

# Lab 4-updated : I/O - VGA and PS/2 Keyboard

ECSE 324 - Computer Organization

Winter 2020

## Introduction

In this lab will use the high level I/O capabilities of the DE1-SoC computer with a simulator. In particular, the tasks will:

- Use the VGA controller to display pixels and characters.
- Use the PS/2 port to accept input from a keyboard

You will learn how to perform the aforementioned tasks on virtual DE1-SoC computer using online Computer System Simulator. A small tutorial on how to use the simulator is given in Section 3.

# 1 VGA

For this part, it is necessary to refer to section 4.2 (pp 40-43) of the De1-SoC Computer Manual.

## Brief overview of the De1-SoC computer VGA interface

The VGA controller hardware has already been introduced in the ECSE 222 labs. The De1-SoC computer has a built in VGA controller, and the data displayed to the screen is acquired from two sections in the FPGA on-chip memory - the *pixel buffer* and the *character buffer* - which are described in sufficient detail in section 4.2.1 and 4.2.3 of the De1-SoC Computer Manual. For this lab, it is not required to make use of the *double buffering* feature described in the manual.

## VGA driver

Create an assembly files `VGA.s` where you will write the required subroutines and functions. You are required to write 5 subroutines:

1. `VGA_clear_charbuff_ASM`
2. `VGA_clear_pixelbuff_ASM`
3. `VGA_write_char_ASM`
4. `VGA_write_byte_ASM`
5. `VGA_draw_point_ASM`

The subroutines `VGA_clear_charbuff_ASM` and `VGA_clear_pixelbuff_ASM` should clear (set to 0) all the valid memory locations in the character buffer and pixel buffer, respectively. Note that these two subroutine do not have any input/output arguments.

The subroutine `VGA_write_char_ASM` should write the ASCII code passed in the third argument ( $R_2$ ) to the screen at the (x,y) coordinates given in the first two arguments ( $R_0$  and  $R_1$ ). Essentially, the subroutine will store the value of the third argument at the address calculated with the first two arguments. The subroutine should check that the coordinates supplied are valid (i.e.  $x = [0,79]$  and  $y = [0,59]$ ).

The subroutine `VGA_write_byte_ASM` should write the hexadecimal representation of the value passed in the third argument to the screen. This means that this subroutine will print two characters to the screen! (For example, passing a value of 0xFF in byte should result in the characters 'FF' being displayed on the screen starting at the (x,y) coordinates passed in the first two arguments). Again, check that the (x,y) coordinates are valid, taking into account that two characters will be displayed.

Hint: Both the above subroutines should only access the character buffer memory.

Finally, the `VGA_draw_point_ASM` subroutine will draw a point on the screen with the colour as indicated in the third argument, by accessing only the pixel buffer memory. This subroutine is very similar to the `VGA_write_char_ASM` subroutine.

**NOTE-1:** Use suffixes 'B' and 'H' with the assembly memory access instructions in order to read/modify the bytes/half-words of the memory contents.

**NOTE-2:** You must follow the conventions taught in the class.

**NOTE-3:** None of the subroutines has output arguments. Only the last three subroutines have three arguments (i.e., the (x,y) coordinates and the data to be stored in the character/pixel buffers.)

## Simple VGA application

Build an assembly based application to test the functionality of the VGA subroutines. Translate the three functions shown in Figure 1 (in C) in assembly.

```
7 void test_char() {
8     int x,y;
9     char c = 0;
10
11     for(y=0 ; y<=59 ; y++)
12         for(x=0 ; x<=79 ; x++)
13             VGA_write_char_ASM(x,y,c++);
14 }
15
16 void test_byte() {
17     int x,y;
18     char c = 0;
19
20     for(y=0 ; y<=59 ; y++)
21         for(x=0 ; x<=79 ; x+=3)
22             VGA_write_byte_ASM(x,y,c++);
23 }
24
25 void test_pixel() {
26     int x,y;
27     unsigned short colour = 0;
28
29     for(y=0 ; y<=239 ; y++)
30         for(x=0 ; x<=319 ; x++)
31             VGA_draw_point_ASM(x,y,colour++);
32 }
```

Figure 1: C functions used to test the VGA driver

You need to use the push-buttons to perform the following functions:

- **PB0 is pressed:** call the `test_byte()` function.
- **PB1 is pressed:** call the `test_char()` function.
- **PB2 is pressed:** call the `test_pixel()` function.
- **PB3 is pressed:** clear both the character and pixel buffers.

