→	DA and DB	(pins 1 *and* 2)

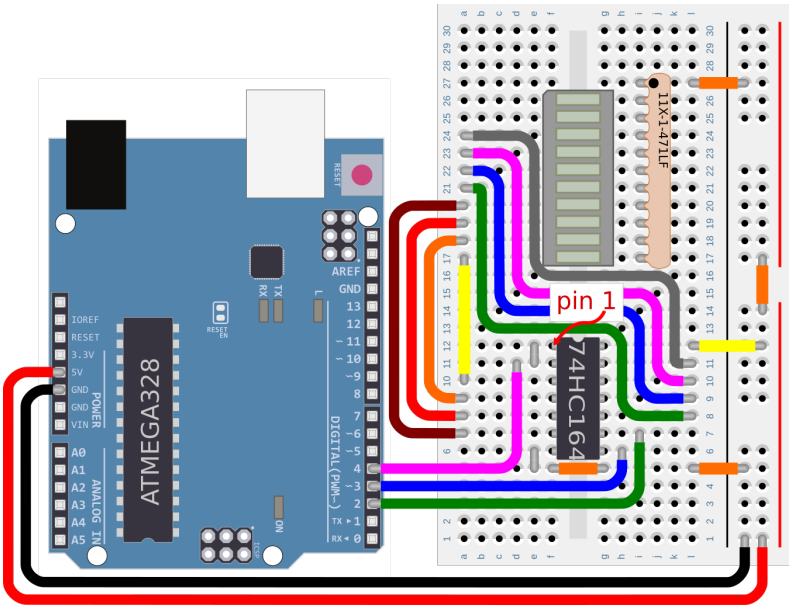
Don't forget to connect 5 V and GND from the microcontroller to the V_{CC} and GND pins (respectively) on the shift register.



► Connect the shift register outputs to the first eight LEDs of a bar-graph array.

The shift register can supply a total of 50 mA of current, or about 6.25 mA per LED. Each LED has a forward voltage drop $V_F = 2.2$ V, so you will have to use current-limiting resistors of at least $2.8/.00625 = 448\,\Omega$. The closest standard value is is 470 Ω. (Your kit includes a network of 10 resistors in a convenient single package, as shown in the diagram.)





► **Check again that the shift register and the LED display are correctly oriented!**

Pin 1 on the shift register is next to the small dot, at the end of the device marked with a semicircular notch. Pin 1 on the LED array is next to the corner that is angled at 45°.

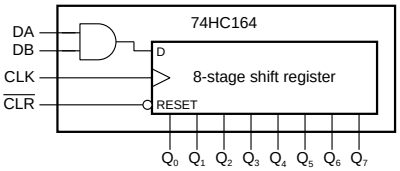
1.2 Create a program to test the shift register's operation

▶ Write a pair of functions called `posPulse(pin)` and `negPulse(pin)`.

The function `posPulse()` should generate a *positive* pulse on the given `pin`. (Use `digitalWrite()` to set `pin` first to `HIGH`, and then immediately back to `LOW`.)

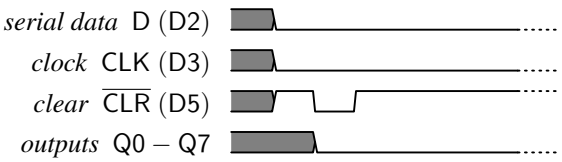
The function `negPulse()` should generate a *negative* pulse on the given `pin`. (Use `digitalWrite()` to set `pin` first to `LOW`, and then immediately back to `HIGH`.)

▶ In the `setup()` function, use `pinMode()` (or `DDRD`) to configure digital pins 2, 3 and 4 as `OUTPUT`s. and then use `digitalWrite()` (or `PORTD`) to initialise them as follows:



signal	*pin*	*initial state*	*explanation*
$\overline{\text{CLR}}$	D2	`HIGH`	disable the clear signal (which is active-low)
CLK	D3	`LOW`	ready to clock data in on a rising edge
D	D4	`LOW`	data value '`0`' (optional)

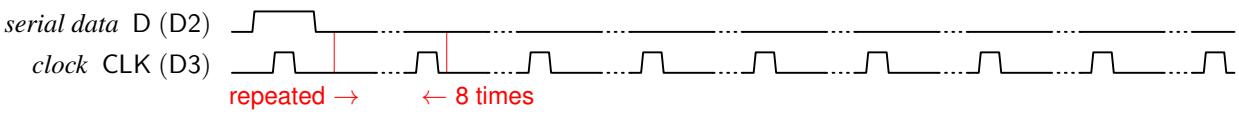
▶ Still in the `setup()` function, clear the shift register outputs by generating a *negative pulse* on $\overline{\text{CLR}}$.



