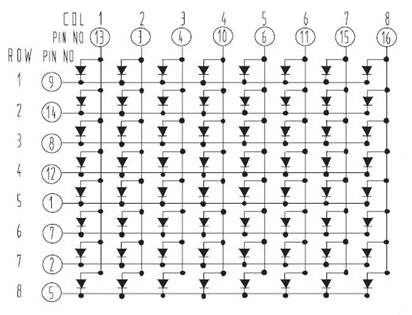
REPORT

iNTERFACING AN 8X8 led DISPLAY WITH ARDUINO USING MAX7219 DISPLAY DRIVER

sree rohith pulipaka, bits pilani, hyderabad campus

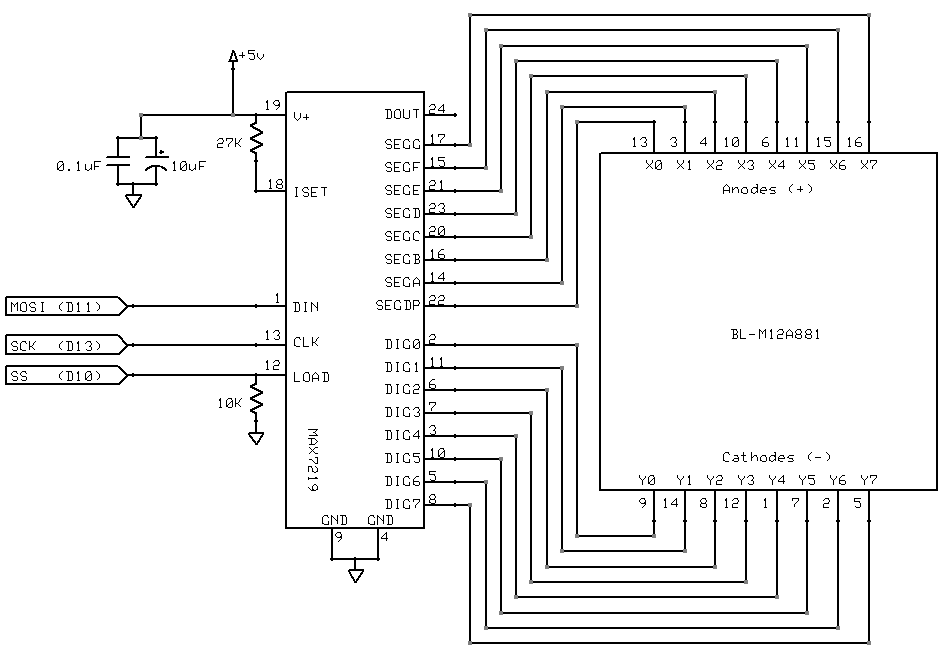
**Introduction**

The 8x8 LED display is an array of LEDs with common terminals which form the rows and columns. For example, as it can be seen from this diagram, row 1 is formed by the common cathodes of all the LEDs from columns 1 through 8. Similarly, Column 1 is formed by the common anodes of all LEDs in rows 1 through 10.

To light up an individual LED, its respective row and column are selected and made HIGH/LOW to forward bias it. For example, LED in row 3, column 2 can be activated by making pin 3 go High and pin 8 go Low.

However, one 8x8 LED matrix requires 8+8 = 16 pins which is very large to be directly connected to a microcontroller like Arduino. That is why, I think, we use a driver IC like MAX7219.

**Connecting with the Arduino**

The hardware that I worked with had the MAX7219 IC already wired to the 8x8 LED display. But if that is not the case, connections can be easily made between the two, as explained later in the document.

If we see the Pin diagram of the IC , we can see three pins named as DIN (pin 1), CLK (pin 13), LOAD(pin 12). These three pins are the pins of interest while connecting to the Arduino apart from Vcc and Gnd pins. These pins are standard pins for any device that communicates through SPI communication (Serial Peripheral Interface).

