Lab 4: High level I/O - VGA, PS/2 Keyboard, and Audio

ECSE 324 - Computer Organization

Fall 2017

Introduction

In this lab will use the high level I/O capabilities of the DE1-SoC computer. 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
- Use the audio controller to play generated tones

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 two files **VGA.s** and **VGA.h** and place them in the correct folders. The code for the header file is shown in Figure 1.

```
1  #ifndef __VGA
2  #define __VGA
3
4    void VGA_clear_charbuff_ASM();
5    void VGA_clear_pixelbuff_ASM();
6
7    void VGA_write_char_ASM(int x, int y, char c);
8    void VGA_write_byte_ASM(int x, int y, char byte);
9
10   void VGA_draw_point_ASM(int x, int y, short colour);
11
12 #endif
```

Figure 1: Code for the VGA.h file

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.

VGA_write_char_ASM should write the ASCII code passed in the third argument to the screen at the (x,y) coordinates given in the first two arguments. 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]).

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 and y coordinates are valid, taking into account that two characters will be displayed.

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: Use suffixes 'B' and 'H' with the assembly memory access instructions in order to read/modify bytes/half-words

Simple VGA application

Build a C based application to test the functionality of the VGA driver. Write three functions as shown in Figure 2

```
 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 2: C functions used to test the VGA driver

Use the pushbuttons and slider switches as follows:

- **PB0 is pressed:** if any of the slider switches is on, call the *test_byte()* function, otherwise, call the *test_char()* function.
- **PB1 is pressed:** call the *test_pixel()* function.
- **PB3 is pressed:** clear the character buffer.
- **PB4 is pressed:** clear the pixel buffer.

2 Keyboard

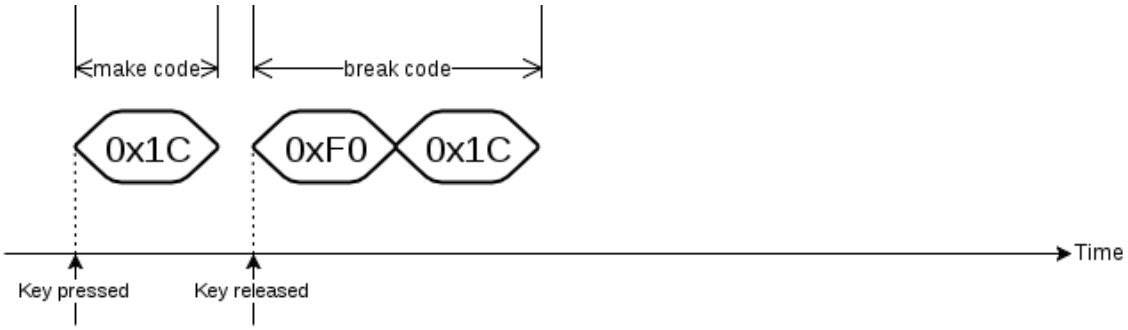
For this part, it is necessary to refer to section 4.5 (pp 45-46) in the De1-SoC Computer Manual.

Brief overview of the PS/2 Keyboard Protocol

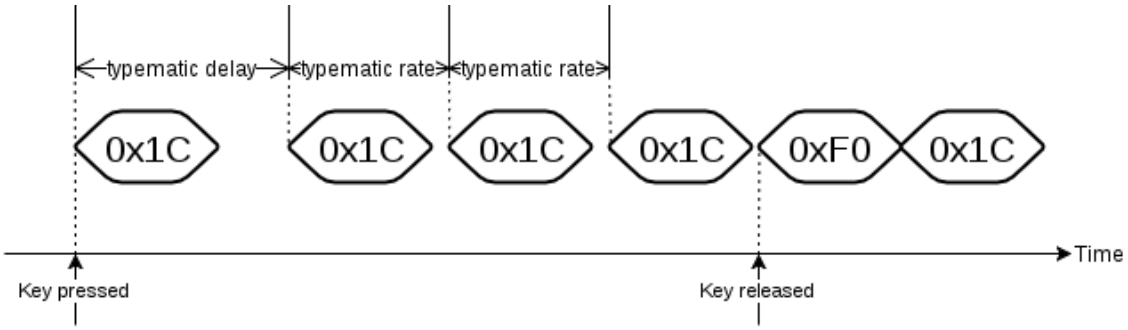
For the purpose of this lab, a very high level description of the PS/2 keyboard protocol is given. A more detailed description can be found at this link.

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 here.

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

PS/2 keyboard driver

Create two files **ps2_keyboard.s** and **ps2_keyboard.h** and place them in the correct folders.

For this lab, simply implement a subroutine with the following specifications:

- **Name:** *read_PS2_data_ASM*
- **Argument:** A char pointer variable *data*, in which the data that is read will be stored
- **Return type:** 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 char 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 from the VGA driver, 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), (5,1) 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 Audio

For this part, it is necessary to refer to section 4.1 (pp 39-40) of the De1-SoC Computer Manual

Write a driver for the audio port following the same procedure introduced so far. The driver should only have one subroutine. The subroutine should take one integer argument and write it to both the left and the write FIFO **only if** there is space in both the FIFOs (Hint: Use the value of WSLC and WSRC in the subroutine). The subroutine should return an integer value of 1 if the data was written to the FIFOs, and return 0 otherwise.

Use the driver in an application that plays a 100 Hz square wave on the audio out port. The frequency can be achieved by knowing the sampling rate of of the audio DAC. For example, if the sampling rate is 100 samples per second and a 2 Hz square wave is to be played, that means there are two complete cycles of the wave contained in 100 samples, so for 25 samples a '1' should be written to the FIFOs, and for 25 samples a '0' should be written to the FIFOs.

For this lab, find the sampling rate from the manual and calculate the number of samples for each half cycle of the square wave. Finally, write 0x00FFFFFF and 0x00000000 to the FIFO instead of '1' and '0'.

4 Grading

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 (30%)
- P/2 Keyboard (20%)
- Audio (20%)

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, which is awarded individually to group members. All members of your group should be able to answer any questions the TA has about any part of the deliverables, whether or not you wrote the particular part of the code the TA asks about. Full marks are awarded for a deliverable only if the program functions correctly and the TA's questions are answered satisfactorily.

Finally, the remaining 20% of the grade for this Lab will go towards a report. Write up a short (2-3) page report that gives a brief description of each part completed, the approach taken, and the challenges faced, if any. Please don't include the entire code in the body of the report. Save the space for elaborating on possible improvements you made or could have made to the program.

Your final submission should be a **single compressed folder** that contains your report and all the code files, correctly organized (.c, .h and .s).

This Lab will run for **two weeks**, from November 6th to November 17th. The report for Lab 4 is due by 11:59 pm, November 24th.