Proving the Concept: Manually Bit Banging a MAX7219

Yilan Liu

Last updated: March 16, 2025

Initial Motivation

The MAX7219 is a popular choice for driving seven segment LED displays, so it was only a matter of time before I got my hands on one for one of my projects. A quick peruse through example projects on the web revealed to me that using a microcontroller is the most common and straightforward way of using the MAX7219. In fact, even the datasheeet for the MAX7219 affirms this - in their "Typical Application Circuit" figure, they include a microcontroller/microprocessor in their design:

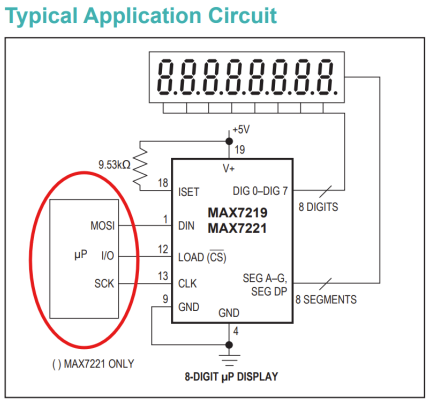


Figure 1: "Typical Application Circuit" figure of the MAX7219 datasheet, pg. 1

However, after reading the MAX7219 datasheet in detail, I had a thought - is a microcontroller really needed for communicating to the chip? This was the driving question and inspiration for this project.

Communication with the MAX7219

Communication with the MAX7219 is quite simple: there are fourteen 8-bit registers, and depending on the data inside those registers, the LED display will display the segment(s) of your choice. To change the data inside any particular register, one must input a 16-bit data packet in serial fashion into the DIN pin of the chip. The first four bits (D15-D12) are always don't care values. The next four bits (D11-D8) are the address of the target register. The last eight bits (D7-D0) are the data you wish to load into the target register.

Each data bit is latched onto the rising edge of a clock pulse inputted into the CLK pin, and after the sixteenth data bit is latched, tying the LOAD pin high will send the data packet to the chip.

Here is a screengrab from the datasheet that illustrates this process:

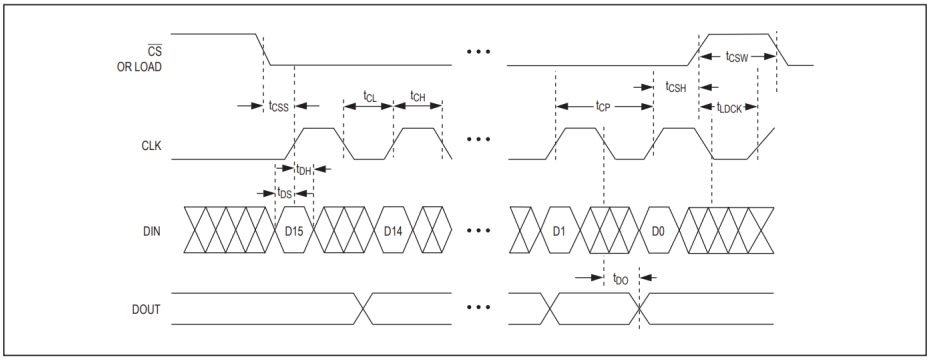


Figure 1. Timing Diagram

Table 1. Serial-Data Format (16 Bits)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB	DATA						LSB

Figure 2: The timing diagram and serial-data format of the MAX7219

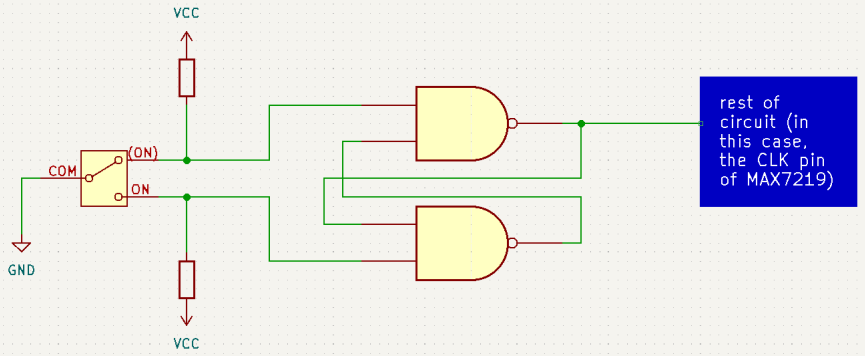
Proving the Concept: The Straightforward Approach