In SPI, there is a master device (here the Arduino) and multiple slaves (here only one slave that is the MAX7219 IC). So the Arduino sends data to the IC on the DIN pin. It is called as MOSI – Master Out, Slave In. As the name of the communication suggests, data is sent in a serial fashion, bit by bit. The SS pin is the slave select pin which is only useful if there are multiple slaves (for example, more than one MAX7219 IC).

**Initial Approach**

Initially, I used a predefined library called ‘LedControl’ to form patterns on the matrix. The library is very simple to use. Similar to any other library, we need to create an instance of the class ‘LedControl’ and then we can use the inbuilt functions with the object.

For example, the I used the ‘setIntensity()’ function to set the intensity of all LEDS at half the maximum (0 -15 are the allowed values and I set it at 8). Also, we need to make sure that the LED matrix is not in shutdown mode. In this mode, data can still be transmitted to the MAX7219 IC, but LEDs will not light up. So, use the ‘shutdown ()’ function to disable this mode.

Individual rows or columns can be turned ON or OFF with the ‘setRow()’ or the ‘setColumn()’ function respectively. It takes three arguments: first- which 8x8 array we want to address (this parameter is only relevant when we are using more than one 8x8 array. But if we are using only one, it is assigned as zero.) The second argument is the row/column number. The third argument is the Value. For example, the value of ‘0xFF’ will light up all the LEDs. If we input ‘0xAA’, alternate LEDs in that particular row/column will light up.

In this way, various patterns can be formed on the matrix. In case, there is a need for individual LEDs to be controlled, there is another command called ‘setLed()’ function. But that is more tedious.

**Other possible approaches**

***Another Library***

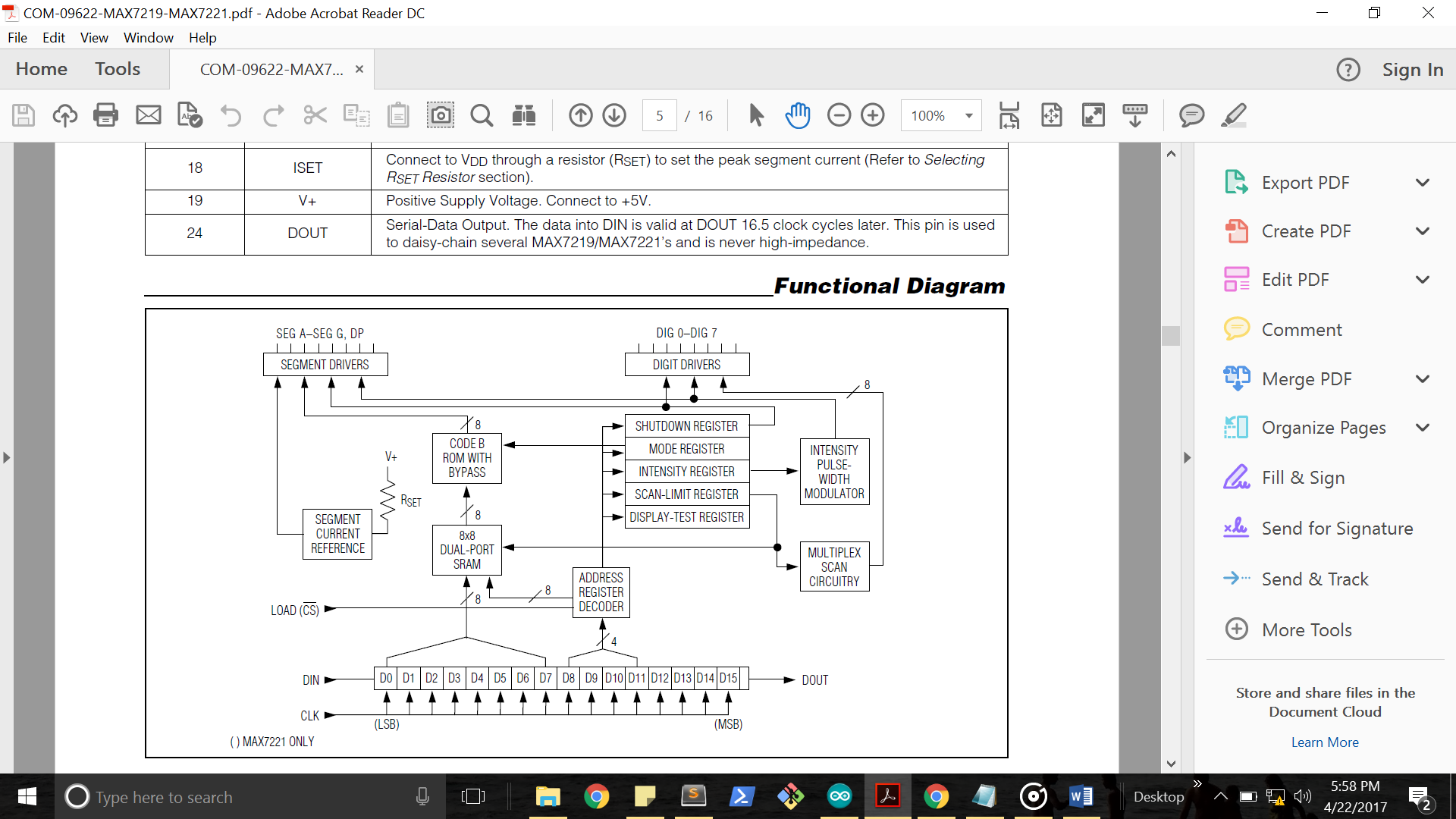
There is another library called as ‘MaxMatrix’ but the idea is similar to the one described above: using functions predefined in the Library, we can form patterns on the LED matrix.

