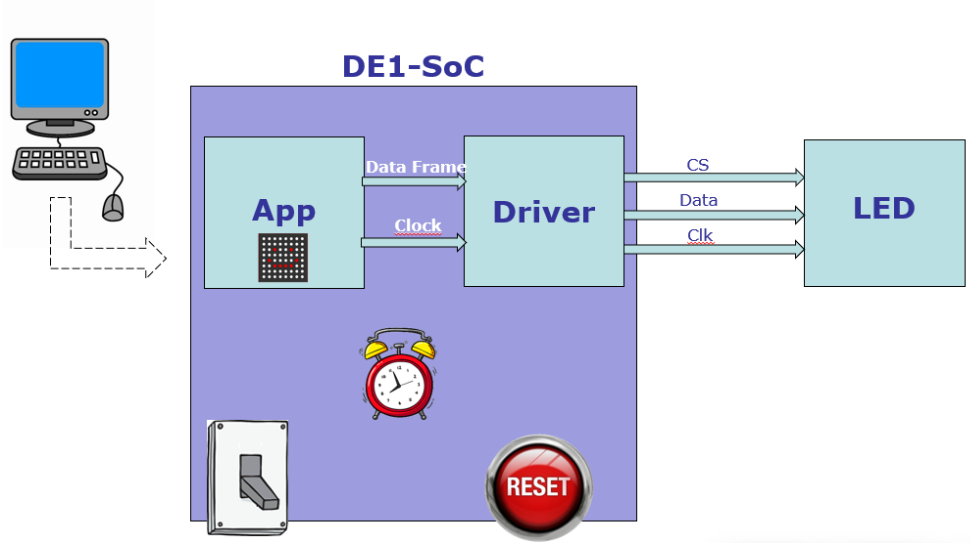
LED Matrix driver for DE1-SoC : Tutorial

Introduction

This document is a tutorial to help you realize the project described in Github. The picture below shows an oversimplified schematic view of our system.



The project uses two main hardware components: the altera DE1-SoC and a led Matrix with a MAX7219 driving system. The internal tools of the board drive the matrix and interact with the user.

- The first step is the design of a driver to make the DE1 and MAX7219 able to understand each other. Once the driver is done a section will show what results must be obtained according to the Test Bench.

- The second consists in the encoding of the desired display. The application is a code separated from the driver that will provide the display information's. This is the alterable part of the project where any user can choose whatever he wants to display.

- Finally, a small part is about the mapping of the pins. Indeed, the signals created by the driver are sent to the proper pin of the board.

After the description of those functions, the document will show how to reproduce the realization.

1.To read before the realization

1.1. Source code presentation

The source code is composed of four different files :“*DriverMatrice.vhd*” and its test bench “*DriverMatriceTB.vhd*”, “Matrice2.vhd” and “MaxLed_inter” which are respectively the application and the mapping code.

A. The Driver, this code will allow the Altera SoC to communicate with the led matrix.

```
entity DriverMatrice is

    port
    (
        iclkl0k  : in   std_logic;
        data     : out std_logic;
        itrame : in std_logic_vector(15 downto 0);
        CS_n : out std_logic;
        Cll0k : out std_logic
    );

end entity;
```

In the first part of the code, we have computed the entity. In it, you can observe 3 outputs that will be sent to the matrix and 2 inputs that will come from the application.

In the second part of the code, we have computed the architecture which is divided in two process. In the first section, we created a process that goes through 16 different states and repeats itself cyclically. State 1 is "Recept". It is in this state that the variable "donnee" receives a 16-bit frame “itrame”. Then, the other 16 states serve as counters.

```
process (iclkl0k,state,itrame)

begin
    if (falling_edge(iclkl0k)) then
        -- Determine the next state synchronously, based on
        -- the current state and the input
        case state is
            when Recept=>
                donnee <= itrame;
               state <= dl5;
            when dl5=>
                    state <= dl4;
            when dl4=>
                    state <= dl3;
            when dl3=>
                    state <= dl2;
            when dl2=>
                    state <= dl1;
```

As you can see in the second part of the code, the process is almost the same. The difference is that it sends the 16-bits frame bit-by-bit to the led matrix. Indeed, the variable “donnee” is sending one bit to data at each process state.

```
process (donnee,state)
begin
        case state is
            when Recept=>
                null;
            when dl5=>
                    data <= donnee(15);
                    CS_n <= '0';
            when dl4=>
                    data <= donnee(14);
                    CS_n <= '0';
            when dl3=>
                    data <= donnee(13);
                    CS_n <= '0';
            when dl2=>
                    data <= donnee(12);
                    CS_n <= '0';
```

