Linux on FPGA + VGA graphics

By Cristina Vasiliu

1 Introduction

Your introduction goes here! Simply start writing your document and use the Recompile button to view the updated PDF preview.

2 Linux on DE1-SoC board

2.1 Installing and running 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 4 where I described in more detail the issues I encountered while testing all the images.

2. 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'.

3. 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.

4. 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).

5. 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.

6. 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**

2.2 Establishing a 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:

1. Open /etc/network/interfaces using text editor of choice (ex: VI)
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
```

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.

3 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.

3.1 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.

3.2 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 lower 10-bits are used, the upper 22 bits being unused. The register can be written or read using word (32 bit) accesses.

For my project I used 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.

3.3 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 = 2^8 KB = 2^8 x 2^{10} B = 2^{18} B = $01000000000000000000_2 = 4000_h$

- 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 = 2^{13} B = $0010000000000000_2 = 2000_h$

- Mapping the red LED parallel port:

```
#define HW_REGS_BASE          0xff200000
#define HW_REGS_SPAN          0x00005000
#define LEDR_BASE             0x00000000
```

3.4 Pixel buffer

The pixel buffer is stored in the FPGA on-chip memory, each pixel being stored on 16 bits. The pixel buffer provides an image resolution of 320 x 240 pixels. Since the resolution of the VGA controller is 640 x 480, the internal logic replicates the pixels in both the x and y dimensions when being displayed on the VGA screen. Pixel coordinate x=0, y=0 represents the top-left corner of the screen. 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 line stride is 1024 bytes (equivalent to 512 pixels). Out of these 512 pixels 'per line', 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 number of 240 lines.
Its memory organization is 64K x 32 bits (equivalent to 64k x 2 pixels).
(See figure 1 and 2).

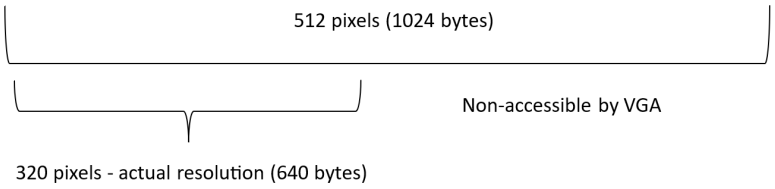

Figure 1: Pixel buffer line resolution

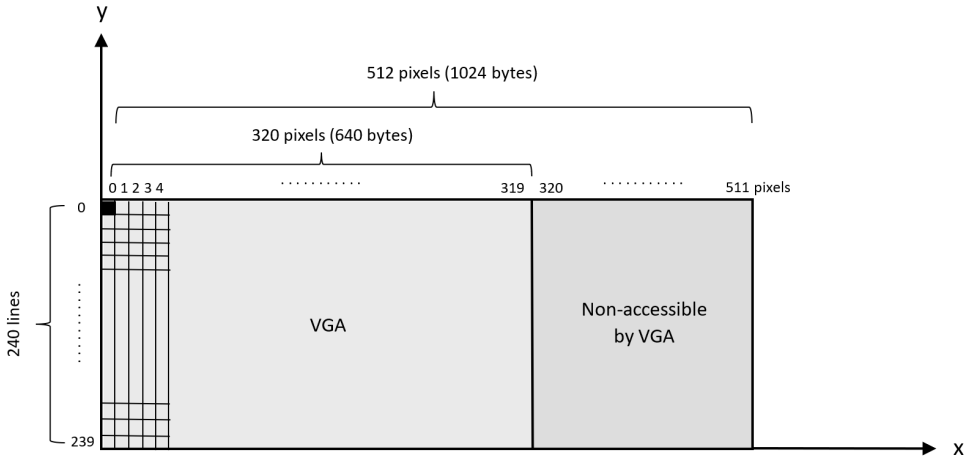

Figure 2: Pixel buffer addressing

Example 1: starting from the pixel buffer base address - 0xC8000000, the address of pixel x=1, y=0 would be 0xC8000000 + 0010_2 = 0xC8000002.

Example 2:
(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

Example 3: 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.

3.4.1 Pixel encoding

The pixel encoding is of type RGB565, meaning that the top 5 bits represent the red colour, middle 6 bits the green colour and bottom 5 bits the blue colour respectively. (See figure 3).

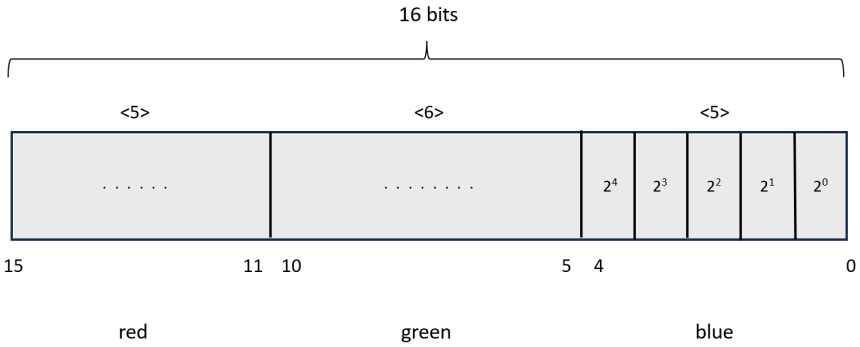

Figure 3: Pixel buffer encoding

Examples:
Full red: 1111 1000 0000 00002 = $F800_h$
Full green: 0000 0111 1110 00002 = $0FE0_h$