----------------------------

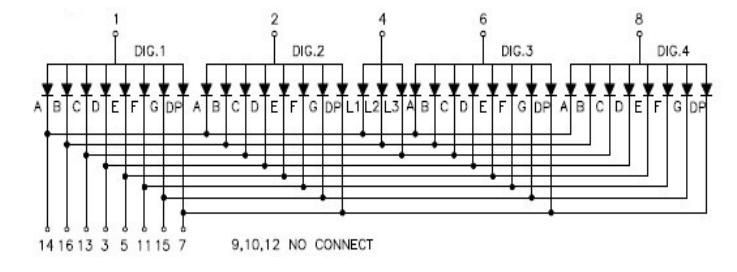
**Note:** In any approach, it is advisable to use pins other than the built in TX and RX pins on the Arduino for Serial communication. This is to avoid bus contention which means that this step prevents interference of serial communication between the USB-to Serial Chip (on the Arduino) and the serial communication with the MAX7219 IC.

I implemented this using another Library called as ‘SoftwareSerial’.

**A more down-to earth approach: Addressing the registers in MAX7219 directly**

To understand this approach, we need to look at MAX7219 at the register level. Here is a schematic of MAX7219 registers. The starting point in understanding this figure is the understanding of the SEG A – SEG G Drivers (top left in the diagram) and the DIGIT Drivers (in the top right) and the internal 16-bit Din Register (bottom of the figure). Relating it back to the Arduino, this 16-bit register is the register that acts as the slave register.

