ECE532 Final Design Report: Can it Run DOOM?

Oltan Sungurlu
Kushagra Goel

Important Note: Our third member, Ghamr Saeed, has dropped out of the course prior to the Final Demo. While he did work on this project, such as doing research and attempting initial implementations, nothing designed by him made it into the final Vivado design/software stack. We believe it is fair to say that this has been a 2-man project, and we ask that grading of the report in terms of project complexity or completeness, as well as the Final Demo be considered in this light.

- This section will be removed for the github version of the report.

Overview

For this project, we tried to answer the question - “Can the Nexys 4 DDR Board, or Nexys Video, run Doom?”.
Elaborating on this: Can the Nexys Boards run Doom:

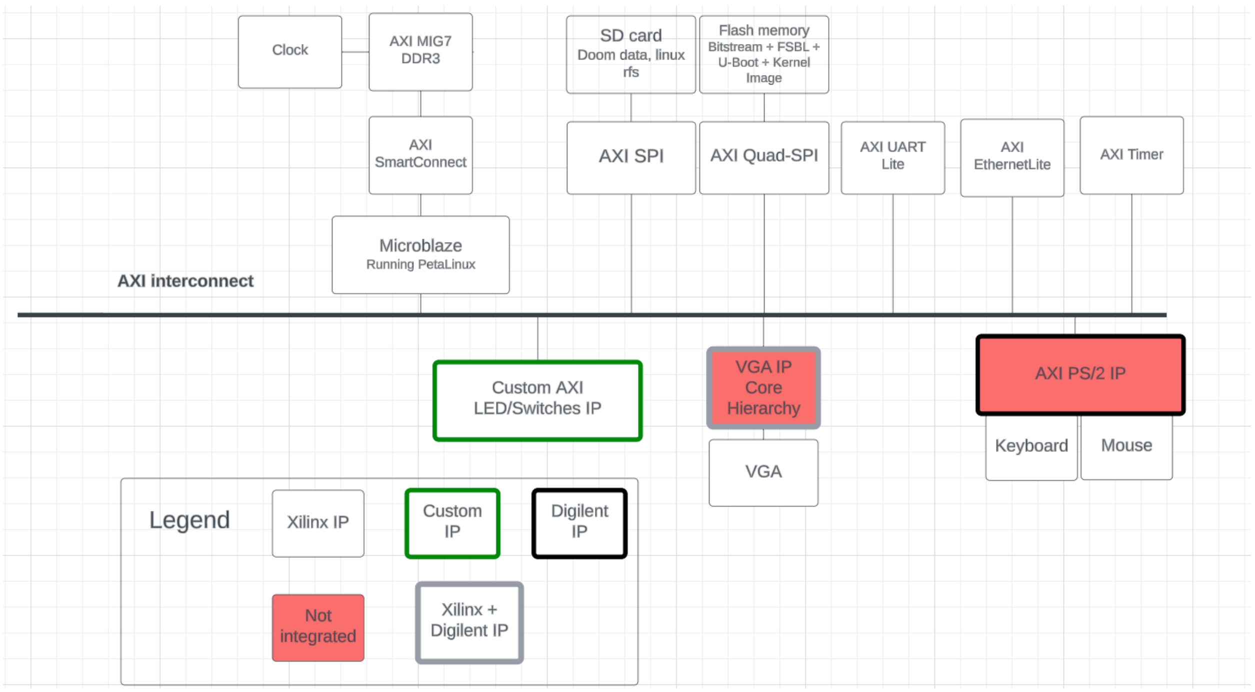
- At a playable framerate
- On a MicroBlaze soft processor
- While taking input from a mouse and keyboard
- Outputs of VGA Display and Audio
- With an accelerator to render frames in hardware
- With networking capabilities (as a stretch goal)?

We thought this was a good project idea as it used many peripherals, so it would be a great learning experience for the team. It also pitted the FPGA against a general-purpose CPU, so we had a chance to be able to show accelerated speeds by making a custom-purpose hardware IP.