▶ In the `loop()` function, test the register's operation by shifting a single bit through all eight stages:

- Set D to '`1`' (`HIGH`).
- Shift D into the register by generating a positive pulse on CLK.
- Set D to '`0`' (`LOW`).
- Repeat eight times:
  - generate a delay for 100 ms.
  - Shift D into the register by generating a positive pulse on CLK.
- generate a delay for 500 ms.

The resulting signals will follow this pattern:



If your circuit and program are correct you should see the LEDs light up one at a time in sequence.

? What happens if you disable (comment out) step 3 ('set D to `0`') in your `loop()` code?

1.3 Challenges

▶ (medium) Make the LEDs light up in the opposite order.

▶ (easy) Alternately turn all of the LEDs **on**, then turn them all **off**.

▶ (medium) Make the LEDs count in binary. (Remove your `delay()` and see what happens. Then try the following challenge...)

▶ (medium-difficult) Turn on LEDs 1, 3, 5 and 7 at **full** brightness, and LEDs 2, 4, 6 and 8 at **half** brightness.

▶ (difficult) Flash LEDs 1, 3, 5 and 7 at full brightness, while flashing LEDs 2, 4, 6 and 8 at half brightness.

1.4 Mini-projects

► (medium) Use an array to make lots of patterns.

This example program simulates an 'executive desk toy'. Modify the contents of the `leds` array to make lots of other patterns.

```
enum { CLK = 2, CLRN = 3, D = 4 };

void setup() {
  DDRD = 0b00011100;
  PORTD = 0b00000;
  PORTD = 0b01000;
}

void sendBit(int n) {
  PORTD = ((n & 1) << 4) | 0b1000;
  PORTD = ((n & 1) << 4) | 0b1100;
}

void sendWord(int n) {
  for (int i = 0;  i < 8;  ++i) {
    sendBit(n >> 7);
    n <<= 1;
  }
}

unsigned int leds[] = {
  0b00111001,
  0b00111010,
  0b00111100,
  0b01011100,
  0b10011100,
  0b01011100,
  0b00111100,
  0b00111010,
};

void loop() {
  for (int i= 0;
       i < (sizeof(leds) / sizeof(leds[0]));
       i = i + 1) {
    sendWord(leds[i]);
    delay(66);
  }
}
```

► (difficult) Add another shift register to your circuit.

Connect the Q_7 output on the first register to the D input on the second register. You now have a 16-bit shift register. Connect the first two outputs on the second shift register to the remaining two LEDs of the LED bar-graph display. Repeat some of the above challenges using all ten LEDs. Add another display, to make a total of 16 LEDs. (You will have to use two breadboards.)

Modify the above program to work with 16 LEDs. Redesign the LED patterns to make a '16-bit executive desk toy'.

► (medium-difficult) Use a more complex shift register.

Find a 74HC595 shift register in your hardware kit. It is a shift register with additional output enable ($\overline{\text{OE}}$) and register clock (RCLK) inputs. Connect these inputs to two additional digital outputs on the microcontroller.

Modify your program to correctly operate the 74HC595, testing it with some of the above challenges.

Microcontroller Systems and Interfacing

week 12 reference material

A Review of digital electronics

voltage	0 V	5 V
voltage level	`LOW`	`HIGH`
binary value	0	1
logic	false	true

Digital electronics deals with binary *signals*. A binary signal represents some logical information: yes or no, true or false, 1 or 0, and so on. These two possible *states* are represented by *voltage levels*, nominally 5 V and 0 V, which can also be written `HIGH` and `LOW`, respectively.

The precise voltage of a digital signal is often not important. What counts is whether it is above or below some *threshold*. For example, the popular 74HC devices using 5 V logic generate output voltages $\geq$ 3.98 V to represent `HIGH`, and $\leq$ 0.26 V to represent `LOW`. They consider any input voltage $\geq$ 3.1 V to be `HIGH` and any voltage $\leq$ 1.35 V to be `LOW`. (Voltages between these thresholds are undefined. They may be considered `HIGH` or `LOW` depending on many unpredictable factors including manufacturing process variations, temperature, etc.)