In the two photos above, we have only shown you a part of each process. However, since the frame contains 16 bits and they are sent one by one, there are 16 different states.

b) After compiling it, it is now time to test its performance. That's why we created the test bench file "DriverMatrix_TB.vhd" is used to analyze the driver. In this file, we implemented a "clock" and we also sent a frame in the "itrame" variable. A port map has also been implemented to link the Test Bench variables to the driver ones.

```
begin
] uut: DriverMatrice PORT MAP (
            iclk10k=>iclk10k,
                data     =>data,
                itrame =>itrame,
                CS_n =>CS_n,
                Cl10k =>Cl10k
    );


-- Definition of the clock process.
]   clk_process :process begin
        iclk10k <= '0';
        wait for clk_period/2;
        iclk10k <= '1';
        wait for clk_period/2;

    end process;

    -- Stimuli process.
]   stimuli: process begin
        itrame <= tramescan;
        wait for 200 ns;
--
    end process;
end rtl;
```

The simulation results will be shown in the "during the realization" section.

C. In this section, we will explain how the application code works. It consists of a "case" function that sends five initialization frames each time the Altera card is turned on. These five initialization frames will be explained in 1.3.

```
when init =>
    case v is       -- Initialisation des trames
        when 0 =>
            otrame <= tramescan;

        when 1 =>
            otrame <= tramedecodemod;

        when 2 =>
            otrame <= trameshtdwn;

        when 3 =>
            otrame <= tramedisplay;

        when 4 =>
            otrame <= trameintensity;
        when others => null;
    end case ;
```

After the initialization, the "case" will periodically send eight different frames. These frames are composed of 16 different bits and each frame correspond to a line of the led matrix.
According to the datasheet of the matrix, the frames sent to the matrix must have a certain form:

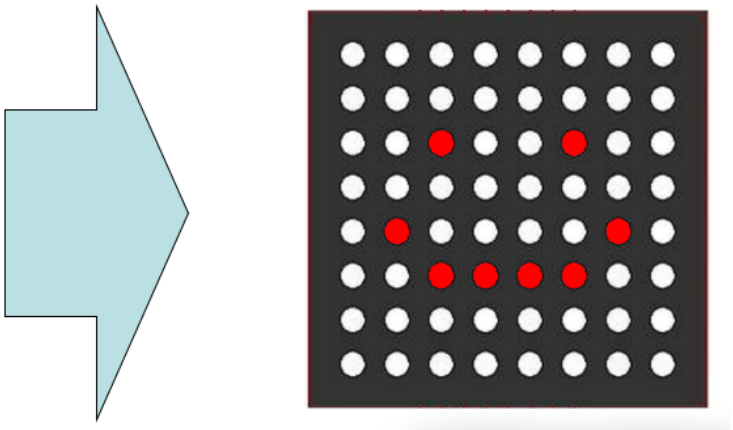
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB	DATA						LSB

The first 4 bits are never used for this device. Then, the bits from D8 to D11 are used to address a line of the matrix. Finally, the last 8 bits each monitor one led in the line. If the bits are equal to 1, the corresponding LEDs will be lit and in the same way, they will be off if the bits are equal to 0.

```
case t is
    when 1 =>
        otrame <="0000" & "0001" & "00000000" ;
    when 2 =>
        otrame <="0000" & "0010" & "00100100" ;
    when 3 =>
        otrame <="0000" & "0011" & "00000000" ;
    when 4 =>
        otrame <="0000" & "0100" & "01000010" ;
    when 5 =>
        otrame <="0000" & "0101" & "00111100" ;
    when 6 =>
        otrame <="0000" & "0110" & "00000000" ;
    when 7 =>
        otrame <="0000" & "0111" & "00000000" ;
    when 8 =>
        otrame <="0000" & "1000" & "00000000" ;
    when others => null;
  state <= cpt;
end case;
```

In the case of our project, we decided to display a smiley on the matrix. We therefore filled in 8 different frames with each the address of the corresponding line. Then, we completed the "data" parts of each line in order to arrive at a happy smiley.

