UMONS
Université de Mons

Faculté Polytechnique

Hardware/Software Platforms

Master 1 civil engineering 'Multimedia and Telecommunications'

Alexandre DE BRAUWER
Kokou KOMEDJA
Murathan YILMAZ

POLYTECH MONS

Under the supervision of Mr. Carlos Alberto VALDERRAMA

June 2019

pôle hainuyer

Table des matières

Introduction

In this written tutorial, you will learn step by step to use a fpga board (DE1-SOC) in order to control an 8x8 led Matrix. It will show you how to code in vhdl and how to assemble the inputs and the outputs who match the specifications of our code. In the end, you will be able to display whatever you want a letter, a number, an emoji,... on the LED matrix.

Chapitre 1

Hardware/Software

1.1 Hardware

The materials needed for this application are

1. a FPGA (DE1-SOC see fig.1.1), who allows us to drive the LED matrix. This board contains some buttons, switch and GPIO inputs. For our goal, we will just use 5 PINS who are VCC, GND, GPIO(1), GPIO(3) and GPIO(5). The three GPIO stand for DIN (data input), CS (chip select) and CLK (clock) who are the signal needed to control the LED matrix.

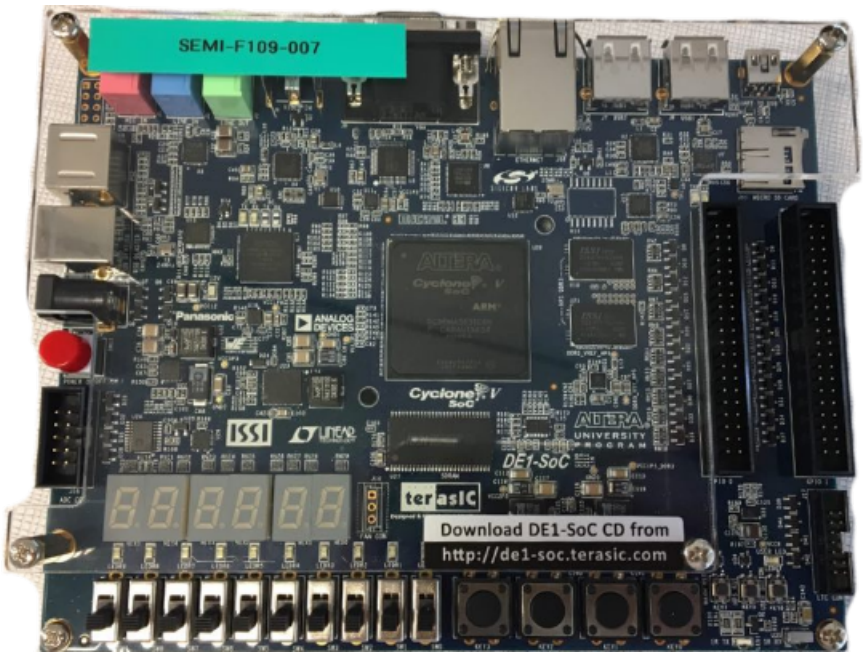

Fig. 1.1 : DE1-SoC Board

2. the LED matrix and the MAX7219 as you can see on fig 1.2. The five inputs that we talked about before are visible in the figure. From these entries, we will connect the DE1-SOC to display what we want on the LED matrix.