Note that this application relies on the push-buttons subroutines that you wrote in Lab 3. If you are unable to use these drivers, you will not be able to score fully on this part of the lab. However, in such case, we will give you some points if you are able to write a test program that calls the different functions above (without using the push-buttons) to demonstrate that they are fully working.

**NOTE:** You must write all the functions in the same file that you write the subroutines (i.e., `VGA.s`). In other words, you are only required to write all the subroutines and functions in the same file named `VGA.s`.

## 2 Keyboard

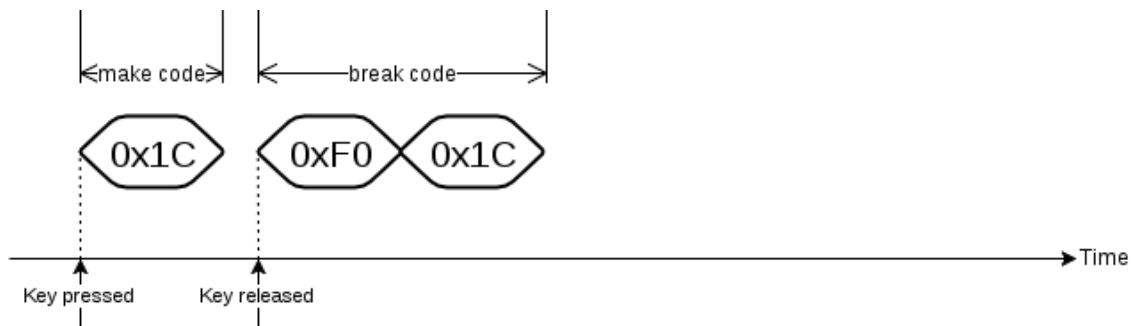
For this part, it is necessary to refer to section 4.5 (pp 45-46) in the De1-SoC Computer Manual.

### Brief overview of the PS/2 Keyboard Protocol

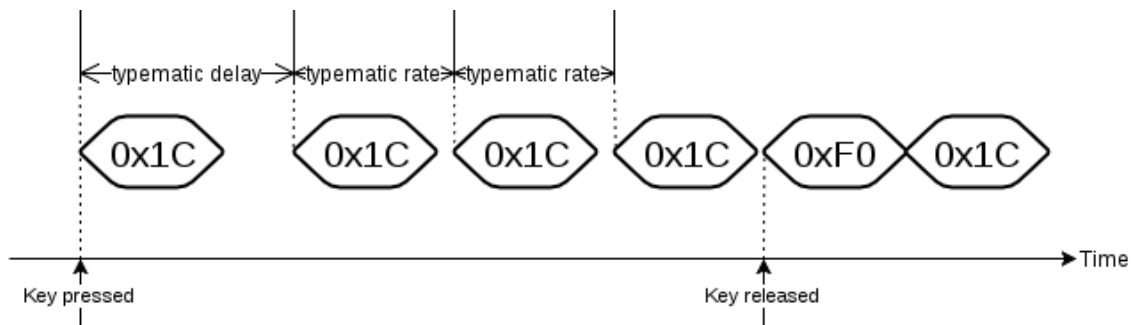
For the purpose of this lab, a very high level description of the PS/2 keyboard protocol is given. A more detailed description can be found in the document named *PS2 Keyboard*.

The PS/2 bus provides data about keystroke events by sending hexadecimal numbers called **scan codes**, which for this lab will vary from 1-3 bytes in length. When a key on the PS/2 keyboard is pressed, a unique scan code called the **make code** is sent, and when the key is released, another scan code called the **break code** is sent. The **scan code** set used in this lab can be found in the document named *PS2 Keyboard* under the *Keyboard Scan Codes: Set 2* section (pages 21–22).

Two other important parameters involved are the **typematic delay** and the **typematic rate**. When a key is pressed, the corresponding **make code** is sent, and if the key is held down, the same **make code** is repeatedly sent **at a constant rate after an initial delay**. The **make code** will stop being sent only if the key is released or another key is pressed. The initial delay between the first and second **make code** is called the **typematic delay**, and the rate at which the **make code** is sent after this is called the **typematic rate**. The **typematic delay** can range from 0.25 seconds to 1.00 second and the **typematic rate** can range from 2.0 cps (characters per second) to 30.0 cps, with default values of 500 ms and 10.9 cps respectively.