Df 1 : '00000000'
Df 2 : '00000000'
Df 3 : '00100100'
Df 4 : '00000000'
Df 5 : '01000010'
Df 6 : '00111100'
Df 7 : '00000000'
Df 8 : '00000000'



1.3. Matrix initialization

When explaining the application, we needed initialization frames to use the.
The different frames are :

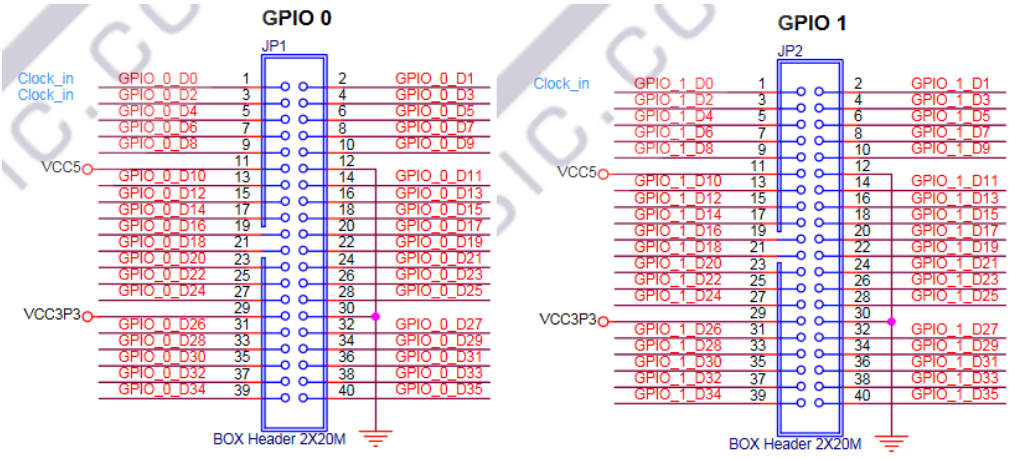
- DecodeMode
- Intensity control which allows to manage the intensity of the LEDs thanks to codes available in the MAX7221 datasheet.
- Display-Test register, it can work in two different ways, either in normal or in test. In our project, we only use the normal mode.
- Scan limit register as well as the shutdown Mode that can be used to save energy.

Those five different frames are presented in the table below.

Register:	**D15-D12**	**D11**	**D10**	**D9**	**D8**	**HexCode**
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xXF

1.2. Mapping

As we know, the DE1-SoC has many pins and can thus fit to any sort of project. In addition to our Driver and our application, we must assign each signal to a separated pin. To do so, we created the file “MatriceLed_inter.vhd” who’s task is to map every required signal to the adequate pin. The picture below shows all the available pins with their names.



We'll focus on GPIO_0 and assign our signals to the first three on the right side : D1,D3 and D5.

```
architecture rtl of maxLed_inter is
    signal strame : std_logic_vector(15 downto 0);
    signal sinput : std_logic;
    signal soclk10k : std_logic;

    component Matrice2 is
        port
        (
            clk      : in   std_logic;
            input    : in   std_logic_vector(0 downto 0);
            otrame   : out std_logic_vector(15 downto 0);
            oclk10k : out std_logic
        );
    end component;
    component DriverMatrice is
        port
        (
            iclk10k  : in   std_logic;
            data     : out std_logic;
            itrame : in std_logic_vector(15 downto 0);
            CS_n : out std_logic;
            Cll0k : out std_logic
        );
    end component;
begin

    IMatrice2 :Matrice2 port map (
            clk => CLOCK_50,
            input => SW,
            otrame => strame,
            oclk10k =>soclk10k
        );


    IDriverMatrice:DriverMatrice port map(
        iclk10k => soclk10k,
        itrame => strame,
        data =>  GPIO_0(1),
        CS_n =>  GPIO_0(3),
        Cll0k => GPIO_0(5)

    );


end rtl;
```

This code is named "MaxLed_inter". It is responsible of the assignment of each pins. We first implement the Driver as a component to get the desired signals and we link them to the desired pins of GPIO_0.

2. To read during the realisation

2.1. Create a new project on Quartus II

First of all, you have to download our code from GitHub. Then you have to create a project in Quartus II :

- *Launch Quartus II*
- *Create a project: file -> New Project Wizard -> Next (until created)*
- *Import all files from downloaded folder*
- *Save project as VHD file and with the same name of the entity*

2.2. Test of the driver

To be sure if the code is operational, we use the previously described Test Bench to simulate the Driver.