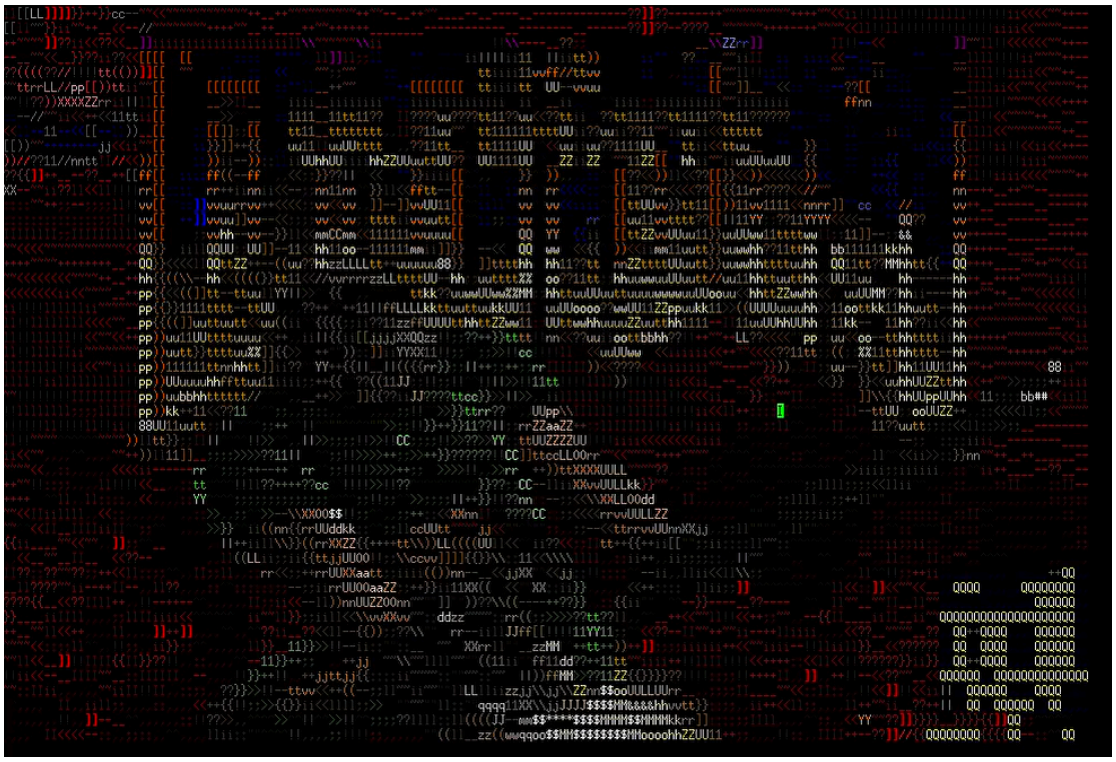
With the suggestion from our TA, Omkar, we decided to run DOOM on PetaLinux to achieve these goals. The idea behind this was that it would make graphics drivers, other software complications, and the interaction between peripherals easier to implement due to the game being originally for PC and having open-source code to be run on Linux.

As you will see later in the report, this could not be further from the truth. The task of running PetaLinux on a pure FPGA like Artix-7 has been monumental. On top of this, it did not help with integration at all.

Block Diagram



Outcome



In the end, we were able to run PetaLinux on both the Nexys 4 DDR board and the Nexys 4 Video board. Furthermore, we did not need to boot PetaLinux from a host PC each time using the video board via JTAG. We created an environment where a First-Stage Boot Loader is embedded into the bitstream, which is configured in the QSPI flash memory on the board, along with U-Boot bootloader, a boot script that loads the kernel image to RAM and boots the system, the kernel image, and the system device tree file (system.dtb). This allowed us to boot Linux by simply powering the board on.

This was a full Linux system running on the MicroBlaze processor, with an SD card having the root filesystem. We were able to configure a laptop to work as a router for the MicroBlaze, which allowed us internet access on the board as well as scp and ssh via Ethernet. This was also supposed to be used for the networking component of DOOM. This shortened our development cycle, as we did not need to move the SD card and reboot the system each time we wanted to transfer files from/to the board.

We are able to run ASCII-doom, which is basically the original source code for DOOM, except that the graphics output is converted into ASCII for the terminal. However, it was running at 0.2 FPS, and the game took a long time to load after selecting difficulty.

We were also able to run headless-doom, a version which would draw frames from a 20 min gameplay, but not render them, on the MicroBlaze processor. This took 55 minutes to run, which meant that the game loop itself was running 3x slower than the absolute minimum speed it had to. This meant no matter how much we accelerated the rendering with a custom IP, the game loop could not keep up. This was revealed way too late into the project, and it meant that MicroBlaze wasn't powerful enough to run Doom on a Linux environment. We would need to run it directly on MicroBlaze to achieve the desired speed (as other profilers of Doom have done using headless-doom.)

Our custom IP allowed us to interface with the on-board switches as controls and LED lights to indicate score. This was integrated into a game of classic snake, programmed in-house to utilize the custom IP, which was shown during the demo. This was a last-minute fix-up in order to have a working demo as we could not integrate PS/2 into PetaLinux.

