Embedded Systems
Processor Architecture Laboratory

DE0-Nano-SoC Lab4 - Design of an Embedded System, Mini-Project

Teacher :

Beuchat René

Student :

Lefebure Nicolas
Lamyae Omari

8 janvier 2020

Table des matières

1 Introduction

The purpose of this lab is to implement the detailed design described in the report of the previous lab of the LCD display controller. More explicitly, its goal is to display images stored in the SDRAM. In order to do that, a DMA will read the data from memory through Avalon read accesses and send them to the FIFO that will provide them continuously to the LCD control part. This latter part will send RGB data following the 8080 I protocol. LT24 was used with DE0-nano and the design has been implemented using QUARTUS and debugged with Modelsim while Eclipse was used for the software part to control the display. The goal of this lab is the design of custom master interface. Unlike the slave interface, it is targeted towards complex use cases where large amounts of data need to be moved. An LCD display controller and a camera controller associated with a Nios II processor need to be design. In this report in particular we focus on the display part (LCD), which is the displacement of the data from a memory to a peripheral.

2 System Schematic

The block diagram of the full system below shows how the various sub components interact with each other :

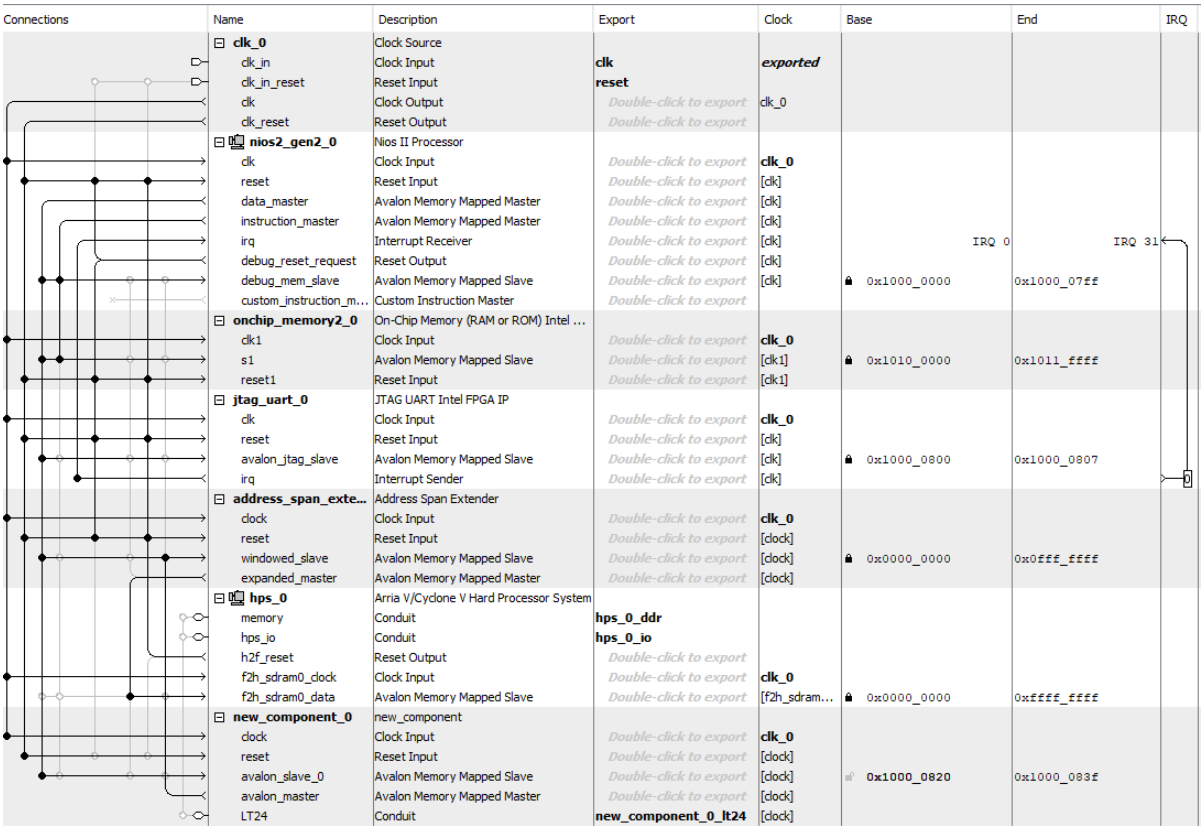

FIGURE 1: Qsys diagram showing the system components

The system is composed of 7 components as shown in figure 1 :