(a) Key 'a' is pressed and released



(b) Key "a" is pressed, held down, and then released

Figure 2: Example of data received on the PS/2 bus

## PS/2 keyboard driver

Create an assembly file and name it `ps2_keyboard.s`. For this lab, simply implement a subroutine with the following specifications:

- **Name:** `read_PS2_data_ASM`
- **Input argument ( $R_0$ ):** A memory address in which the data that is read from the PS/2 keyboard will be stored (pointer argument)
- **output argument ( $R_0$ ):** Integer that denotes whether the data read is valid or not
- **Description:** The subroutine will check the *RVALID* bit in the PS/2 Data register. If it is valid, then the data from the same register should be stored at the address in the pointer argument, and the subroutine should return 1 to denote valid data. If the *RVALID* bit is not set, then the subroutine should simply return 0.

## Simple keyboard application

Create a simple application that uses the PS/2 keyboard and VGA monitor. The application should read raw data from the keyboard and display it to the screen if it is valid. Only the `VGA_write_byte_ASM` subroutine is needed, and the input byte is simply the data read from the keyboard.

**Note:** In the program, keep track of the x,y coordinates where the byte is being written. For example, write the first byte at (0,0) and the second byte at (3,0) and so on until the first line on the screen is full, and then start writing bytes at (0,1), (3,1), (6,1) etc. A gap of 3 x co-ordinates is given since each byte will display two characters, and one more for a space between each byte.

### 3 CPulator: Computer System Simulator

In this section, we provide a simple tutorial on how you can use the online simulator for this Lab.

- 1- Open the Computer System Simulator using this link: <https://cpulator.01xz.net>.
- 2- select ARMv7 architecture and ARMv7 DE1-SoC (v16.1) system and click on **Go** (see Figure 3).

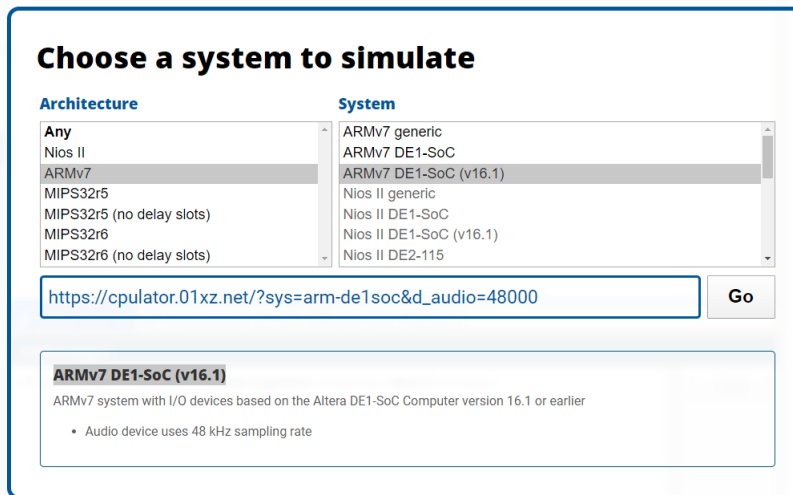


Figure 3: ARMv7 DE1-SoC (v16.1)

- 3- In the Editor panel select ARMv7 language (see Figure 4). Write all of your subroutines and functions in the Editor panel and save it with the desired name as specified in this Lab instruction.

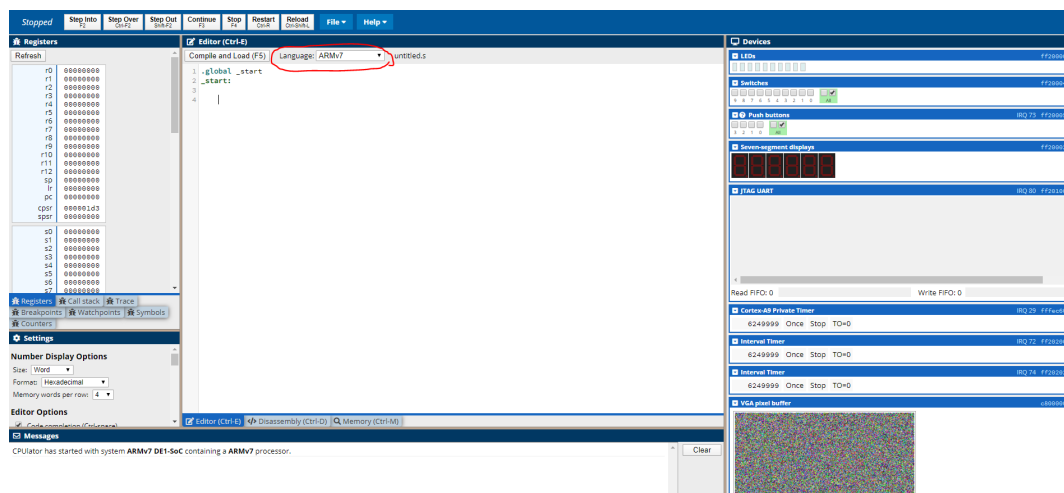


Figure 4: ARMv7 DE1-SoC (v16.1)

