The code works with the **74HC595 shift register** to control the output states, but let's break this down to understand how the **timing, voltages, and microcontroller compatibility** come into play:

1. How the Code "Knows" About Voltages:

- The **74HC595** is designed to operate within a specific voltage range (typically 2V to 6V for most chips).
- The microcontroller (e.g., Arduino) supplies the necessary voltage levels to the **74HC595** through its GPIO pins.
- When you write `digitalWrite(DS, HIGH)` or `digitalWrite(DS, LOW)`, the microcontroller sends a signal:
    - `HIGH` = Output voltage (usually 5V or 3.3V depending on the microcontroller logic level).
    - `LOW` = 0V.

The **shift register** interprets these signals as logic `1` or `0` and updates its internal shift register accordingly.

2. How Timing Works:

Timing is controlled using the **clock pin** (`SHcp` in your code). Here's how:

- The **clock pin** tells the shift register when to read the data on the **data pin** (`DS`).
- Each rising edge (transition from LOW to HIGH) of the clock tells the **74HC595** to shift its internal data register by one bit and read the next bit on the **data pin**.
- The timing of these transitions is determined by your microcontroller when you call `shiftOut()`.

3. `shiftOut()` Function:

This function handles the timing for you:

- It generates the proper sequence of signals on the `DS` (data pin) and `SHcp` (clock pin).
- It shifts out 8 bits of data (one at a time), with each bit synchronized to a clock pulse.

Example for `B00000011`:

1. **First Clock Pulse:**
   - `DS = 1` (logic HIGH for Q0).
   - Clock goes HIGH, shifting `1` into the first register bit.
2. **Second Clock Pulse:**
   - `DS = 1` (logic HIGH for Q1).
   - Clock goes HIGH again, shifting `1` into the second register bit.
3. **Remaining Clock Pulses:**
   - `DS = 0` (logic LOW for Q2 to Q7).
   - Clock pulses shift in the remaining `0`s.

4. When Data Is Sent to Outputs:

- After all 8 bits are shifted into the internal register, the `STcp` (latch pin) is pulsed HIGH to transfer the data from the internal shift register to the output pins (`Q0-Q7`).

5. Microcontroller Compatibility:

The same pattern (`dataArray`) should work with different microcontrollers, provided:

1. **Voltage Levels Match**:
   - Ensure the microcontroller's output voltage is compatible with the **74HC595** logic levels (e.g., 5V or 3.3V).
2. **Clock Signal is Properly Generated**:
   - The clock timing (frequency) must meet the **74HC595's** requirements.
   - `shiftOut()` handles this for most microcontrollers, but you can use manual bit-banging if needed.
3. **Pin Mapping is Correct**:
   - The code should correctly define the microcontroller pins connected to the **DS**, **SHcp**, and **STcp** pins of the **74HC595**.

6. Role of the Clock:

The **clock pin** is crucial because it synchronizes the data transfer:

- The shift register doesn't "know" when the next bit is coming—it relies on the clock signal to shift bits correctly.
- Without the clock, the **74HC595** cannot determine when to update its internal data.

Summary:

- **Voltages:** The microcontroller sets the HIGH/LOW levels (logic 1/0) to control the shift register.
- **Timing:** The clock (`SHcp`) tells the shift register when to shift and read the data.
- **Compatibility:** The same pattern can work for different microcontrollers if the voltage levels, clock timing, and pin connections are correct.