- Nios II processor.
- On chip memory (RAM or ROM) :used only for the Nios II processor's instruction and data memory.
- JTAG UART .
- HPS (the DDR3 memory controller) : As the size of the on chip memory is not enough to store

a full frame, we used the external DDR3 memory (1 GB memory).
— The address span extender :represents the bridge used to interface with port 0 of the HPS to not access reserved memory of DDR3. To resume basically The role of the bridge, The accesses via this latter are redirected to a specific offset in order to allow access to a reserved 256 MB of continuous memory.
— Clock & Reset.
— new component : represents Our LCD controller .

3 Simulations

In order to check if our system adheres to the specifications of its register map, We have performed some simulations with Modelsim.

3.1 DMA

The state machine of the DMA is the same as the one described in the previous report. We just added some modifications as shown below.
— Start bit : This bit allows to trigger the DMA i.e it allows to pass from the state Idle to the state Read Request. Start goes high now when a new address is written in the image address register and goes low when we goes to the state Read request
The timing diagrams obtained using Modelsim with a burstcount=2 an Length =1(1burst) are illustrated below.

3.1.1 initialization and starting the DMA

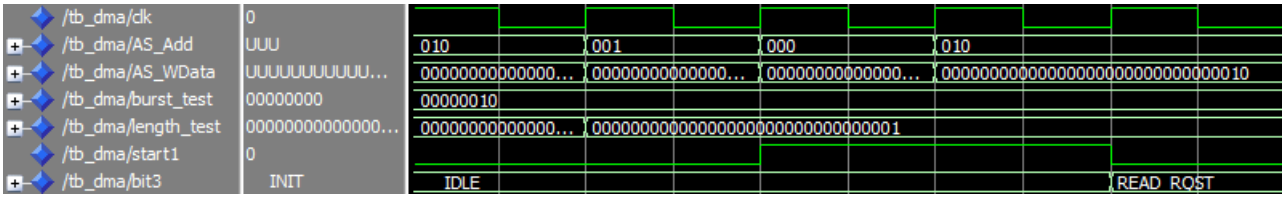

FIGURE 2: Transition from state Idle to state Read request

We start by configuring the burst and length registers, and then we specify the image address which allows the start to go high and thus trigger the DMA. So, we pass to the state Read request as shown in Figure 2 .

3.1.2 Avalon Burst read

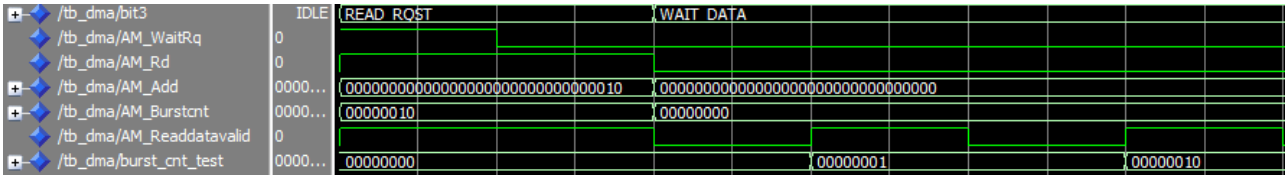

FIGURE 3: Transition from Read request to Wait data

The timing diagram fits well with the diagram in Figure 4.The address, BurstCnt and Read are available only for the first cycle composed of two clock cycle. The waitrequest is high only for the first