In terms of having a basic implementation of peripherals, we were able to interface with a PS/2 keyboard, and receive key scancodes and display them on the 7 segment display. We got a VGA + PS/2 mouse demo working, and were able to display colors and patterns on a monitor, as well as the movement cursor with functional mouse buttons. However, we ran out of time before any of this could be integrated into the PetaLinux system. It should be mentioned that even if we had the time, it would've taken a lot of non-digital design related work to do the integration, primarily writing Linux device drivers.

Given more time, we would want to integrate the currently discrete parts of the project (VGA, PS/2 mouse and keyboard) into the game and possibly use high level synthesis to accelerate the slow functions of the game loop. Ultimately, we are sure that we would have succeeded given more time in this project - the proof of concept has been shown.

If we were to start over, we would not use PetaLinux and instead run the Doom game code directly on the MicroBlaze, since getting PetaLinux running was the biggest time sink in the entire project. On top of this, MicroBlaze is not even fast enough to run the game on top of running Linux. We had a hard time accepting this fact, but this is one project that would've been way easier as a hardware-only project.

Project Schedule

- Milestone 1
  - Research USB/PS2 interface methods and decide which to use
  - Research VGA, audio, and SD card interface
  - Research methods for implementing the renderer
  - **Actual**: Getting familiar with tools and project (purely research and experiments based)

- Milestone 2
  - Implement PetaLinux environment to be worked on
  - **Actual**: Implemented PetaLinux, work underway on keyboard integration, research underway on VGA component.

- Milestone 3
  - Implement USB/PS2 interface
  - **Actual**: SD card is detected and can read/write from it. Work underway on using SD card as root filesystem for PetaLinux(detection working, but not initializing). Keyboard & Mouse demo implementation completed. Research on interfacing custom IP block with PetaLinux.

- Milestone 4
  - Implement SD card interface
  - **Actual**: SD card working with PetaLinux, ASCII DOOM working, VGA on MicroBlaze demo working, DOOM porting research started. Decided to scrap Audio as there is very little support and we have enough things on our hands as it is.

- Milestone 5
  - Implement VGA interface
  - **Actual**: Board internet connection via PC working, scp+ssh via Ethernet working, headless-doom compiled, attempting to implement HDMI (unsuccessful) since VGA was not working on PetaLinux. Switched to Nexys Video to implement QSPI Flash of PetaLinux (22MB needed)

- Milestone 6
  - ~~Implement audio interface~~ Finalize implementation functionality within the game
  - **Actual**: PetaLinux running on Nexys Video, QSPI Flash works. Profiling attempts for DOOM underway. Still working on integration of VGA and PS/2 Keyboard/Mouse to PetaLinux.

- Milestone 7 (and the week leading to Final Demo)
    - Implement the hardware renderer
    - **Actual:** Could not integrate VGA and PS/2 Keyboard/Mouse to PetaLinux. Finalize the project by custom IP to read/write LED/Switches and create a basic snake game to show proof-of-concept.

We were doing very good early on in the project. Milestones were being met (although it took way more hours than anticipated), and we thought that we definitely would make it. However, the final integrations of everything is where it all crumbled and we had to settle for a demo that was way less ambitious than what we had in mind.

In retrospect, our goals were unrealistic for a 3 person group, and it was outright impossible for a 2 person group, which it ended up being.

Description of the Blocks

In both of the Vivado Projects (one for Nexys 4 DDR, one for Nexys Video), we have used Vivado 2023.2.

VGA Hierarchy:
https://www.rehsdonline.com/post/arty-vga-walkthrough This guide was followed to every detail. Ended up not being able to integrate it into PetaLinux, but it remains in the block design. We have tested the basic demo on the website so we assume full functionality in RTL.

Every existing IP from Xilinx and Digilent have been used in their latest versions. Grab all the Digilent IPs from https://github.com/Digilent/vivado-library .

IP Name	Version	Comment
AXI EthernetLite	3.0 (Rev. 28)	
AXI GPIO (all instances)	2.0 (Rev. 31)	Interrupt Disabled
AXI Interconnect	2.1 (Rev. 30)	
AXI Interrupt Controller	4.1 (Rev. 18)	Have to concat interrupt signals using Concat module