For this project, I decided to make the objective simple: change the data inside the Shutdown Mode register in order bring the MAX7219 into Normal Operation, and do this without a microcontroller. On initial power-up, all registers are cleared and the MAX7219 automatically enters shutdown mode. The idea is that one would program the MAX7219 (i.e. send all the data to tell it which

segments of which digits to turn on, set the intensity, etc.) before turning off shutdown mode. To turn off shutdown mode, one would need to load the following 16-bit data packet into the MAX7219: $\times \times \times \times 1100 \times \times \times \times \times \times \times 1$.

The approach I took to do so was as follows:

1. I wired the output of an SPDT switch to the CLK pin of the MAX7219. This is so that I could manually control the speed of each clock pulse. Please note that in this case, it is critical that the switch does not emit any false triggers such that would cause some data bit to get latched in erroneously. Therefore, I debounced my SPDT switch using an SR latch as shown in the figure below:

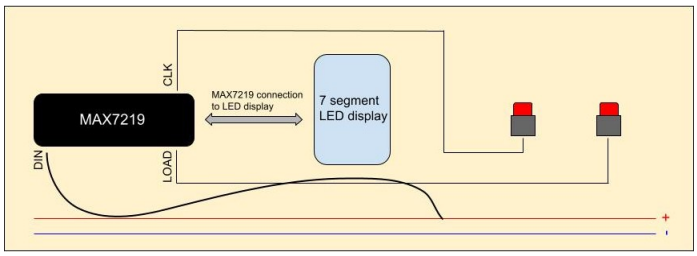


2. I connected a jumper wire with one end connected to the DIN pin of the MAX7219 and the other end either connected to ground or Vcc. The idea was that depending on which data bit I wanted to latch onto the rising edge of the clock pulse, I would connect the jumper wire to either the ground of Vcc bus of the breadboard, and then push down on the clock switch to latch the data bit.

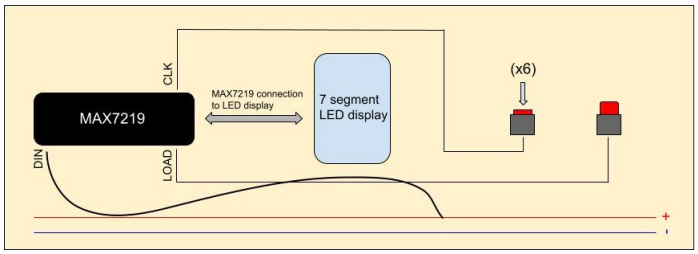
3. I connected a switch to the LOAD pin of the MAX7219. Once all sixteen data bits were latched, I would press the switch to load the packet into the chip.

Putting the above into practice, here's what the precise equence of events looked like in order to bring the MAX7219 out of shutdown mode:

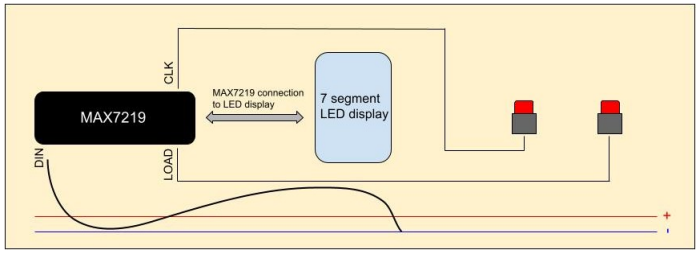
1. Connect DIN to Vcc via the jumper wire.



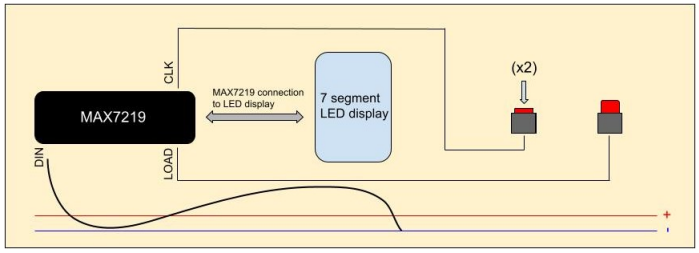
2. Press the clock pulse button six times.