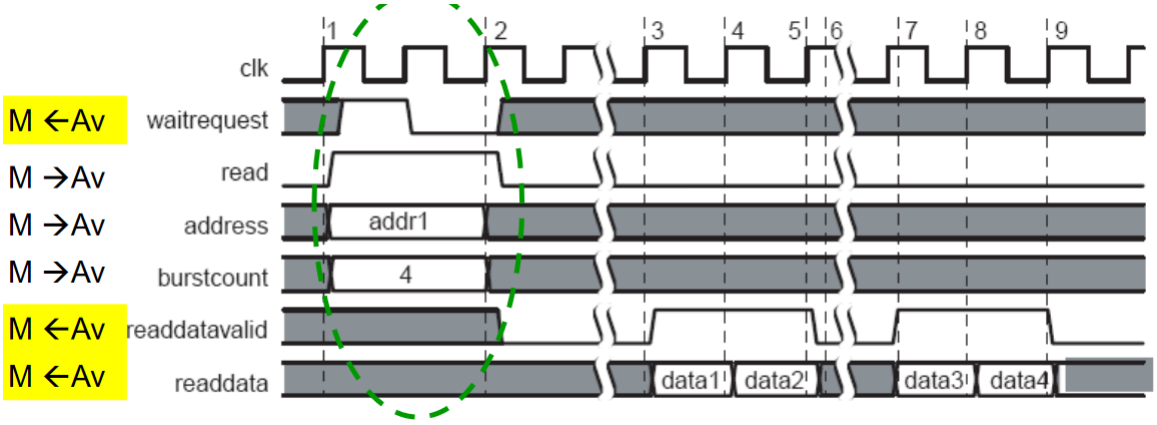

FIGURE 4: Avalon burst read

period and then goes low. The number of burstcount needs to be generated with Readdatavalid which is connected with the signal wrreq of the FIFO to allow writing valid data. When the waitrequest signal is deasserted, if we have free places to write a burst in the FIFO, we pass to the state Wait Data.

3.1.3 Transition between states

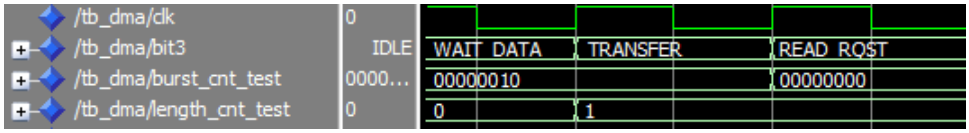

FIGURE 5: Transition from Wait Data to Transfer

As soon as the Burst counter reaches the value of BurstCount(equal to 2 in our case), the DMA goes to the state Transfer. In this state, the length Counter is incremented. As The length register is set to 2 in the simulation, the DMA should go to the state Read request and wait for another burst to finish.

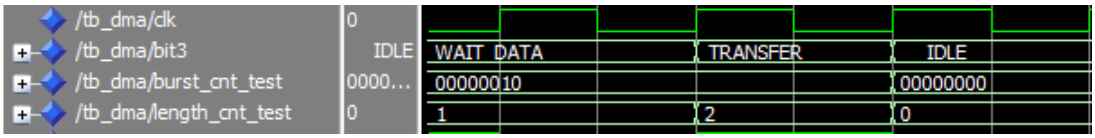

FIGURE 6: Transition from Transfer to IDLE

As soon as the Length counter reaches the value of Length, the DMA goes to the initial state Idle (the DMA finishes writing a frame) and wait for another frame.

3.2 LT24

For the LT24, we use a register CSX_Reg that stays low if the command or data that we send require following instructions. For example some commands have multiple parameters, so the CSX_Reg shall stay low until the parameter before the last to ensure that CSX (Chip select) stays low (active low) until sending all the parameters of the command.