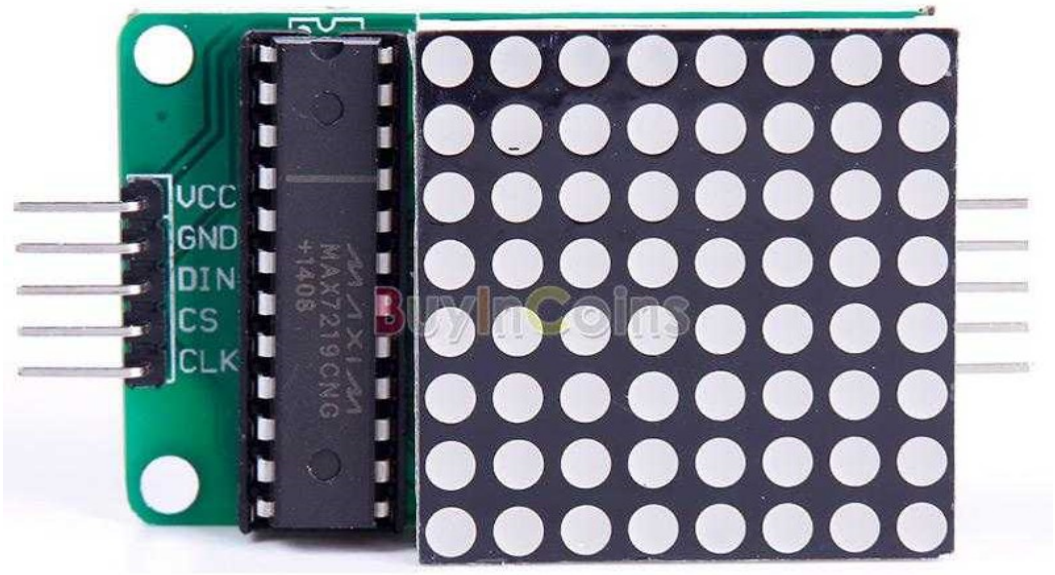

FIG. 1.2 : LED matrix

1.2 Software

In this section, we need to generate the signals (DIN, CS and CLK) in order to send them to the matrix. In fig 1.3, we can see the PIN configuration of the MAX7219 and for

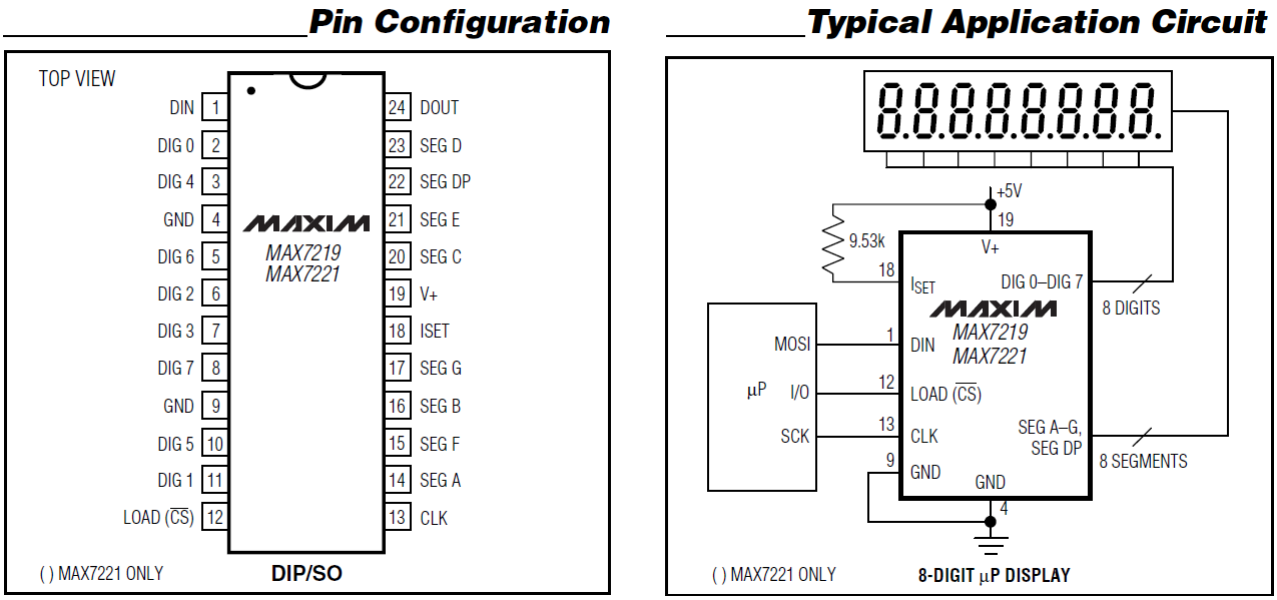

FIG. 1.3 : Matrix controlling

our application it is an 8x8 LED Matrix. We'll use a spi driver to generate those three signal that will drive the MAX. Following this, we'll see what is the format required of the bits that we'll send to the MAX. We use a 16 bits format. As you can see on fig 1.4,

we have 8 bits for the data, 4 for the address to know which LED or line of LED and so on that we want to switch or not. There are 4 other bits, used for other features. We can also see the profile of the signals needed (DIN, CS and CLK) to drive the MAX7219.

