Assignment 3

This task has two parts, both of which are concerned with the on-board LCD display. Part 1 is software- focused, part 2 is hardware-focused.
They can be done in parallel, but the testing of the second part can be done more easily if the first part has been completed before-hand. Both tasks deal with the LCD display mounted on the development board.
For both parts, it is of fundamental importance that you read the relevant sections of the board description (`Genesys_rm.pdf`) and the LCD datasheet (`ST7066.pdf`) to get an overview about how the LCD controller works.

Question 3.1

The file `LCDController.vhd` contains a wishbone slave which acts as a controller to connect to the LCD Display. Take a careful look at it. Note that it does not contain any timing information regarding the LCD Display interface. This has to be implemented by software.

a) You must integrate the above LCD display controller component into the system.
In order to do so first map the LCD display's interface signals into your system. Uncomment the lines in the *User Constraint File* corresponding to the LCD signals (signals starting with `LCD`), and add them to the top level entity's port list.
Proceed to integrate the 'LCDController' component by the same process described in the previous assignment. Choose a reasonable base address for the component between `0x000F0000` and `0x004F0000`.
Commit your Changes.

b) Programming for the LCD display requires the controller and the timing information for the display interface to be taken into account.
Create a file `assignment31code.prog` in the `programs` subdirectory, in which you should write the software for this part.
Write a subroutine that features sending a single command to the display. It should be used to send all commands after initialization.
While exact timing information is written in the datasheet, a rough implementation of the timing will do:

1. Write a subroutine to send a command with the enable bit set to command register and wait for 10 microseconds.

2. Write a subroutine to send a command without the enable bit set and wait another 10 microseconds.

Do not forget to add and commit your software.

c) Write an initialization routine for the LCD. The LCD initialization process is explained in the LCD data sheet.
Using both the command subroutine and initialization subroutine, write a main program that displays the phrase "Assembler rocks!".
Commit your changes.
Before synthesizing and testing your code on the FPGA, consider how you can test via simulation.
After a working bitfile is achieved, save it for a later demonstration.
Hint:
VHDL supports a signal type 'time'. You can track the passage of time in your test bench by using a variable of this type and incrementing it at clock events. Asserting the amount of time passed before events happen can be achieved with wait statements. An example is given in Listing 1, which waits for either an event on the signal `LCD_Enable` or for ten microseconds simulation time to pass.
It is not mandatory to write a testbench, but it is recommended, since you can reuse it for the second part of the assignment.
If you write a testbench, do not forget to add and commit it to the git repository.

Listing 1: Wait statements

```
process(...) is
    ...
begin
    ...
    some_time <= current_time;
    wait on LCD_Enable for 10 us;
    assert current_time >= (some_time + 10 us) report "Time requirement missed!" severity ERROR;
    ...
```

Question 3.2

As in the previous task, the LCD display must be interfaced to issue commands .
But this time it is to be implemented in hardware, taking the task of initialization and timing off the hands of the software.
For this purpose, the original controller component is to be extended, implementing a more sophisticated state machine which initializes the display and takes care of the timing.

a) Create a copy of the `LCDController.vhd` file named `LCDControllerAdv.vhd`. Rename the component in the copy.
For the extension of the design, a few design recommendations are given: Implement two state machines to control the display. One will generate the enable signal 'E' (called 'E signal generator', shown in Fig. 2) for the display, while the other (called 'display state machine', shown in Fig. 1) sets the other rest of the signals according to the user input, and ensures that all timing requirements are satisfied simply by waiting for a standard delay of 1.5 ms. The two state machines communicate with each other through certain signals. The second state machine will also feature the initialization sequence of at least the function set. (It can be extended to the whole initialization sequence.) The register interface should feature a flag which indicates if the module is initialized, which can be easily derived from the current state of the display state machine.
The graphics are just an illustration and not golden models. Also, the implementation does not need to be of Mealy automata type.
As software part, a simple routine can be used to poll for the busy flag of the module and then write the desired data into its register.
You should be able to reuse the testbench(es) from the previous task with only minimal changes.
Do not forget to add and commit your module (and testbench).

b) Similar to 1.c, write software to interface your hardware component to display a message. The message is "Hardware rulz!"
Demonstrate both solutions to the lab assistants to complete the assignment.

Question 3.3 - Challenge

As a *challenge*, rewrite your display controller to check the busy flag of the LCD display in hardware. Write a test software which runs a counter when waiting on the hardware busy flag check. The counter should be implemented in the same way as the wait routine for the software controlled variant.
Print the elapsed counter cycles on the LCD and compare it to the original version.