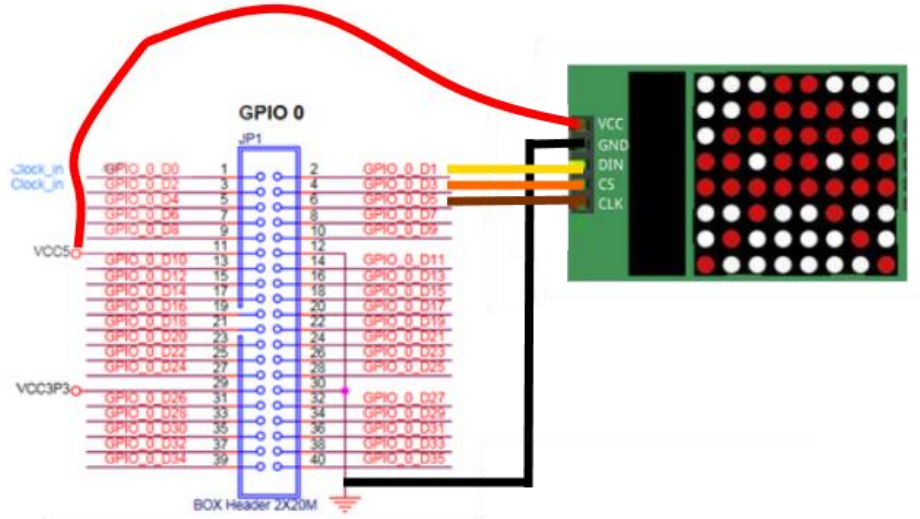
To do so, a data frame is randomly chosen to see if the program fits the expected message. The picture below shows the simulation results of the Test Bench.

The frame chosen contains 16 bits and represent one initialization frame of data frame for the matrix.

"*0000101100000111"* is the Frame, we see that when CS comes to zero, the clock begins, and the data are sent. If we look closer to the data waveform, we can observe that the correct message has been sent.

2.3. Wiring

The wiring of the different components should be done accordingly to this picture :



Indeed, the connection between the Matrix need a 5 volts DC alimentation. We thus connect the VCC and the GND respectively on the VCC5 and GND pins of GPIO 0.

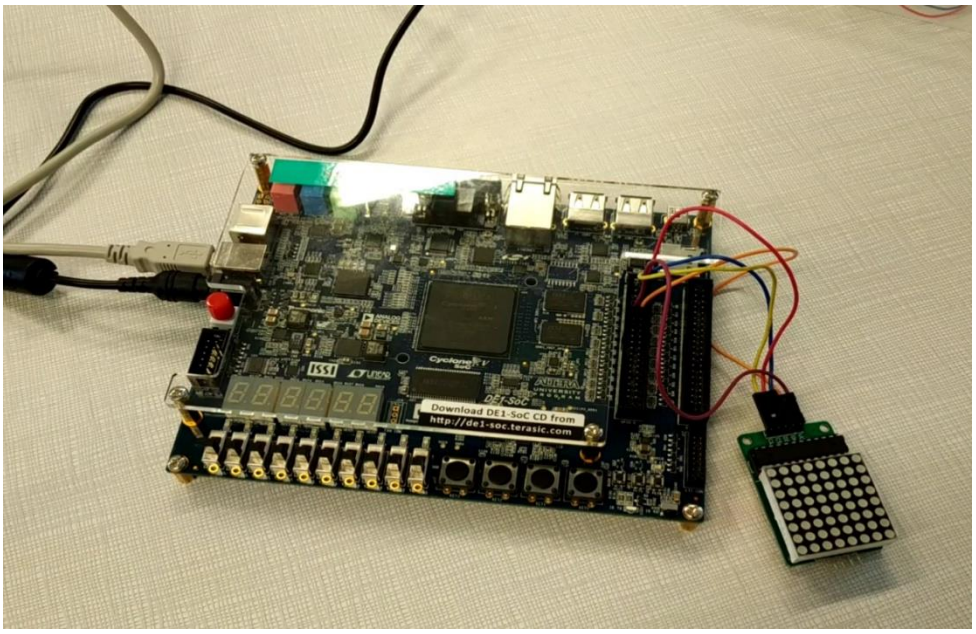
As described in the mapping section, the driver transfers three signals to the MAX7219:

- data => mapped on GPO_0_D1
- CS => mapped on GPO_0_D3
- CLK => mapped on GPO_0_D5

2.4. Start

Once the wiring is completed and the Driver has been checked, the realization of the display is obtained through the following steps.

- Fully compile the project
- Go to Program device
- Switch on the card
- Select it if it’s directly displayed
- SoC series (if it does not automatically start)
- Push on big button “START”


Download DE1-SoC CD from
http://de1-soc.terasic.com

Enjoy !


Achievement unlocked
LED Matrix Skills Mastered