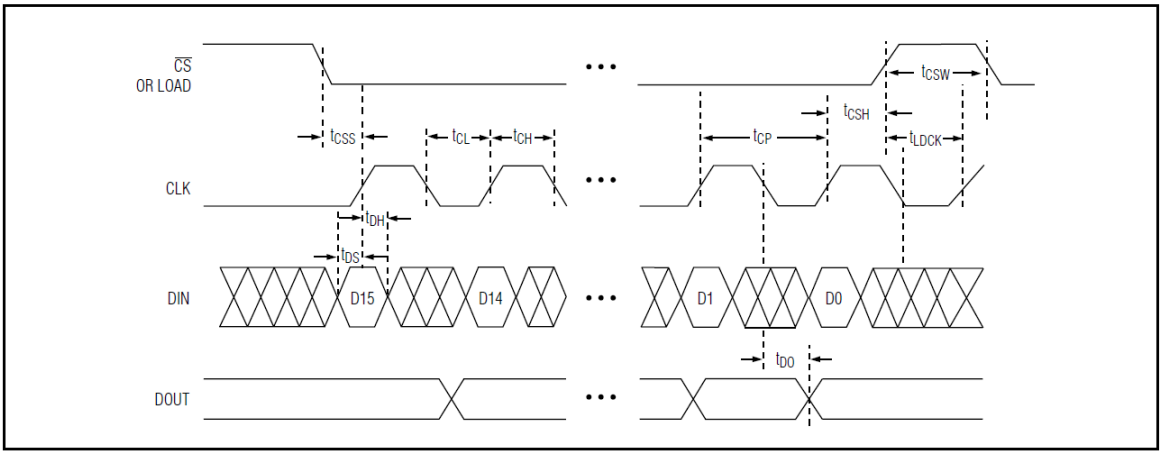

Figure 1. Timing Diagram

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

FIG. 1.4 : Data format

1.2.1 Codes

Now we will use the Quartus altera (15.0 version) environment to construct a vhdl code allowing to match all requirements : generate signals needed (DIN, CS and CLK), associate the right pins (GPIOs,...), ...

First, we'll show the main libraries used and their utilities. We have 3 files for this :

1. Virtual clock : Used as internal clock for some processus.

2. Fonts : Define how to code the numbers between 0 to 9. It's a simplification in the manipulation of the LED to not code every time all the bits when we want to display a number. Kind of preset of the bit

3. Types : Used to create a state machine with 4 states who are initialize, ready, execute and busy. It helps us to know in which state of the cycle we are. You will see later what's happening in each state.

   Before coding the main application (counting from 1 to 99), we need to map the DE1-SOC Pins with the input that we need for the MAX (DIN, CS and CLK). The figure 1.5 show the way to do it. You can also find the file in the Github.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library utils;
use utils.types.all;
use utils.fonts.all;

-- Top-Level entity
entity muro is
    PORT (
        CLOCK_50 : in std_logic;

        GPIO_0 : OUT STD_LOGIC_VECTOR(5 downto 1)

    );
END entity;

ARCHITECTURE muro_arch OF muro IS

   COMPONENT LEDMatrix is
   PORT (
        CLOCK_50 : IN STD_LOGIC;
        LED_DIN,
        LED_CS,
        LED_CLK  : OUT STD_LOGIC

       );
   END COMPONENT;

   constant clk_period : time := 100 ns;
BEGIN

i1 : LEDMatrix
   PORT MAP (
-- list connections between master ports and signals
      CLOCK_50 => CLOCK_50 ,
      LED_DIN => GPIO_0(1),
      LED_CS => GPIO_0(3),
      LED_CLK =>  GPIO_0(5)
   );
END muro_arch;
```

FIG. 1.5 : Mapping

4. The spi_master file helps to control the channel between the DE1-SOC and the MAX. All the control process about master and slave communications, using a virtual clock, are defined in this code. In fig. 1.3, the DIN of the MAX is linked to the MOSI (Master out Slave in), the CS to I/O of the spi-driver and the clock of the MAX is linked with the internal clock of the spi-driver. It also use the state machine.

5. Then, we have the max7219 code to generate the differents signals for the LED according to all requirements of state machine and the clock. There is, also, a little process to configure the LED.

6. To finish, we have the main code who will update the display and prepare the counter mechanism. And the process of the counter itself is implemented.

Chapitre 2

Test

2.1 Code

In the fig. 2.1, we have a code allowing to run the max7219 and have the right signal of output to the led matrix.

