**Linux on DE1-SoC board:**

1. Download a Linux image for the DE1-SoC board from one of the three following websites:

https://www.intel.com/content/www/us/en/developer/topic-technology/fpga-academic/materials-sd-card-images.html

https://fpgacademy.org/courses.html

<https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836&PartNo=4>

After testing every single Linux image for the DE1-SoC board, I would recommend getting ‘Linux LXDE Desktop’ from Tersasic’s website. For more information regarding Linux images, please see section 2.3 where I described in more detail the issues I encountered while trying all the Linux images.

1. Unzip the Linux image then plug the SD card into your computer and launch Win32 Disk Imager. Select ‘de1soc\_lxde\_1604.img’ and the device letter that corresponds to your SD card. Click ‘Write’.
2. The next step is to boot Linux on the DE1-SoC board. Before inserting the microSD, card and turning on the board, make sure that the MSEL configuration is set correctly. The Mode Select (MSEL) switches can be found on the back of the board. There are three different pin settings that you can choose from. The default setting of the DE1-SoC board comes with the MSEL [5:0] set to 100100. For loading a Linux image with console (no GUI), the MSEL [5:0] has to be set to 010100. If the user chooses to load a ‘Linux console with frame buffer’ or ‘LXDE Desktop’, MSEL [5:0] will have to be set to 000000. Depending on the rev of the board, some might have 5 or 6 MSEL switches. The last switch (MSEL5) does not have a default therefore, it can be set to 0 or 1, it doesn’t matter.
3. After configuring the MSEL switches you can insert your microSD card and turn on the board to boot Linux. If you have ‘Linux console with frame buffer’ or a Linux desktop, you have to connect a VGA desktop, keyboard and mouse to communicate with the GUI. If you have a Linux console, the user will communicate through the command line interface (CLI) via UART terminal (see step 5).
4. Download Putty then connect the DE1-SoC board to you PC using the micro-USB cable. If using Windows, go to device manger to check what number COM port was assigned to the USB to UART cable. Once the COM number has been determined, open the main Putty window. On the main page ‘Basic operations for your PuTTY session’, specify the ‘serial line’ (ex: COM3) and delete the default ‘speed’ (leave this space empty for now.) On the left-hand side there is a hierarchy of categories, select ‘serial’ and it should take you to another page called ‘Options controlling local serial lines’. Here, double check that ‘Serial line to connect to’ has been set correctly (ex: COM3). Change the ‘Speed (baud)’ to 115200 and for the ‘Flow control’ select ‘None’. After all of the settings have been set, press ‘Open’ to start the terminal. Once you are connected to the CLI, press the WARM RESET button on the DE1-SoC board. After pressing the button, you will see it booting. Once you are logged in as the ‘root’ user you are good to go.
5. After booting Linux it’s a good idea to upgrade/update your system by using the following commands**: sudo apt-get update** and **sudo apt-get upgrade**

!! Internet connection – when using the Linux Desktop (with GUI), you shouldn’t have any problems with the internet connection, simply plugging in the ethernet cable is sufficient. If you have a Linux console you will have to manually do it:  
Step 1: Open /etc/network/interfaces using text editor of choice (ex: VI)  
Step 2: Write the following in the file (for eth0 with dhcp) :

# The loopback network interface  
auto lo eth0  
iface lo inet loopback

# The primary network interface  
iface eth0 inet dhcp

Step 3: Save then execute: /etc/init.d/networking restart  
For more information see: <https://askubuntu.com/questions/214170/whats-the-default-etc-network-interfaces>

Afterwards reboot the device: **reboot -h now**You might also want to shut it down: **shutdown -h now**(Note: Do not shut down the system by just turning of the power since it could corrupt the file system!)

Use the following commands to check the internet connection: **ping google.com** (or any website of choice) or **nslookup google.com**. The ping command needs to receive the same number of packets that were transmitted for it to be successful and the nslookup command should output a list that includes: server, address and name. If you get an output like ‘connection timed out; no servers could be reached’ it means that there is no internet connection.

**VGA graphics using DE1-SoC board:**

The Intel DE1-SoC Computer System features two components: The Hard Process System (HPS) and Cyclone V SoC chip. The HPS includes an ARM Cortex A9 dual-core processor, a DDR3 memory port, timers and other I/O peripherals, while the Cyclone V FPGA contains two Intel Nios II soft processors, I/O peripheral ports for audio and video, timers, JTAG ports, an on-chip memory and parallel ports for switches, LEDs and 7-segment display.

All of the I/O peripherals can be accessed via its address range since the processor accesses the I/O peripherals as memory mapped devices.