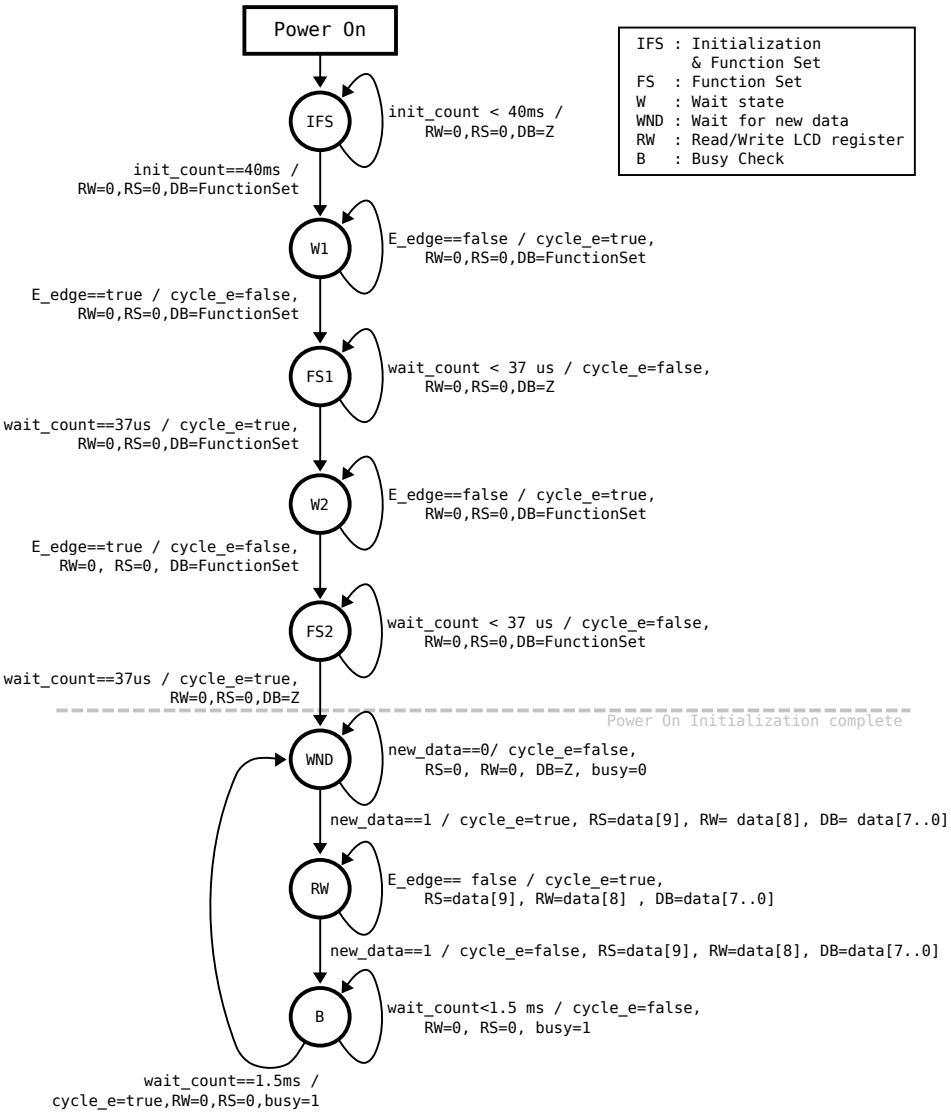

Figure 1: Proposed LCD interface state machine

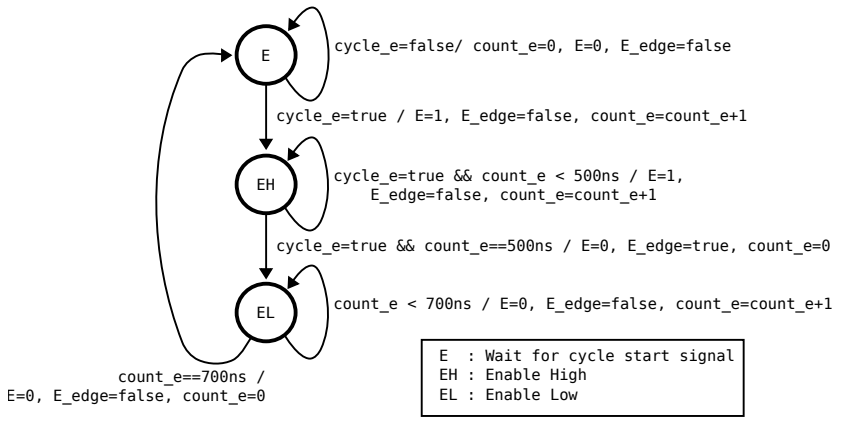

Figure 2: E-signal generator