```
ARCHITECTURE max_arch OF max_TB IS
   -- constants
   -- signals
   SIGNAL CLOCK_50 : STD_LOGIC;
   SIGNAL virt_clk : STD_LOGIC;
   SIGNAL LED_CLK : STD_LOGIC;
   SIGNAL LED_CS : STD_LOGIC;
   SIGNAL LED_DIN : STD_LOGIC;
   signal state   : types;
   signal input           : STD_LOGIC_VECTOR(15 downto 0) := (1 =>'1', 2 => '1', 8 => '1', 9 => '1', 10 => '1', 11 => '1', others => '0');
   signal  run            : STD_LOGIC := '1';
   COMPONENT max is
        PORT (
            CLOCK_50        : IN STD_LOGIC;
            input           : IN STD_LOGIC_VECTOR(15 downto 0);
            run             : IN STD_LOGIC := '0';
            state           : BUFFER types := initialize;
            virt_clk        : IN STD_LOGIC := '0';
            CLK, DIN, CS    : OUT STD_LOGIC
        );
   END COMPONENT;
   constant clk_period : time := 100 ns;
BEGIN
   vclock : entity work.virtual_clock PORT MAP (CLOCK_50 => CLOCK_50, virt_clk => virt_clk);
   i1 : max
   PORT MAP (
-- list connections between master ports and signals
      CLOCK_50 => CLOCK_50,
      input    => input,
      run      => run,
      virt_clk => virt_clk,
      state    => state,
      CLK => LED_CLK,
      DIN => LED_DIN,
      CS =>  LED_CS
   );
   init : PROCESS
   -- variable declarations
   BEGIN
      CLOCK_50 <= '1';
         wait for clk_period/2;
      CLOCK_50 <= '0';
         wait for clk_period/2; -- code executes for every event on sensitivity list
   END PROCESS init;