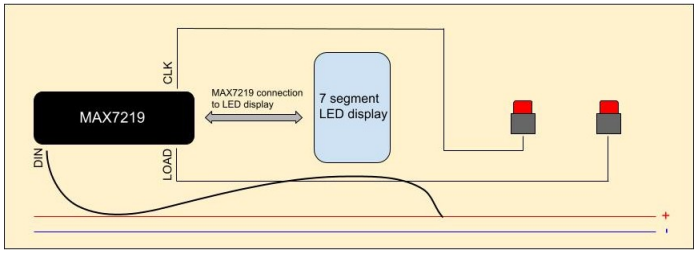
3. Connect DIN to ground via the jumper wire.



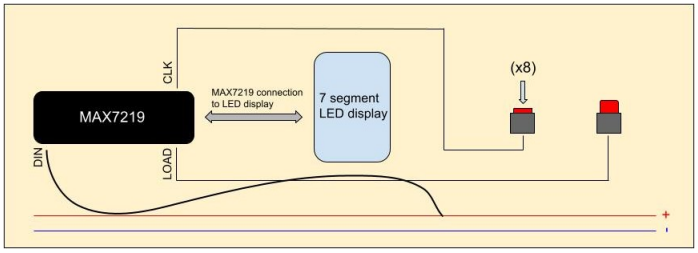
4. Press the clock pulse button twice.



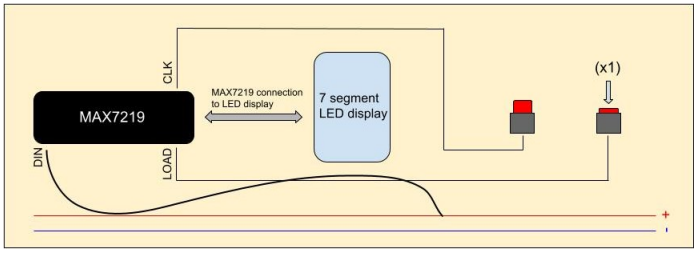
5. Connect DIN to Vcc via the jumper wire.



6. Press the clock pulse button eight times.



7. Press the load button once.



Now, you could imagine that such a procedure would be quite tedious: With one hand, I would press the clock pulse switch, and with the other hand, I would need to move the jumper wire back and forth between the ground/power rails depending on which data bit I needed to send. On top of that, I would need to keep track of which clock pulse I'm currently on so as to not mess up the sequence of data bits.

Most importantly, though, this process would need to be repeated every single time the MAX7219 is powered up.

The tedious nature of this process prompted me to wonder: Is there a way to cut down on the manual workload in this process?

Refinining the Process

The first idea I had was to use a Parallel In, Serial Out shift register. With this method, one would only need to wire the data inputs of the shift register to ground/Vcc once during the initial construction of the circuit. After the shift register is set up, one would only need sixteen clock pulses to serially shift out each data bit into the MAX7219.

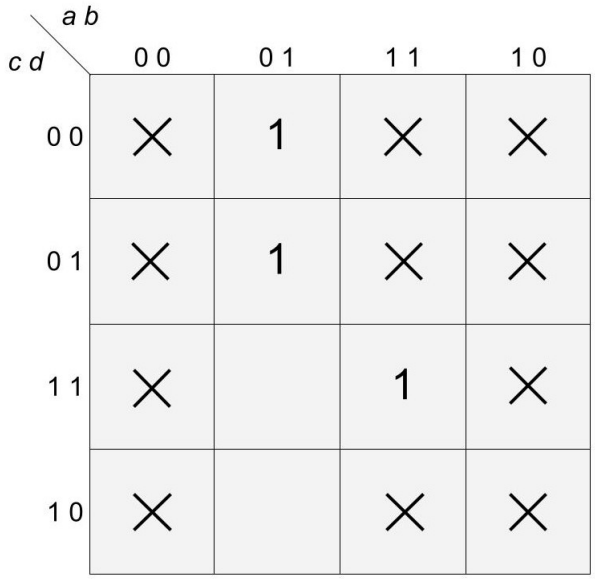
While this idea seemed promising, I couldn't help but feel like the shift register was doing too much of the hard work, thereby trivializing the rest of the process. Thus, I decided to brainstorm more to think of a creative solve to the problem.