3.2.1 Send a Command

LT24 has been simulated for the case where we send a command (DCX is low) that not require following instruction (CSX_Reg is high and CSX= CSX_Reg)

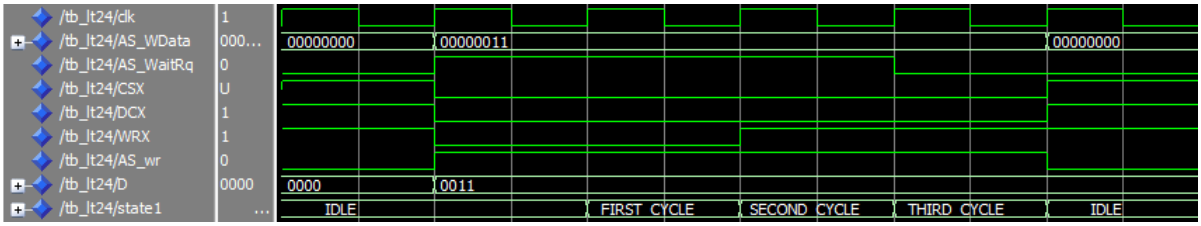

FIGURE 7: Send a single command(DCX is low)

As shown in Figure 7, As_WaitRq stays high for three cycles. In this way, we guarantee that the bus stays busy for 4 cycles as described in the state machine of the previous report.

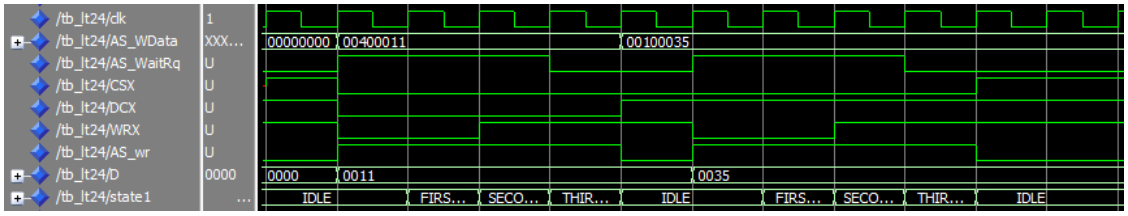

FIGURE 8: Send a command followed by Data

3.2.2 Send Command with 1 parameter

In this case, CSX_Reg should stay low to ensure that the CSX stays active until sending the parameter.

AS_WData	CSX_Reg(not AS_WData(22))	DCX_Reg
00000000	0	0
00400011	0	0
00100035	1	1

3.2.3 Send Write Command

The controller enters in the 'Wait FiFo' state only if the command is 2Ch : Memory write and start loading pixels from the FIFO if there is more than a burst of information stored in it (state 'Write In'). The CSX goes high in the state 'Wait FiFo' to pause parallel interface and then goes low in the state 'Write in'.

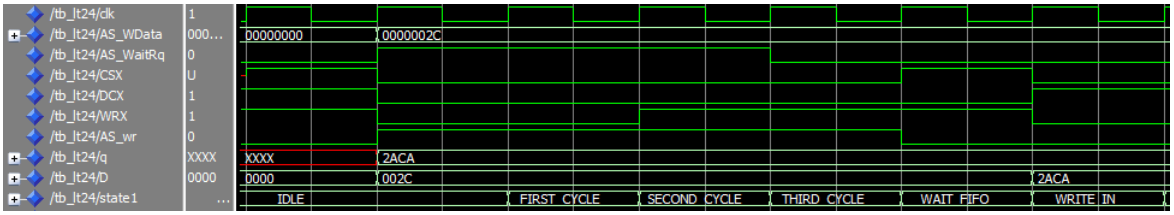

FIGURE 9: Send Write Command

In the state write in, we see that the output D take the value of the input q connected to the output of the FIFO. The controller then waits four cycles of 80 ns before going to 'Wait FiFO' state or 'Write in' state, in order to respect the timing protocol as shown in Figure 10

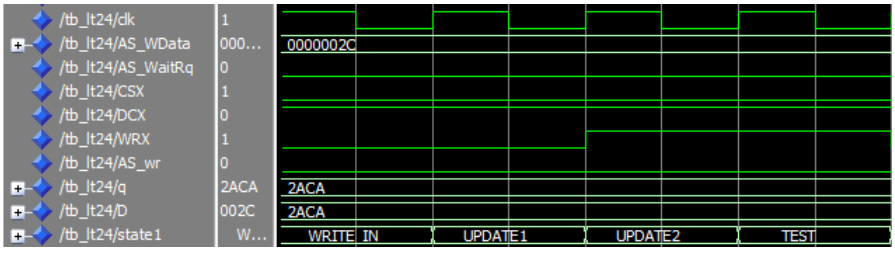

FIGURE 10: Timing protocol

3.3 LCD Controller

In this section, we will test the costum IP LCD controller with AM_RData="00020001". We obtain the following timing diagrams :