For my project I will use only a few I/O peripherals, to find the full list see: Intel’s DE1-SoC Computer System with ARM Cortex-A9 documentation.

**HPS memory components:**

DDR3 memory: 1GB, organized as 256M x 32 bits, is accessible using word accesses (32 bits), halfword and bytes. It is mapped in memory as: 0x0000 0000 -> 0x3FFF FFFF.

Small 64kB on-chip memory in each ARM A9 processor: 64KB, organized as 16K x 32 bits, mapped in memory as: 0XFFFF 0000 -> 0xFFFF FFFF.

**Cyclone V FPGA components:**

SDRAM: 64MB, organized as 32M x 16 bits, is accessible using word (32 bit), halfword (16 bit) or byte operations. It is mapped in memory as: 0xC000 0000 -> 0xC3FFF FFFF.

On-chip memory: 256KB, organized as 64K x 32 bits (4 bytes). The on-chip memory is used as a pixel buffer for the video-in and video-out ports. It is mapped in memory as: 0xC800 0000 -> 0xC800 FFFF.

On-chip memory character buffer: 8KB, organized as 8K x 8 bits. It is used as a character buffer for the video-out port and is mapped in memory as 0xC900 0000 -> 0xC900 1FFF.

Red LED parallel port (LEDR9-0): The ten red LEDs on the DE1-SoC board are driven by an output port. It is a 32-bit data register mapped at the address 0xFF20 0000, however, only 10-bits are used and the upper 22 bits are not. The register can be written or read using word (32 bit) accesses.

For my project I will use the on-chip memory as a pixel buffer for drawing various shapes, the on-chip memory character buffer for displaying characters on the VGA display and the red LED parallel port for LED control.

**Header file:**

Mapping the VGA pixel buffer:

#define FPGA\_ONCHIP\_BASE 0xC8000000  
#define FPGA\_ONCHIP\_END 0xC803FFFF  
#define FPGA\_ONCHIP\_SPAN 0x00040000

The span is calculated as following: 256KB on-chip memory = 28 KB = 28 x 210 B = 218 B = 0100 0000 0000 0000 00002 = 4000h

Mapping the character buffer:

#define FPGA\_CHAR\_BASE 0xC9000000   
#define FPGA\_CHAR\_END 0xC9001FFF  
#define FPGA\_CHAR\_SPAN 0x00002000

The span is calculated as following: 8KB on-chip memory character buffer = 213 B = 0010 0000 0000 00002 = 2000h

Mapping the red LED parallel port:

#define HW\_REGS\_BASE 0xff200000  
#define HW\_REGS\_SPAN 0x00005000  
#define LEDR\_BASE 0x00000000

**Pixel buffer:**

The pixel buffer is stored in the FPGA on-chip memory. The resolution of the VGA controller is 320 x 240 pixels, where each pixel is 16 bits (2 bytes) long. Each pixel holds its coordinates (starting at x=0, y=0) and colour. Please note that one pixel from the pixel buffer is not the same as one pixel from the monitor. Assuming that the monitor is 1920x1080p, that would mean that one pixel from the pixel buffer equals to 6x4.4 pixels from the display!

The pixel buffer contains 512 pixels per line, but, since each pixel is 2 bytes long, the resolution is 1024 bytes. Out of these 512 pixels, only the first 320 pixels are accessible. The area from pixel 320 to pixel 512 is non-accessible by the VGA, this means that even if you try to write to this area, you will not see any output on the VGA display. The pixel buffer has a total of 240 lines.

Ex: starting from the pixel buffer base address - 0xC8000000, the address of pixel x=1, y=0 would be 0xC8000000 + 00102 = 0xC8000002. See figures ...

Experiment: As specified, each pixel is on 2 bytes. But, what if you were to add ‘1’ to ‘vga\_pixel\_virtual\_base’ instead of ‘2’? Assuming that the pixel is orange, instead of seeing an orange pixel shifted by one to the right, you’ll see 2 pixels side by side. One pixel would be red and the other yellow, which mixed in RGB make orange. The on-chip memory (pixel buffer) is organized as 64K x 32 bits, even though the size of one pixel is 16 bits. This is because each pixel is replicated in both the x and y dimensions when being displayed on the VGA screen.

512 pixels (1024 bytes)

320 pixels - actual resolution (640 bytes)

Non-accessible by VGA

. . . . . . . . . . .

. . . . . . . . . . .

240 lines

239

0

320 pixels (640 bytes)

511 pixels

y

x

319 320

0 1 2 3 4

Non-accessible by VGA

. . . . . . . . . . .

VGA

512 pixels (1024 bytes)

Other examples:

(vga\_pixel\_virtual\_base + (319x2)) – top right pixel on the VGA display.