Another common standard uses 3.3 V instead of 5 V. 3.3 V devices are usually not compatible with 5 V logic, and can be damaged if exposed to voltages higher than 3.3 V. The data sheet for a device contains authoritative information such as the voltages it tolerates, and what levels it expects to represent `LOW` and `HIGH`.

A.1 Active-high and active-low signals

Every digital signal provides a yes/no answer to some question. When the signal represents 'yes' is it *active*; when it represents 'no' it is *inactive*.

For example, some devices only respond to input when they are *selected* (the entire chip is enabled). These devices have a *chip select* input. When the 'chip select' input is *active* the device will respond to its other inputs; when 'chip select' is *inactive* the device ignores its other inputs.

Either of the two logic levels (`HIGH` or `LOW`) can be used to represent 'active' (and the opposite level used to represent 'inactive'). Signals that are *active high* are considered active when their level is `HIGH`. Signals that are *active low* are considered active when their level is `LOW`.

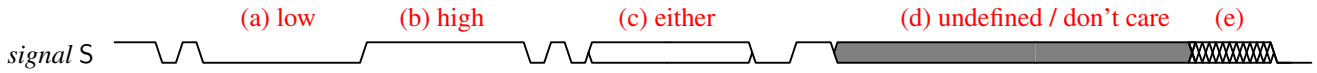
	signal meaning	
voltage	*active high*	*active low*
`LOW`	inactive (off, 'no')	active (on, 'yes')
`HIGH`	active (on, 'yes')	inactive (off, 'no')

The active level for a signal is almost always indicated in its name. The two most common conventions are to write active-low signals with a line over their name, or the letter 'N' after their name. For example, if a 'chip select' signal is active high then it might be called CS; if it is active low then it might be called $\overline{\text{CS}}$ or CSN.

Active low signals are sometimes identified in circuit diagrams by drawing a small hollow circle at the point where the signal enters or leaves the device.

A.2 Timing diagrams

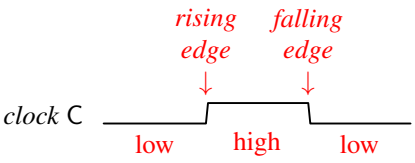
Digital signals are displayed graphically in a *timing diagram*. The horizontal axis is time, increasing towards the right.



The vertical axis shows signal voltages. Each signal's state is displayed as a voltage level, `LOW` (a) or `HIGH` (b). To show that a well-defined signal is present, without specifying its level, it can be drawn with both levels simultaneously (c). To show that a signal is undefined (for outputs) or 'don't care' (for inputs) it can be drawn with both levels and either shading (d) or crosses (e).

A.3 Rising and falling edges

Digital signals change state. A change from one state to the opposite state is called a *transition*. The signal voltage will either rise abruptly from `LOW` to `HIGH`, or fall abruptly from `HIGH` to `LOW`. In the corresponding voltage waveform, this abrupt change is called an *edge* and is usually qualified with its direction. A *rising edge* happens during a transition from `LOW` to `HIGH`, and a *falling edge* during a transition from `HIGH` to `LOW`.

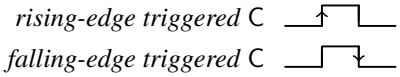


Edges often represent significant instants in time. A regular periodic signal can be used as a *clock*, with activities synchronised to the rising or falling edge of the signal (or both). Clock inputs on devices are often identified in circuit diagrams by drawing a small triangle just inside the device outline.

The edge of an irregular signal can indicate that an asynchronous event has occurred. When the edge occurs, we say the signal is *asserted* to indicate the event is happening. For example, many devices have a RESET input. When RESET is inactive,

the device performs its normal function. When the RESET signal is asserted (changes from inactive to active) the device resets by placing itself in a specified, default state.

Devices that respond to the edges of a signal are called *edge triggered*. They can be *positive edge triggered* (responding to the rising edge) or *negative edge triggered* (responding to the falling edge). A trigger edge is sometimes emphasised in timing diagrams by drawing an arrow on it.