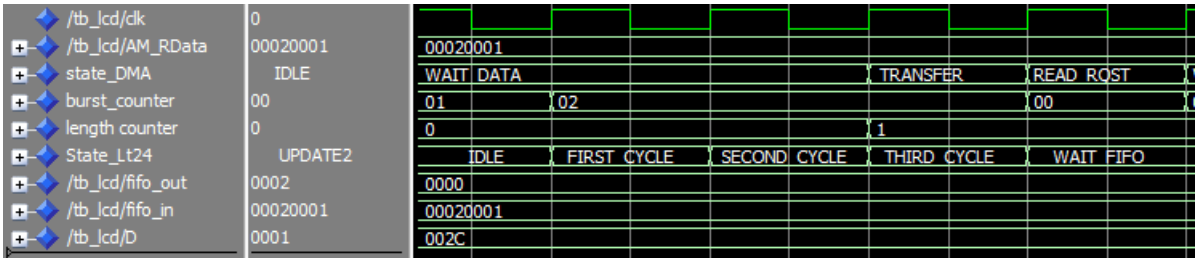

FIGURE 11: Writing burst to the FIFO

The burstCnt and the length are set to 2 in this simulation. From the figure 11,we can see that the DMA writes well the first burst so the length counter increases by 1 in the state 'Transfer' and the LT24 receives a write command (D="002C"). But to start reading from DMA, the LT24 should wait until there is at least a burst of information in the FiFO.

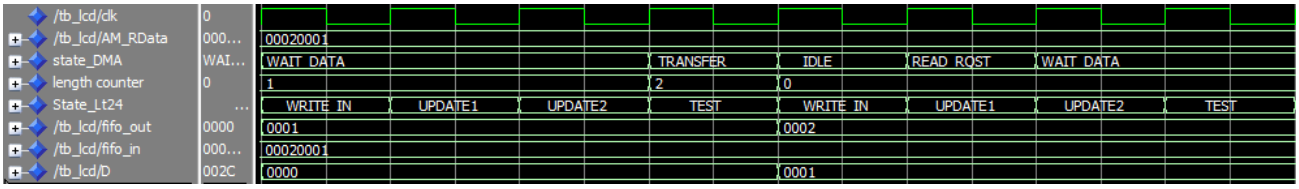

FIGURE 12: Reading data from FIFO

The FIFO out and FIFO In signals match well with what was expected. The DMA returns to the "Idle" state because the Length Counter reaches the Length value.The output of the LT24 "D" receives well the value stored in the FIFO.

4 VHDL Code

4.1 Top Level

The Figure 13 denotes the ports of our custom IP component. The LCD_Controller.vhdl file represents the top level of our component and shows how the ports of the different sub blocks (LT24, DMA, FIFO) are connected. If the slave address is "100",The write signal is send to the LT24. In the other cases, the write signal is send to the DMA. The signal wait request of the slave is active if the wait request of the slave of the DMA or the wait request of the LT24 is active.

```
entity LCD_controller is
        port(
                        clk              : in std_logic;
                        Reset            : in std_logic;

                        AS_Add           : in std_logic_vector(2 downto 0);
                        AS_CS            : in std_logic;
                        AS_wr            : in std_logic;
                        AS_WData         : in std_logic_vector(31 downto 0);
                        AS_Rd            : in std_logic;
                        AS_RData         : out std_logic_vector(31 downto 0);
                        AS_WaitRq        : out std_logic;

                        AM_Add           : out std_logic_vector(31 downto 0);
                        AM_BE            : out std_logic_vector(3 downto 0);
                        AM_Rd            : out std_logic;
                        AM_RData         : in std_logic_vector(31 downto 0);
                        AM_WaitRq        : in std_logic;
                        AM_Burstcnt      : out std_logic_vector(7 downto 0);
                        AM_Readdatavalid : in std_logic;

                        CSX                     : out std_logic;
                        DCX                     : out std_logic;
                        WRX                     : out std_logic;
                        D                     : out std_logic_vector(15 downto 0);

                        LCD_Config                : out std_logic_vector(1 downto 0)

        );

end LCD_controller;
```