(vga\_pixel\_virtual\_base + (500x2)) – no visible output on the VGA display.

(vga\_pixel\_virtual\_base + 1024 \* 2) – left-most pixel on the second line

(vga\_pixel\_virtual\_base + 1024 \* 239) – pixel on the bottom left corner

(vga\_pixel\_virtual\_base + 1024 \* 239 + (319x2)) – pixel on the bottom right corner  
  
\* Where 1024 is calculated as: (511x2)+2  
\* And vga\_pixel\_virtual\_base is: base address of the pixel buffer

Pixel buffer colour:

The pixel buffer colour is on 16 bits. The colour is of type RGB where 5 bits is blue, 5 bits red and 6 bits green. See figure ...

24 23  22 21  20

15 11 10 5 4 0

<5> <6> <5>

red green blue

16 bits

. . . . . . . . . . . . . .

Full red: 1111 1000 0000 00002 = F800h  
Full green: 0000 0111 1110 00002  = 0FE0h  
Full blue: 0000 0000 0001 11112  = 001Fh  
Pink colour: 1111 0000 0000 11112  = F00Fh

**Character buffer:**

The character buffer is stored in the FPGA ‘on-chip memory character buffer’. It has a resolution of 80x60 characters, where each character is 8x8 pixels. Please note that a ‘character buffer’ pixel is not the same as a ‘pixel buffer’ pixel (one character is much larger than 8x8 pixels from the ‘pixel buffer’) !

Characters are stored using their ASCII code (ASCII is 8 bits = 1 byte). When a character is displayed on the VGA screen, the ASCII code is automatically converted to a pattern. The pattern is created using pixels and the font is built-in, which the user is unable to change. Similarly to the pixel buffer, the character buffer also has an area that is non-accessible by the VGA, this means that even if you try to write to this area, you will not see any output on the VGA display. See figure….

Ex: The first character starts at location x=0, y=0. Starting from the character buffer base address - 0xC9000000, the address of character x=1, y=0 would be 0xC9000000 + 00012 = 0xC9000001.

128 bytes line stride

. . . . . . . . . . .

. . . . . . . . . . .

60 characters

59

0

80 characters (640 pixels)

y

x

0

Non-accessible by VGA

. . . . . . . . . . .

0 1 2 78 79

128 129 130 206 207

7552 7553 7554 7630 7631

. . . . . .

. . . . . .

. . . . .

. . . . .

1

127 byte

79

2

**Issues:**

1. Most Linux images for the DE1-SoC board had no internet connection because the versions were too old.

The best Linux image that I tried was ‘Linux LXDE Desktop’ from the Terasic website. Since it has a GUI it’s advised to use a fan to control the temperature levels for the DE1-SoC board. It has internet connection.

The ‘Linux Ubuntu Desktop’ from the Terasic website also works, however it consumes too many resources and therefore is quite slow. It also overheats the board.

I manged to establish an internet connection with the Linux Console from Intel’s website (<https://fpgacademy.org/courses.html>), however was unable to install/build OpenCV. This version is Ubuntu 3.8, which is even older than the ‘end of life’ Ubuntu versions (the last on the list being Ubuntu 4.10). See: <https://wiki.ubuntu.com/Releases>

The rest of the Linux images from both Intel’s and Terasic’s website are not good. I could not establish a proper internet connection (although the ping and nslookup command seemed to work), I was unable to download text editors or connect to Git.

1. Git error when cloning repository to Linux:

error: server certificate verification failed. CAfile: /etc/ssl/certs/ca-certificates.crt CRLfile: none while accessing <https://github.com/cristinavasiliu/FPGALinuxVGAGraphics.git/info/refs>

SOLUTION: export GIT\_SSL\_NO\_VERIFY=1  
Then continue with git clone https//: ...........

For more info visit: <https://stackoverflow.com/questions/21181231/server-certificate-verification-failed-cafile-etc-ssl-certs-ca-certificates-c/69403278#69403278>

1. OpevCV is really big in size, partition was 4GB, had to resize the partitions to allocate the deallocated space to my partition, without corrupting the existing data.

Make sure you clone original SD card in case you corrupt it! Example:

df-h  
dd if=/dev/sda of=/dev/sdb status=progress

After you copied to the new drive, you need to extend the 4GB Linux partition into the free (unallocated) area, using disk utility in Linux.  
  
Resize the Linux file system:

df-h – (returns path to every dive and storage (/dev/sda1))

Unmount the drive if it is mounted

sudo resize2fs /dev/sda1

1. You need a fast SD when building cv2, otherwise building could take a whole day!

Bibliography

Urmeaza: Comet, square, image and accelerometer.