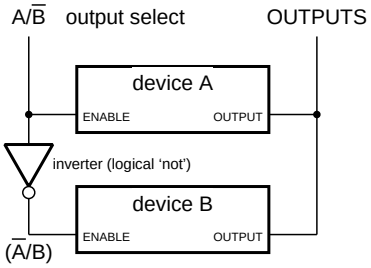
A.4 Some standard logic gates

The common logic operators are augmented with versions having negated outputs (e.g., nand, nor, and xnor).

B A	buffer	not	and	nand	or	xor
0 0	0	1	0	1	0	0
0 1	1	0	0	1	1	1
1 0			0	1	1	1
1 1			1	0	1	0

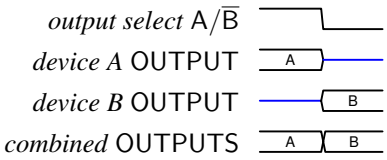
A.5 Tri-state outputs

Tri-state (or 3-state) logic allows multiple output signals to share a single electrical connection. They are used whenever multiple outputs need to be connected to a single input. Tri-state outputs can have three states: the usual LOW and HIGH, plus a third *high impedance* (or high-Z) state which disconnects the output from the internal circuitry of the device. When an output is in the high-Z state, the voltage level of the signal that it is connected to can be set by some other output that shares the same connection.



In timing diagrams, a tri-state signal that is disabled (high-impedance) is often drawn as a line half way between LOW and HIGH.

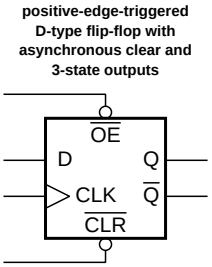
For example, two devices, A and B, have their tri-state OUTPUTs connected to form a single combined OUTPUTS signal. Each device has an ENABLE input. When the ENABLE input is active, the device's OUTPUT will provide a LOW or HIGH voltage level. When the ENABLE input is inactive, the device's OUTPUT will be in a high-impedance state that allows another device to determine the state of the combined OUTPUTS. An inverter between the two ENABLE ensures only one of the devices is enabled at a time.

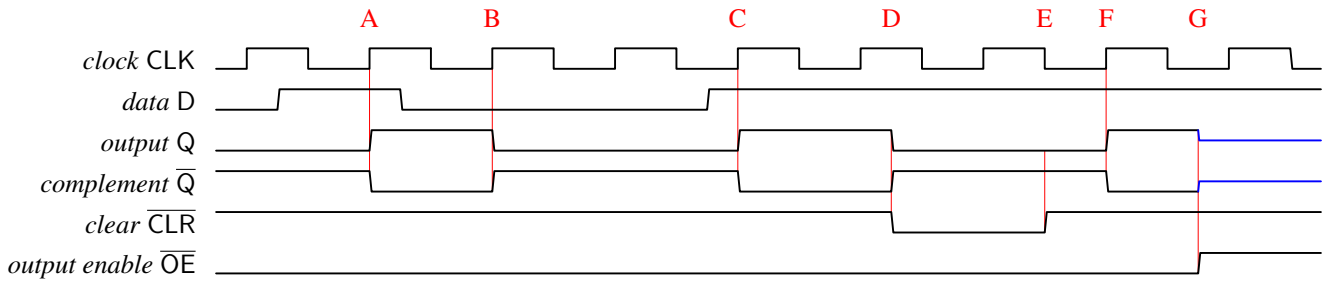


In the first half of the timing diagram device A is enabled and it determines the state of the combined OUTPUTS; device B is disabled and its output has no effect on the circuit. In the second half device B is enabled and it determines the state of the combined OUTPUTS; device A is disabled and its output has no effect on the circuit.

A.6 A comprehensive example device: the D-type flip-flop

A D-type flip-flop is an edge-triggered device that copies the state of its input pin D to the output pin Q whenever it is triggered by the rising edge of its clock input CLK. D-type flip-flops often have a 'clear' (reset) input CLR that sets Q to LOW independently of D and CLK. Some have an output enable OE that isolates the output pin Q from the internal circuitry when inactive. Some have a complementary output $\overline{\text{Q}}$ that is always the inverse of Q. An example of an edge-triggered D-type flip-flop with active-low clear and output enable pins is shown on the right. A timing diagram illustrating its operation is shown below.