Full blue: 0000 0000 0001 11112 = $001F_h$
Pink colour: 1111 0000 0000 11112 = $F00F_h$

3.5 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.

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 (each 8-bit ASCII character written in the memory will be converted into a 8 x 8 pixel character on the VGA screen). 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 (80-byte line content vs 128-byte line stride), this means that even if you try to write to this area, you will not see any output on the VGA display. (See figure 4)

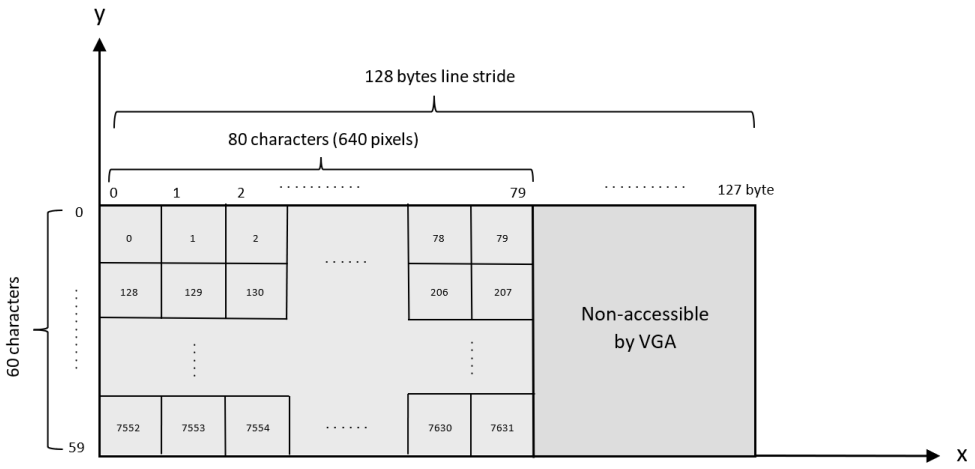

Figure 4: Character buffer

Examples:
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 + 0001_2 = 0xC9000001.
For x=0, y=1 the adress is: 0xC9000000 + 0x80 = 0xC9000080.

4 Obstacles encountered

1. (SOLVED) The Linux images for the DE1-SoC board are old (built between 2014 - 2016), they don't have much support therefore I encountered many issues while installing tools. For example, some of them had no internet connection, others could not install PIP or update Python to Python3 required by many packages like OpenCV.

   The least problems I encountered was with 'Linux LXDE Desktop' from the Terasic website (version 4.5). Since it has a GUI it's resource hungry and the FPGA gets hot, so it's advised to use a fan to control the temperature levels. There is no access from the top of the board for a fan therefore the board was placed on top of a laptop cooler.

   The 'Linux Ubuntu Desktop' from the Terasic website has the same problems with overheating but also Linux seems to run slower on it.

   The other Linux image I tried was the Linux Console from Intel's website (https://fpgacademy.org/courses.html). I manged to establish an internet connection, however I was unable to install many tools like Python3 required by 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 seem to be too old (updates/upgrades won't work). For example, I could not establish a proper internet connection although the ping and nslookup command seemed to work, e.g., I was unable to download text editors or connect to Git.

2. (SOLVED) 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

3. (SOLVED) Linux partition/file-system is set to 4GB, so even if you have a bigger SD card, Linux won't be able to see or use the space. I had to resize the Linux file system in order to make use of the full SD card capacity so I could install tools e.g. OpenCV. How to fix this:

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

4. (SOLVED) Encountered many problems while building/installing OpenCV.

Prerequisites: OpenCV is really big in size (> 5 GB), therefore you need a fast SD card when building as it could take 24 hours! Also, it's recommended to use use a 16 GB SD card or larger (I used a 64 GB SD card). Before building OpenCV, ensure that the Linux file-system/partition makes use of the full SD card capacity (see issue no. 3).

I followed the official instructions for installation on Linux from the OpenCV website. See: https://docs.opencv.org/4.x/d7/d9f/tutorial_linux_install.html I followed the 'Quick start - build core modules' instructions. I managed to install the zip file, unpack it then build however got an error when installing:

```
[ 98%] Building CXX object modules/python2/CMakeFiles/opencv_python2.dir/__/src2/cv2.cpp.o
^[[^[[Ac++: internal compiler error: Killed (program cc1plus)
Please submit a full bug report,
with preprocessed source if appropriate.
See <file:///usr/share/doc/gcc-5/README.Bugs> for instructions.
modules/python2/CMakeFiles/opencv_python2.dir/build.make:62: recipe for target 'modules/python2/
make[2]: *** [modules/python2/CMakeFiles/opencv_python2.dir/__/src2/cv2.cpp.o] Error 4
CMakeFiles/Makefile2:3965: recipe for target 'modules/python2/CMakeFiles/opencv_python2.dir/all'
make[1]: *** [modules/python2/CMakeFiles/opencv_python2.dir/all] Error 2
Makefile:160: recipe for target 'all' failed
make: *** [all] Error 2
root@DE1_SoC:~/Desktop/build# [A
[ 98%] Building CXX object modules/python2/CMakeFiles/opencv_python2.dir/__/src2/cv2.cpp.o
^[[^[[Ac++: internal compiler error: Killed (program cc1plus)
Please submit a full bug report,
with preprocessed source if appropriate.
See <file:///usr/share/doc/gcc-5/README.Bugs> for instructions.
modules/python2/CMakeFiles/opencv_python2.dir/build.make:62: recipe for target 'modules/python2/
make[2]: *** [modules/python2/CMakeFiles/opencv_python2.dir/__/src2/cv2.cpp.o] Error 4
CMakeFiles/Makefile2:3965: recipe for target 'modules/python2/CMakeFiles/opencv_python2.dir/all'
make[1]: *** [modules/python2/CMakeFiles/opencv_python2.dir/all] Error 2
Makefile:160: recipe for target 'all' failed
make: *** [all] Error 2
root@DE1_SoC:~/Desktop/build#
```

At first I thought that the error was caused because there wasn't enough space on the SD card, so I resized the Linux file system to make use of the full SD card capacity. I removed the build that was previously created then repeated the whole process again. However, this time it gave a different error when building:

```
[ 98%] Building CXX object modules/gapi/CMakeFiles/opencv_test_gapi.dir/test/common/gapi_operato
[ 98%] Building CXX object modules/gapi/CMakeFiles/opencv_test_gapi.dir/test/common/gapi_compoun
[ 98%] Building CXX object modules/gapi/CMakeFiles/opencv_test_gapi.dir/test/common/gapi_video_t
[ 98%] Building CXX object modules/gapi/CMakeFiles/opencv_test_gapi.dir/test/gapi_fluid_roi_test
[ 98%] Linking CXX executable ../../bin/opencv_test_gapi
[ 98%] Built target opencv_test_gapi
Scanning dependencies of target gen_opencv_python_source
[ 98%] Generate files for Python bindings and documentation
WARNING! Typing stubs can be generated only with Python 3.6 or higher. Current version sys.versi
WARNING! Typing stubs can be generated only with Python 3.6 or higher. Current version sys.versi
Note: Class cv::Feature2D has more than 1 base class (not supported by Python C extensions)
      Bases:  cv::Algorithm, cv::class, cv::Feature2D, cv::Algorithm
      Only the first base class will be used
Note: Class cv::detail::GraphCutSeamFinder has more than 1 base class (not supported by Python C
      Bases:  cv::detail::GraphCutSeamFinderBase, cv::detail::SeamFinder
      Only the first base class will be used
```

```
[ 98%] Built target gen_opencv_python_source
Scanning dependencies of target opencv_python2
[ 98%] Building CXX object modules/python2/CMakeFiles/opencv_python2.dir/__/src2/cv2.cpp.o
c++: internal compiler error: Killed (program cc1plus)
Please submit a full bug report,
with preprocessed source if appropriate.
See <file:///usr/share/doc/gcc-5/README.Bugs> for instructions.
modules/python2/CMakeFiles/opencv_python2.dir/build.make:62: recipe for target 'modules/python2/
make[2]: *** [modules/python2/CMakeFiles/opencv_python2.dir/__/src2/cv2.cpp.o] Error 4
CMakeFiles/Makefile2:4011: recipe for target 'modules/python2/CMakeFiles/opencv_python2.dir/all'
make[1]: *** [modules/python2/CMakeFiles/opencv_python2.dir/all] Error 2
Makefile:160: recipe for target 'all' failed
make: *** [all] Error 2
root@DE1_SoC:~/Desktop/build#
```

It seemed as though the error was coming from the compiler. The compiler version used when building the Linux kernel was different from the one I was using when building OpenCV. I thought that this might be the issue, so I downgraded the gcc/g++ compiler I was using to match the other one (4.8). This didn't solve the issue, I got an error at around 38% when building.

SOLUTION: OpenCV is very memory heavy, the DE1-SoC has a 1GB DDR3 memory which is not enough to build and install OpenCV. The solution is to increase the swap size from the default 100MB to 2GB:
sudo dphys-swapfile swapoff
sudo sed -i 's:CONF_SWAPSIZE=.*:CONF_SWAPSIZE=2048:g'
/etc/dphys-swapfile
sudo reboot

Afterwards, build and install OpenCV by using the following commands:
Install prerequisits:
sudo apt install -y g++ sudo apt install -y cmake
sudo apt install -y make
sudo apt install -y wget unzip
Download sources:
wget -O opencv.zip https://github.com/opencv/opencv/archive/4.x.zip
unzip opencv.zip
mv opencv-4.x opencv
Configure and build:
mkdir -p build && cd build
cmake ../opencv
make -j4

References