After some time, I finally had a thought that I could construct a boolean function that would take some number from 0 through 15 and return either 0 or 1. The number from 0 through 15 would represent which clock pulse we're on, and the output of 0 or 1 would be the data bit we wish to latch on the following clock pulse.

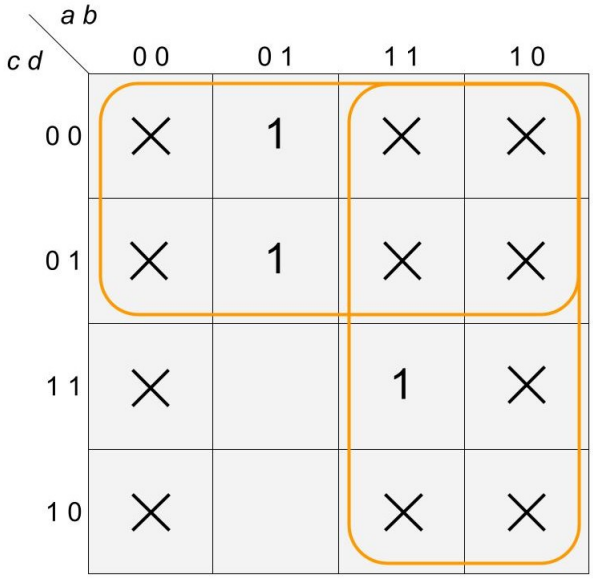
The truth table for the function is as follows:

$abcd$	$f(a,b,c,d)$
0000	$\times$
0001	$\times$
0010	$\times$
0011	$\times$
0100	1
0101	1
0110	0
0111	0
1000	$\times$
1001	$\times$
1010	$\times$
1011	$\times$
1100	$\times$
1101	$\times$
1110	$\times$
1111	1

In the interest of making our combinational circuit representing f as simple (and cheap!) as possible, I decided to use a K-map to find the minimum sum of products (SoP) expression and the minimum product of sums (PoS) expression for f. The K-map to obtain the minimum SoP expression is as follows:



We see that minterm m_5 and its adjacent 1s and $\times$s are covered by the prime implicant c', which makes c' an essential prime implicant. We also see that minterm m_{15} and its adjacent $\times$s are covered by the prime implicant a, which makes a an essential prime implicant as well:



Hence, we have that the minimum SoP expression for f is

$$f = a + c'.$$

Please note that because the minimum SoP expression contains only one addition operation on two literals, this is also the minimum PoS expression for f. The proof of this is as follows: Suppose the converse is true - that there exists a PoS expression for f that is more minimal than $f = a + c'$. Then, this means that such an expression must contain at least one fewer literal (so either $f = a$ or $f = c'$). But note that $f = a$ or $f = c'$ would also be a SoP expression that would be more minimal than the minimum SoP expression $f = a + c'$ found by the K-map, a contradiction. Therefore, we conclude that $f = a + c'$ is both the minimum SoP and PoS expression for f.

With this, the only thing left to do was to figure out how to associate each clock pulse with a number, going in order from 0 through 15. Wait a minute - this sounds exactly like what a counter is supposed to do, and just coincidentally, I had just built a mod-16 counter!

Now, everything that I needed was in place.

Implementing the Refined Circuit

In order to construct $f = a + c'$, one could straightforwardly take a two-input OR chip and NOT chip. However, when I was at this phase of the project, I only had some NAND gates and some NOT gates with me. Therefore, I just used the theorem that

$$\text{NAND}(\text{NOT}(x), \text{NOT}(y)) = \text{OR}(x, y)$$

in order to simulate an OR gate using the NANDs and NOTs that I had. One can verify the above theorem with a truth table:

x	y	NOT(x)	NOT(y)	**NAND(NOT(x), NOT(y))**	**OR(x, y)**
0	0	1	1	**0**	**0**
0	1	1	0	**1**	**1**
1	0	0	1	**1**	**1**
1	1	0	0	**1**	**1**

After the circuit representing f was constructed, I connected the most significant bit of the counter output to a and the third most significant bit of the counter output to c. Then, all that was left to do was to connect the output of the combinational circuit to DIN of the MAX7219 and connect the clock push button to the 4-bit counter.