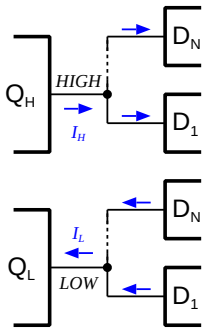


At A, B and C, the rising edge of CLK triggers the flip-flop to copy D to Q. At D, $\overline{\text{CLR}}$ becomes active (low) and resets the device, forcing the output Q low. The output remains low, regardless of D and CLK, until $\overline{\text{CLR}}$ becomes inactive at E. At F, D can be copied to Q again as usual. At G, $\overline{\text{OE}}$ becomes inactive (high) causing the outputs to be isolated from the internal circuitry.

A.7 Fan-out

The *fan-out* of an output pin is the maximum number of inputs to which it can be safely connected.

One output can be connected to one or more inputs on other devices. When the output is high it *sources* (provides) current and the inputs *sink* current. When the output is low it *sinks* current and the inputs must *source* current.



Each output has a maximum current it can source or sink, and each input has a maximum current it will sink or source when driven. For reliable operation (voltage levels having well-defined logical meaning) an output must not source or sink more current than its rated maximum. (These maximums are specified in the data sheet for the device.)

For an output to safely drive N inputs, two conditions must therefore be satisfied:

- output `HIGH`: the maximum source current of the output must be larger than the sum of the maximum sink currents of all the inputs to which it is connected; and
- output `LOW`: the maximum sink current of the output must be smaller than the sum of the maximum source currents of all the inputs to which it is connected.

For a given circuit configuration, the maximum value of N for which the above constraints are satisfied is called the *fan-out* for the output.

$$I_H >= \sum_{i=1}^{N} \text{sink}(D_i)$$

$$I_L <= \sum_{i=1}^{N} \text{source}(D_i)$$

A.8 Shift registers

A *shift register* is a two-input, N-output device. A single serial data input D is *sampled* every time a rising edge is seen on the clock input CLK. The last N values of D that were sampled are stored and made available on the N outputs Q_0 to Q_{N-1}.

Depending on the shift register, additional inputs and outputs might be available. Some examples include:

CLK								
D	0	0	1	0	0	1	1	0	0
Q_1	0	0	1	0	0	1	1	0	0
Q_2	0	0	0	1	0	0	1	1	0
Q_3	0	0	0	0	1	0	0	1	1
Q_4	0	0	0	0	0	1	0	0	1

(× = 'don't care')

CLK								
RCLK								
D	0	1	0	0	×	1	×	1
Q_1	0	0	0	0	0	0	1	1
Q_2	0	0	0	0	0	0	0	0
Q_3	0	0	0	0	1	1	0	0
Q_4	0	0	0	0	0	0	1	1
Q'_4	0	0	0	0	0	1	1	0

OE (input) is an *output enable*. Whenever OE is active, the outputs are enabled and generate `LOW` or `HIGH` voltages to reflect the state of the stored bit. When OE is inactive, the outputs are disabled and behave as if they are not electrically connected to a voltage source.

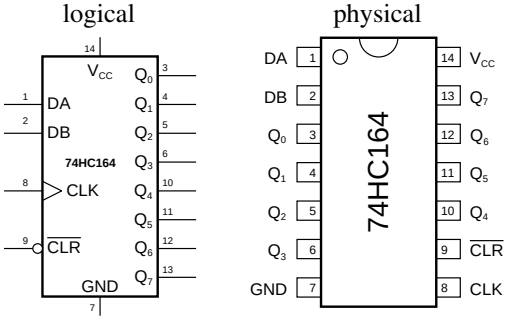
RCLK (input) is the output *register clock*. If this input is available then the values of b_i are not copied to the output pins Q_i when CLK rises. The shift register works as normal, shifting the stored bits b_i one stage each clock cycle, but their values are only copied to the outputs Q_i when RCLK rises. This provides *buffered* output, and allows a long value to be shifted into the register without disturbing the outputs until a rising RCLK causes them to be updated.

Q'_N (output) provides an *unbuffered* copy of Q_N that can be used to cascade two shift registers together, effectively making a shift register with twice as many bits.