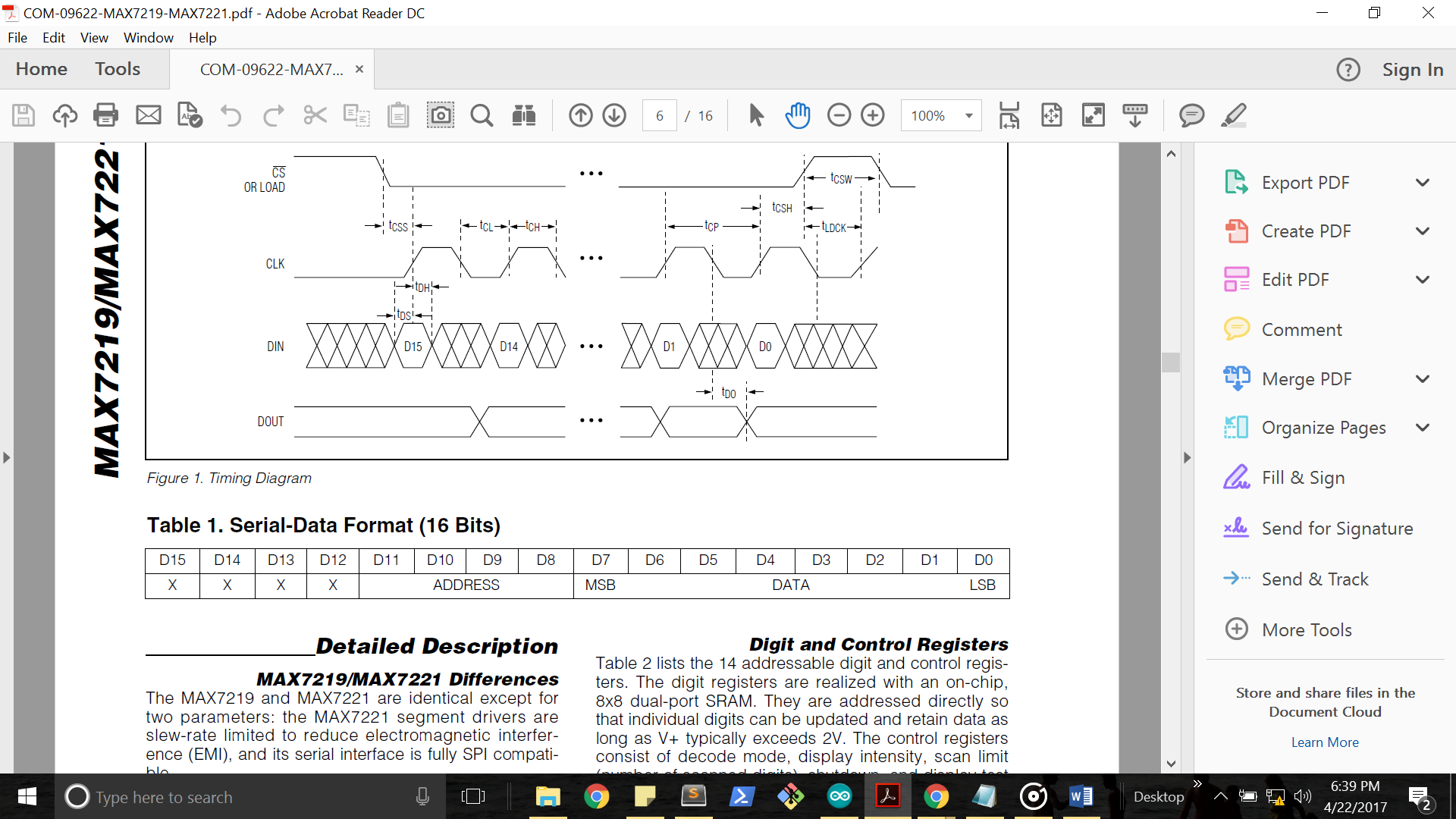
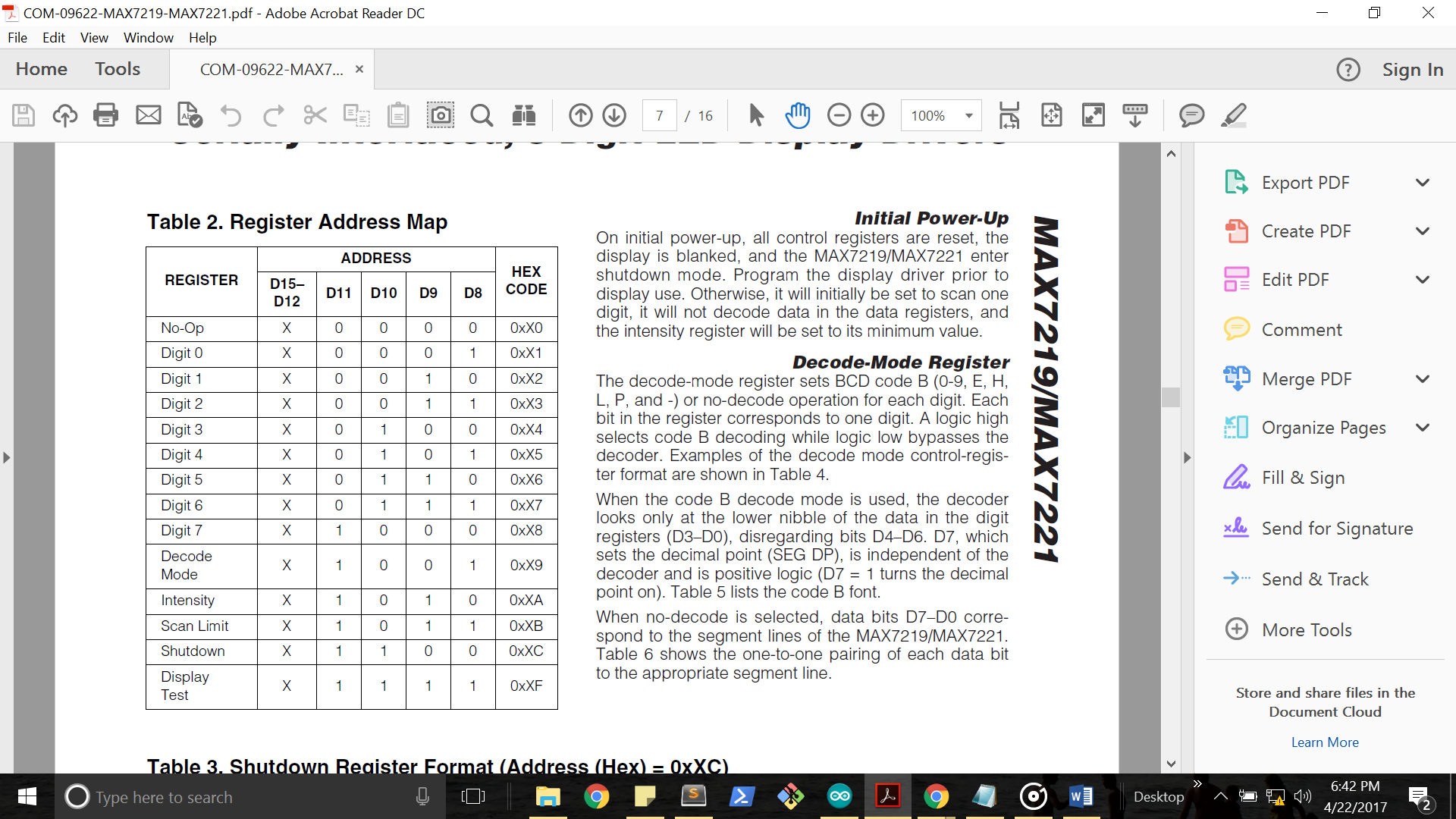
This schematic primarily focuses on using the MAX7219 IC as a multi-digit (maximum: eight) seven-segment display driver. Hence the terminology ‘SEG ’, ‘Digit’ etc. Once we understand this from this perspective, I shall then explain how this is applicable to the 8x8 LED matrix also.