IP Name	Version	Comment
AXI PS/2	1.0 (Rev. 2)	
AXI Quad SPI	3.2 (Rev. 28)	SPI mode for SD card, QSPI mode for Flash.
AXI SmartConnect	1.0 (Rev. 21)	For MIG7 only, provides faster read/write.
AXI Timer	2.0 (Rev. 31)	
AXI Uartlite	2.0 (Rev. 33)	
AXI Video Direct Memory Access	6.3 (Rev. 17)	VGA Tutorial
AXI4-Stream to Video Out	4.0 (Rev. 17)	VGA Tutorial
Block Memory Generator	8.4 (Rev. 7)	
Clocking Wizard	6.0 (Rev 13)	
Concat	2.1 (Rev. 5)	Concat interrupt signals of all AXI IPs and input to AXI Interrupt Controller
Constant	1.1 (Rev. 8)	**IMPORTANT: HAVE TO DRIVE SD_RESET PIN LOW TO ENABLE SD CARD**
Dynamic Clock Generator	1.2 (Rev. 2)	VGA Tutorial
LMB BRAM Controller	4.0 (Rev. 23)	
Local Memory Bus (LMB) 1.0 (all instances)	3.0 (Rev 13)	
Memory Interface Generator (MIG 7 Series)	4.2 (Rev. 1)	Board Specific (DDR2 vs DDR3), config files available on Digilent.

IP Name	Version	Comment
MicroBlaze	11.0 (Rev. 12)	Setup with “Linux with MMU” preset. Further details can be found on PetaLinux on Microblaze guides online.
MicroBlaze Debug Module (MDM)	3.2 (Rev. 25)	
Processor System Reset	5.0 (Rev. 14)	
RGB to VGA output	1.0 (Rev. 3)	VGA Tutorial
Slice	1.0 (Rev. 3)	
Video Timing Controller	6.2 (Rev. 7)	VGA Tutorial

We fail timing on the Ethernet (pertains only to the Nexys 4 DDR project). However, this is a single path that upon further inspection was a false path. We did not change its constraints on the project due to switching boards, but everything worked fine! (pending QA approval.)

The testing procedure has mainly been whether we get crashes or not. The Linux stack running is already a good stress test - our MicroBlaze configuration and MIG7 failed instantly a couple of times, before we got to settings that resulted in no crashes.

Description of Our Design Tree

Github repository: https://github.com/oltansung/ece532-petalinux-doom

On the repository, we have the Vivado designs for both the Nexys 4 DDR, and the Nexys Video. Both of them are configured to be able to run PetaLinux. The Nexys 4 DDR project is older, and thus doesn't have some of the I/O we used later (GPIO and VGA hierarchy). However, it has Ethernet connection, which we did not implement in Nexys Video as they have different protocols.

In both directories, there's also a device-tree folder which contains the system's auto generated and user defined device trees. These are used in the PetaLinux flow and Oltan has a PetaLinux for the Nexys 4 DDR/Nexys Video guide on the repository to get started with that.

We also have the root filesystem used in Nexys Video only - it is a large file so a copy for Nexys 4 DDR is not provided.

Tips and Tricks

Note to future students of ECE532 when deciding on their project:

If you want to use PetaLinux as a part of your project, prepare to have one person fully dedicated to PetaLinux only, as it requires integration of every single IP block added. On top of that, the person also needs to know how every piece of custom IP works as well.

Overall, it does not make sense to use PetaLinux as the main running stack of the project. The integration work is too much, and there is barely any documentation for running PetaLinux on MicroBlaze - it is mainly intended to be run on SoC chips such as Zynq, Versal.

One good way to utilize PetaLinux might be if you want an Ethernet network connection and a way to read file system standards (EXT-4, FAT32, etc.) from your SD card. Linux would alleviate the need to establish such protocols, and you can directly load your image/program files from SD into register addresses using the "devmem" command, and read final outputs (not changing in high frequency) to transfer to another board/PC via Ethernet using C/Python scripts stored in the SD card.

Either way, trying to run PetaLinux on a pure FPGA, as we have realized in this project, takes too much attention away from actual Digital Design, such as creating custom IP blocks. On top of this, if the student in charge of PetaLinux does not have prior embedded development experience (i.e. running Linux on a Raspberry PI with GPIO connections, or running Linux on a microcontroller), it will take a lot of time - definitely way more than the time a course at UofT Engineering typically requires from a student.

If you still want to use PetaLinux after all the warnings, we suggest you read up on embedded Linux design trees, as it is essential in connecting I/O to Linux:
https://elinux.org/images/f/f9/Petazzoni-device-tree-dummies_0.pdf

We have also uploaded a PetaLinux (courtesy of Oltan) on the Nexys 4 DDR/Nexys Video guide on our github repo, which you should definitely check out in order to save quite a bit of time in your introduction to PetaLinux.

Video

We will leave our mid-presentation demo as that is the only recording we have available that looks cool - though it seems to capture the soul of the project pretty well:
ASCII-Doom on PetaLinux running on Nexys 4 DDR -
https://drive.google.com/file/d/1BfkxjuJC0R69rgOOqxmrgOtc47M6TyRe/view?resource
key

Also available on the github repo as a video file in case the link stops working in the future.