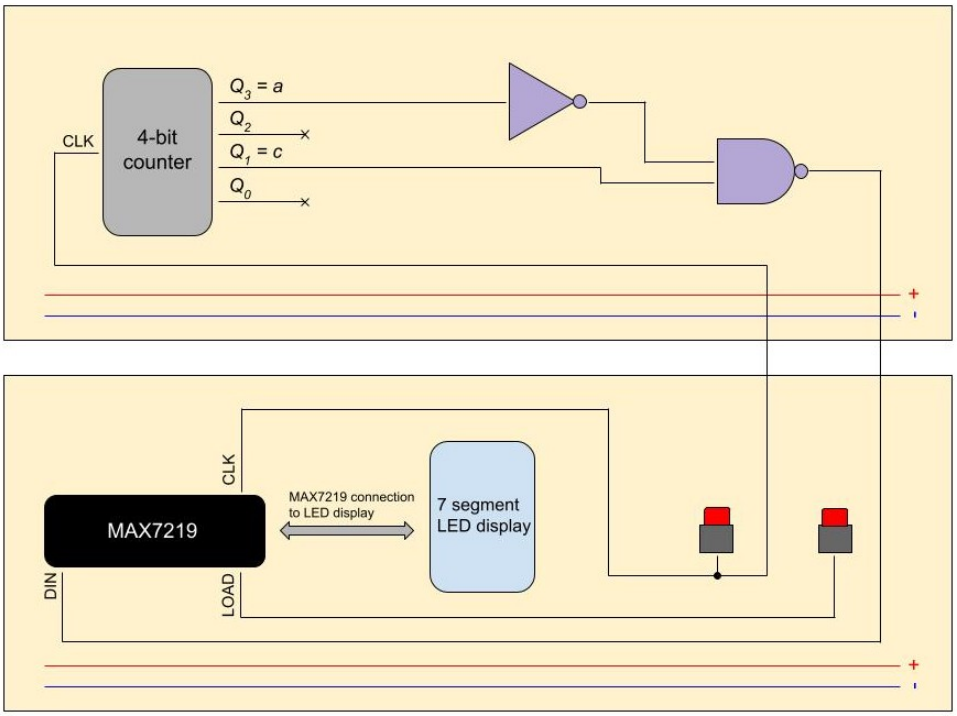
There's only one last caveat that we have to consider: timing. Assuming that both the data bits get latched and the JK flip flops change state on the rising edge of the clock pulse, we now have a race condition on our hands. The moment we press the clock pulse button, what we want is for the data bit to be latched before the JK flip flops progress to the next state - however, we can't

guarantee that this will always happen if the speeds of those two processes are similar.

Thankfully, one could solve this race condition problem a couple of ways. The first way would be to simply delay the propagation of the clock pulse to the JK flip flops, but the only caveat to this solution is that one would have to ensure that the delay period is less than the length of one clock cycle. Another way (and perhaps cleaner solution) would be to put the clock pulse going to the 4-bit counter through a Not gate, effectively shifting the clock pulse phase by 180 degrees.

Yet another solution could be to simply use JK flip flops that are triggered on the falling edge of the clock pulse. This is the solution I ended up going with, as I had coincidentally bought this kind of JK flip flop to start.

With that sorted out, the whole circuit is complete, and we have successfully reduced our workload to just needing to press the clock button sixteen times and the load button once to bring the MAX7219 out of shutdown mode! A simplified diagram of the wiring is below:



Below are also screen grabs from a video showing how the circuit works in action. The multimeter is connected to the DIN pin of the MAX7219, the red LEDs

depict the output of the 4-bit counter, and the green LED is connected to the CLK pin of the MAX7219. Each screen grab was taken on a rising clock edge.