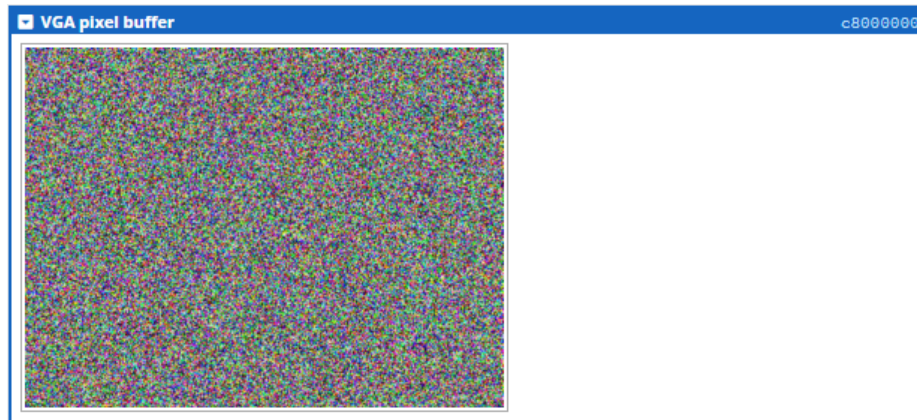
- 4- **Compile and Load** your code. If no errors occurred, start running your code using **Continue**. You may also need to debug your code using **Step Into**, **Step Over** or **Step Out** options available on the top-leftmost on the screen.

5- If for some unknown reason you encounter errors during debugging your code, you can simply ignore these errors by deselecting the features from the **Debugging Checks** in the **Settings** panel. Pay attention to the errors you are receiving. The simulator has a powerful documentation that you can access it by clicking on the errors. Read the documentation carefully before deselecting any features from the **Debugging Checks**.

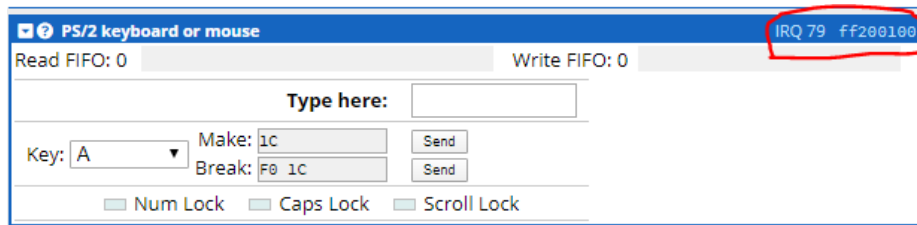
6- You only need to interact with the **Push buttons**, **VGA pixel buffer** and **PS/2 keyboard** from the **Device** panel. The Device panel is located on the rightmost of your screen. Only use the PS/2 keyboard with the `0xFF200100` base address (see Figure 5).



(a)



(b)



(c)

Figure 5: (a) Push buttons (b) VGA and (c) PS/2 keyboard panels

## 4 Grading

This lab will be conducted and graded **individually**. The TA will ask to see the following deliverables during the demo (the corresponding portion of the grade for each is indicated in brackets):

- VGA (50%)
- P/2 Keyboard (30%)

Full marks are awarded for a deliverable only if the program functions correctly and the TA's questions are answered satisfactorily.

A portion of the grade is reserved for answering questions about the code. Full marks are awarded for a deliverable only if the program functions correctly and the TA's questions are answered satisfactorily.

Your final submission should be a **single compressed folder** that contains all the code files, correctly organized (.c, .h and .s). For this lab, there is no report to write.

This Lab will run for **two weeks**, from March 30th to April 14th and the demo must take place within these two weeks during your lab session. The code files must be submitted on the same time as the scheduled demo via mycourses. The code itself will not be marked, but we will check for any cases of plagiarism (e.g. code copied from another student, or copied from a prior year).