A.8.1 The 74HC164 shift register

The 74HC164 is a simple shift register, supporting clock frequencies up to 78 MHz. It has four inputs:

DA	serial data input
DB	serial data input
CLK	serial data clock (positive-edge triggered)
$\overline{\text{CLR}}$	asynchronous clear (active low)



The 74HC164 has eight outputs:

Q_0–Q_7 eight parallel data outputs

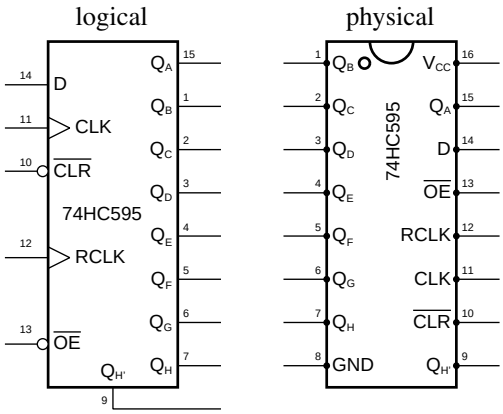
The two data inputs, DA and DB are 'and'ed together internally before becoming the input to the first shift register stage. Both must therefore be `HIGH` for a '1' to be shifted into the register.

Multiple 74HC164s can be *cascaded* to form an arbitrarily-long shift register, by connecting the Q_7 output of each device to the D input of the next and running them all from the same clock.

A.8.2 The 74HC595 shift register

The 74HC595 is a typical (and popular) shift register, supporting clock frequencies up to 20 MHz. It has five inputs:

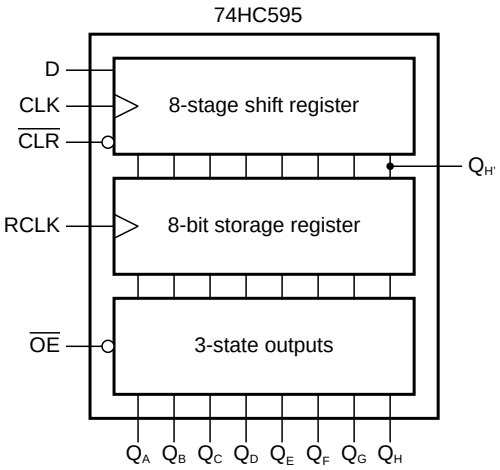
D serial data input
CLK serial data clock (positive-edge triggered)
RCLK output register clock (positive-edge triggered)
$\overline{\text{CLR}}$ asynchronous clear (active low)
$\overline{\text{OE}}$ output enable (active low)

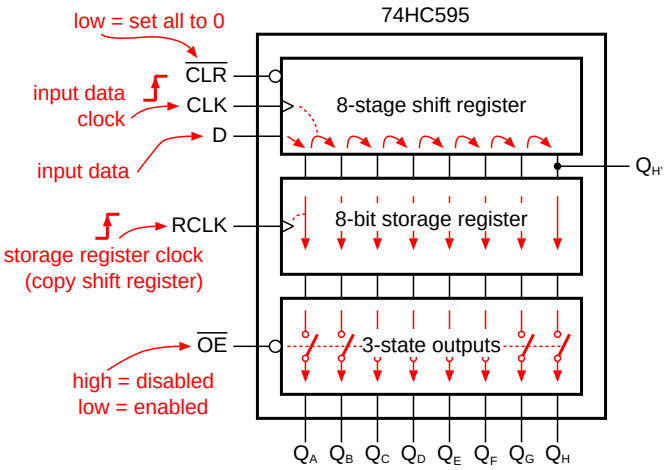


The 74HC595 has nine outputs:

Q_A–Q_H eight parallel data outputs
Q'_H serial data output (can be used to cascade several shift registers together)

The device functions as a normal shift register, except that the eight shift register stages are not connected directly to the outputs. Instead they are connected to an internal 8-bit parallel output register, with its own clock input. The bits stored in the shift register itself are clocked into the output register on the rising edge of the RCLK input.





The device can be operated with CLK and RCLK connected together, which causes the shift register to be copied to the outputs at the same time as D is copied into the first stage of the shift register. However, this introduces an additional clock cycle's delay between the data being sampled at D and it appearing at the first output bit.

When the $\overline{\text{OE}}$ input is high, the outputs are high-impedance. The outputs can be connected to other tri-state outputs, from other devices, provided only one of the devices has its output enabled at any given time.