END max_arch;
```

FIG. 2.1 : Max7219 Test Bench

2.2 Simulation

We can see on the fig2.2 the result of the simulation test bench.

1. We have the right clock and virtual clock.

2. The input (signal input) can be modified to see the bits we want to transfer.

3. The machine state (signal state) indicate the timing of data transfer, when busy the data is transferred and when ready the data can be sent.

4. Also the signals LED_DIN, LED_CLK, LED_CS have the features we wanted because the data is transferred at the right moment of LED_CLK and LED_CS.

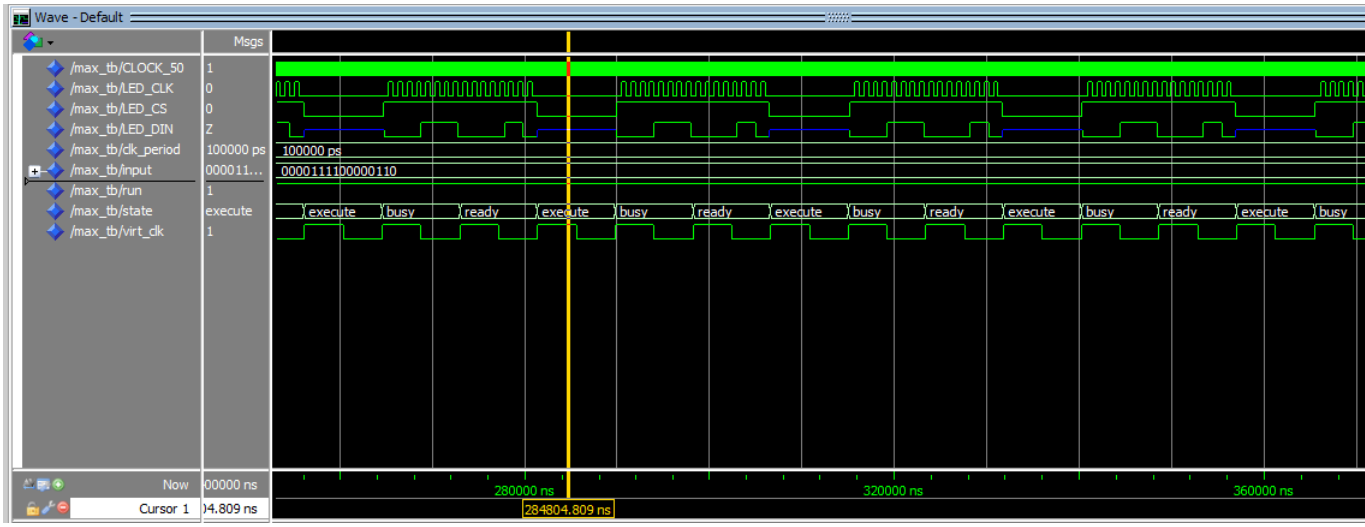

FIG. 2.2 : Simulation

Chapitre 3

The code on the board

3.1 Flashing the files in the DE1-SOC board

3.1.1 Configuration and connections

Now after compilation we will send the application on the Altera FPGA board, We first check the configuration mode of the board relying on the data sheet of the board. For our application The FPGA should be set to AS x1 mode i.e. MSEL[4..0] = 10010 to use the quad Flash as a FPGA configuration device.

We do all pins connections : correct GPIOs, GND, VCC links between the DE1-SOC board and the max7219 + 8x8 LEDmatrix board. Then the connections of usb, Uart, alimentation cables between the computer, the power supplier and the DE1-SOC board.

3.1.2 Transfer of program

1. Click on tools in the Quartus environment > Programmer

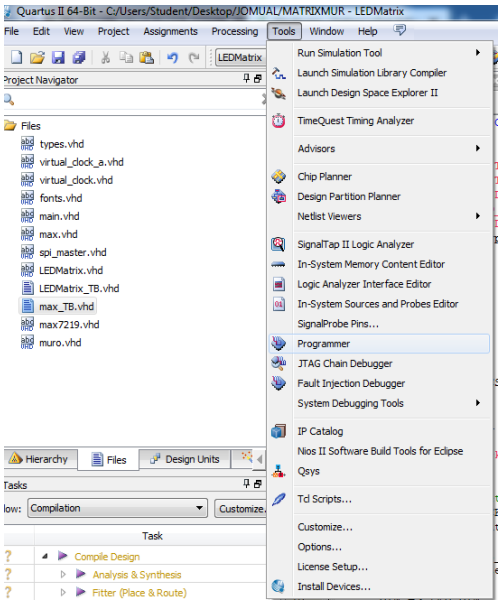

FIG. 3.1 : figure