FIGURE 13: Top level of our custom IP

4.2 LT24

```
entity LT24 is
        port(

                clk             : in  std_logic;
                Reset           : in  std_logic := '1';

                AS_CS           : in  std_logic;
                AS_wr           : in  std_logic;
                AS_WData        : in  std_logic_vector(31 downto 0);
                AS_WaitRq       : out std_logic;

                rdreq                 : out  std_logic;
                q                        : in std_logic_vector(15 DOWNTO 0);
                rdusedw             : in STD_LOGIC_VECTOR(9 DOWNTO 0);


                CSX                   : out std_logic;
                DCX                   : out std_logic;
                WRX                   : out std_logic;
                D                   : out std_logic_vector(15 downto 0);

                LCD_Config              : out std_logic_vector(1 downto 0)

                );
end entity LT24;
```

FIGURE 14: Ports of the LT24

4.3 DMA

```
entity DMA is
        port(

                clk             : in std_logic;
                Reset           : in std_logic;

                AS_Add          : in std_logic_vector(2 downto 0);
                AS_CS           : in std_logic;
                AS_wr           : in std_logic;
                AS_WData        : in std_logic_vector(31 downto 0);
                AS_Rd           : in std_logic;
                AS_RData        : out std_logic_vector(31 downto 0);
                AS_WaitRq       : out std_logic;

                AM_Add          : out std_logic_vector(31 downto 0);
                AM_BE           : out std_logic_vector(3 downto 0);
                AM_Rd           : out std_logic;
                AM_RData        : in std_logic_vector(31 downto 0);
                AM_WaitRq       : in std_logic;
                AM_Burstcnt     : out std_logic_vector(7 downto 0);
                AM_Readdatavalid : in std_logic;

                aclr              : out std_logic;
                data              : out std_logic_vector(31 DOWNTO 0);
                wrreq             : out std_logic;
                wrusedw           : in std_logic_vector(8 DOWNTO 0)


        );
end entity DMA;
```

FIGURE 15: Ports of the DMA

5 Software part

In order to test the proper functioning of our LCD, we have developed some functions in C using the Nios II Software Build Tools for Eclipse.The C code is composed of two files : functions.h and helloworld.c.

- functions.h :This header file contains some macros that allow us to configure the LT24 and the DMA easily.It contains also declarations of the C functions used in helloworld.c
- helloworld.c :contains the functions used to initialize the components and test them.

5.1 functions.h

The table below describes some of the macros that we have implemented :

function name	Description
LCD_WR_REG(REG)	send a command address
LCD_WR_DATA(REG)	send command data
LCD_ON_UP()	set LCD_on to 1
DMA_START_ADRESS(Address)	send the image address to the DMA
DMA_START()	triggers the DMA
DMA_RESTART()	to reset the DMA

5.2 helloworld.c

The table below shows the different functions that we implemented.The file contains also the main function.

function name	Description
Init_DMA ()	Initialization of the DMA :Reset and set the length and burst registers.
init_LCD()	Initialization of the LCD :all the setting configuration commands are sent
write_cmd()	send a write command to the LCD
Test()	Used to test the LCD independently from the DMA ,it allows to write a frame with the same color directly to the memory of the LCD
test_DMA(int* addr0)	write a frame in the DRAM

6 Conclusion

To conclude on this project, we can see that several things unfortunately did not work : we have never managed to read an image from SDRAM with our DMA interface and we did not had time to assemble our project with the one of the Camera group. However we still managed to display some colors on the LCD thanks to our LT24 interface, which was a satisfying moment given the great amount of time that we have spend on the project.