To begin with, I shall start with the description of a multi-digit seven segment display:

Here, I have included a picture of the diode connections of a 4-digit 7-segment display. It can be inferred from the diagram that the seven segment LEDs (A through G) and DP (Decimal Point) LED are grouped into four ‘Digits’ (Dig.1 to Dig. 4).

Inside each digit( for example, Dig. 1), the seven segment LEDs have the same Anode (positive terminal), which is the “Dig. 1” pin ( they can also be in the common cathode configuration). This means that the first digit can be activated by activating the Dig.1 pin. We can also see that the seven segment LEDs of one digit are multiplexed with the seven segment LEDs of the other three digits. This means that the individual segments each digit share the same pins.

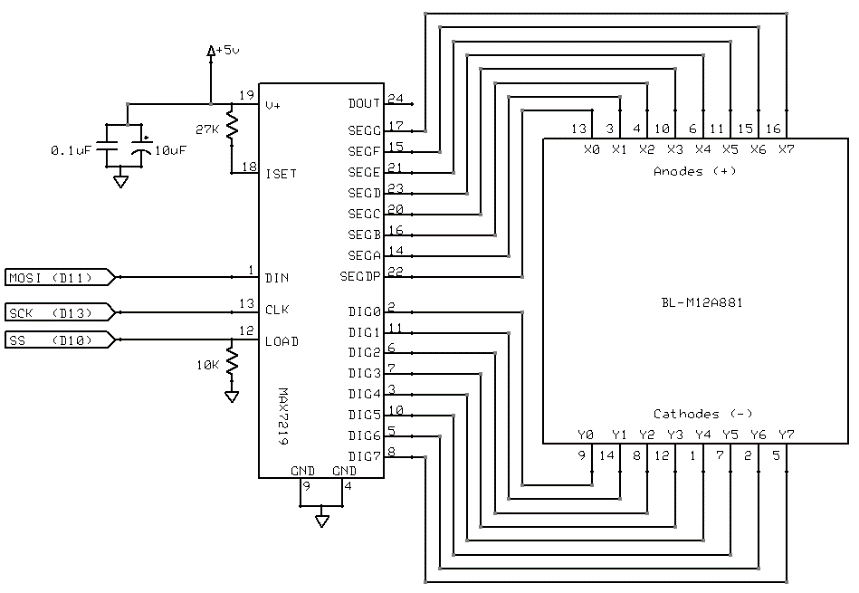
To summarize, let us take an example. If we want to display the number “1” one digit 2, segment pins SEG B and SEG C should go low and all the others segment pins high. Also the ‘Dig. 2’ pin must go High. Because of this mechanism, in actuality it is not possible to selectively display the Segments on each digit. For example, we cannot have Segment A go high in Digit 1 and go low in Digit two at the same time. This means that in actuality it is not possible to display more than one combination of the seven SEG pins. So we need to display different combinations of the Segments A through G after a small time lag. But the IC will generate a time lag that is incomprehensible to the human eye. Hence, we tend to see all the digits glowing at once.

After understanding the purpose of the Segment and the Digit drivers, we shall move onto the in-built 16 bit register. The 8 most significant bits determine the register in the IC that is being addressed .The lower 8 bits determine the value that is given to that particular register.

In the upper 8 bits, the highest nibble is put in the don’t care condition and the lower nibble determines the address of the digit and the control registers. The exact address of each register is given in a table in the datasheet. The table is replicated here. Using this table, we can write the program on the arduino.

If we select the Decode Mode (0x09), the IC will then act as a BCD to seven segment decoder ( SEG A through G) and subsequently numbers can be displayed. But for our purpose of using it with the 8x8 array, we probably will not use this mode.

After this understanding, understanding the 8x8 LED matrix connection is very simple:

First of all, let us see how to connect the IC to the LED array: From the adjacent schematic, we can see how innovatively we have adapted the IC to be useful for driving the 8x8 LED array.

The Digit Driver pins now control the rows and the Segment Pins control the Columns. We will not be using the IC in the decode mode, so the lower byte in the internal 16-bit register now corresponds to the Segment lines.

So, to select a particular LED on the matrix, we need to make its corresponding Digit and Segment Pins active.

Additionally, we need to use the SPI library to commmunicate each and every command to the MAX7219 IC from the Arduino. Also, we need to understand that first the SS pin should go LOW (because it is active low) after which we send the two bytes. Then, we drive the LOAD/SS Pin HIGH so that the 16-bit value in the internal value is latched into either the digit or control registers. This is the general formal in which we execute commands. I provide the skeleton code here below:

SPI.begin();

SPI.setBitOrder(MSBFIRST);

digitalWrite(SS\_PIN, LOW);

SPI.transfer(address); // Put the upper 8 bits here

SPI.transfer(value); // Put the lower 8 bits here

digitalWrite(SS\_PIN, HIGH);

Based on the patterns the user desires, custom fucntions can be built but the basic structure of communicating the 16-bits through SPI from the Arduino to the MAX7219 chip remains the same as above.

In this way, the alternative method I described above is more educational yet more tedious.