2. Click Auto Detect and then select the correct device (5CSEMA5). Both FPGA device and HPS should be detected, as shown

3. Double click the green rectangle region shown and Select to add the file. Programming File page will appear. Select the correct ṡof file located in your project directory inside the folder "output_files". the file in our case is the "LEDMatrix.soc", select it and open

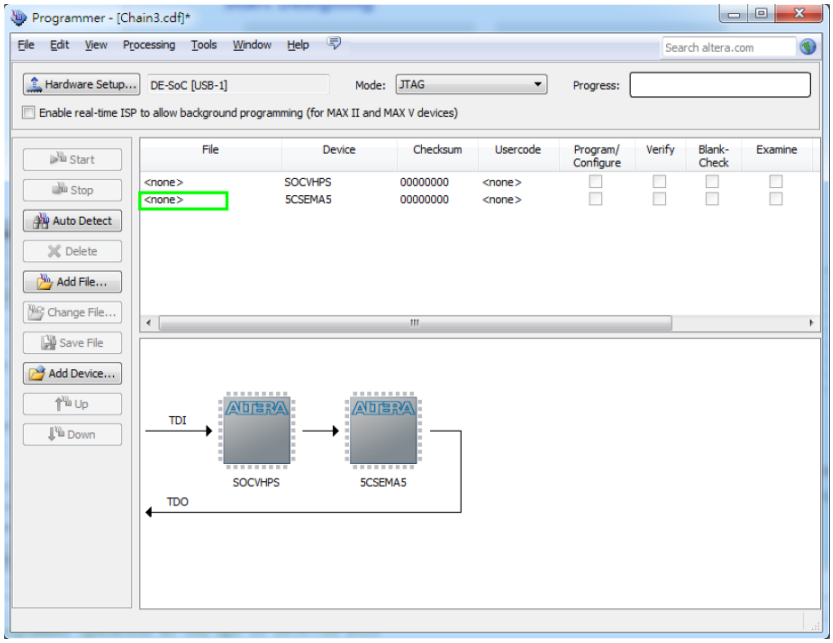

Fig. 3.2 : figure

4. Now be careful to click on the file then you are ready to flash the FPGA board. The ṡof file must be correctly selected then click on "start"

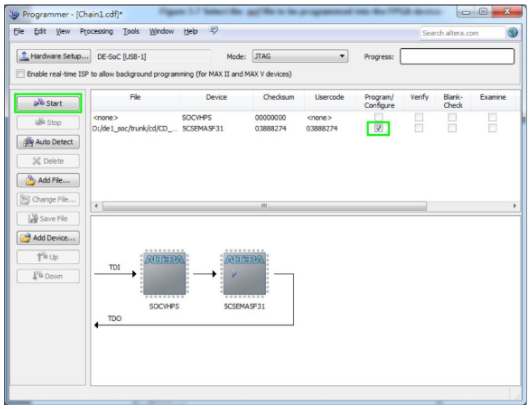

Fig. 3.3 : figure

3.2 The LED matrix

After flashing process we can now see on those pictures the result on the led. A counter going from 0 to 99 continuously.

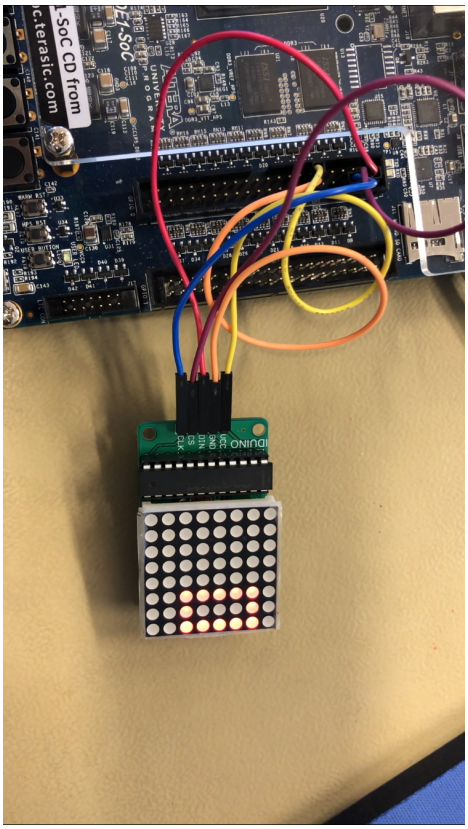

FIG. 3.4 : LED demonstration : counter 0

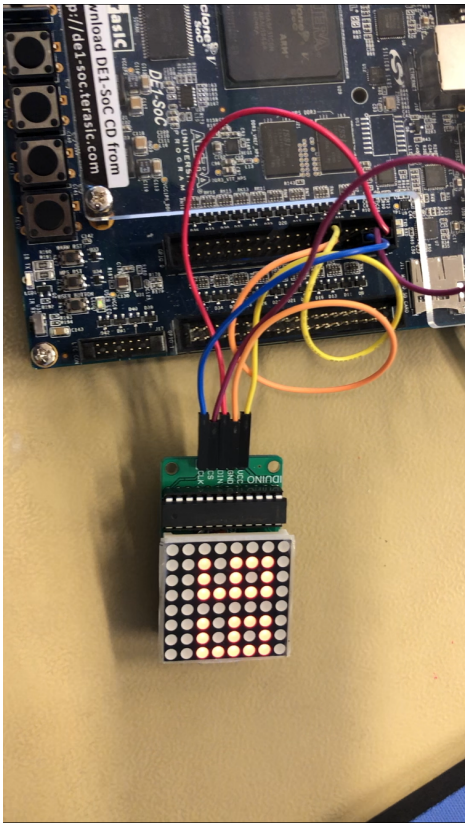

FIG. 3.5 : LED demonstration : counter 99

Conclusion

To conclude, The manipulation of LED Matrix is based on a driver who command the MAX 7219. The MAX 7219, in his turn, will command the display on the LED Matrix. Once we code all of this, we can display whatever we want just by changing some value in our code, it's quite simple. However, it's really important at first to understand how all this "connections" works and the properties of each block. Also, how the blocks are bring together is a question that you need to be able to answer before being able to code properly.