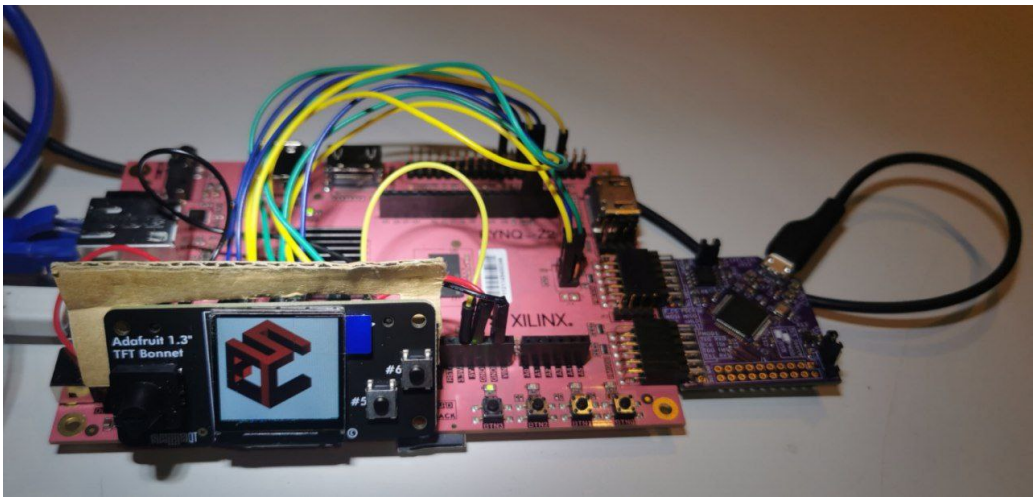
École Polytechnique Fédérale de Lausanne

Semester Project

DOOM in HEEP:
Implementation of the classic DOOM game on a fully in-house ASIC

By Ismael Tobias Frei



Project Advisor:
Prof. David Atienza

Project Supervisors:
Dr. Jose Miranda and Dr. Miguel Peon Quiros

STI IEL ESL
ELG 130 (Bâtiment ELG)
Station 11
CH-1015 Lausanne
June 21, 2024

Acknowledgments

I would like to thank Dr. Jose Miranda and Dr. Miguel Peon for their outstanding help as supervisors for the project. Their availability and knowledge of the subject matter greatly facilitated the advancement of the project. Furthermore, I would like to thank Ruben Rodriguez and Juan Sapriza for their technical insight regarding SPI communication, FPGA emulation, debugging with logic analyzers, and for organizing the X-HEEP agora, an event to exchange advancements made with X-HEEP.

Abstract

This report presents the research and development efforts undertaken to port the classic game DOOM to the X-HEEP platform. X-HEEP, a fully open-source RISC-V microcontroller, offers configurability for various hardware implementations, including simulation in software and virtualization using FPGAs. The project leverages the PYNQ-Z2 FPGA to emulate X-HEEP, integrating peripherals such as the Adafruit Bonnet for input and display. Despite significant progress in optimizing RAM usage and interfacing peripherals, DOOM is not yet fully functional on X-HEEP. This report details the challenges encountered, solutions implemented, and future work required to achieve the project's goals.

All the source code for this project can be found in the author's fork [4] of the original X-HEEP repository.

Contents

Chapter 1

Introduction

During the development of HEEPocrates, several critical questions emerged: What will the performance of HEEPocrates be? How well will it run various applications? And most compellingly: **Does it run DOOM?**

This project aims to port DOOM to the X-HEEP platform, an energy-efficient RISC-V microcontroller. Originally released in 1993, DOOM has been adapted for many hardware platforms due to its iconic status and technical demands. The latest implementation of X-HEEP, the HEEPocrates ASIC, presents a modern and intruiging platform for this challenge.

To achieve this, the project involves several key components:

1. Developing a display driver to enable output from a screen buffer in X-HEEP.
2. Porting DOOM to the X-HEEP platform on FPGA, involving modifications to interface files (video, sound, timers, SPI, etc.) and RAM optimization.
3. Further optimizing RAM usage to facilitate running DOOM on the HEEPocrates ASIC itself.

This report details the progress made, the challenges encountered, and the future work necessary to fully realize DOOM on the X-HEEP platform.

Chapter 2

Background and Research

This chapter provides essential background information for understanding the subsequent sections of the report, focusing on X-HEEP, HEEPocrates, PYNQ-Z2 FPGA, and the porting of DOOM to embedded systems.

2.1 X-HEEP and HEEPocrates

X-HEEP is an open-source RISC-V microcontroller known as the eXtendible Heterogeneous Energy Efficient Platform. It offers extensive configurability for various hardware implementations. X-HEEP can be fully simulated in software, emulated using FPGAs, and since 2024, implemented as an ASIC known as HEEPocrates [11].

HEEPocrates is a 65nm ASIC implementation of X-HEEP equipped with an Ibex core, 256kB of SRAM, and several peripherals. Designed for power efficiency, it is particularly suitable for biomedical wearables due to its compact size (see Figure 2.1) and low power consumption. The ASIC can operate at clock speeds of up to 470MHz [5].

Figure 2.1: HEEPocrates chip next to a 2 CHF coin

2.2 PYNQ-Z2 FPGA

X-HEEP can be fully implemented on the PYNQ-Z2 FPGA board. The current prototyping of X-HEEP into this FPGA is running at 15MHz with up to 512 kB of RAM. Despite its lower clock speed compared to HEEPocrates, it offers twice the RAM capacity.

2.3 DOOM on Embedded Systems

DOOM, released in 1993, revolutionized gaming as one of the first FPS (First Person Shooter) games. It introduced the DOOM Engine, capable of rendering 3D graphics with non-orthogonal walls and floors. The game's source code and a demo level were released as open source, allowing enthusiasts to modify and create new levels and port the game to various platforms. DOOM separates code from game data, with all levels stored in `.wad` files containing textures, sounds, and other game assets. During gameplay, level data is loaded into RAM, and graphics are rendered in a screen buffer with a fixed resolution of 320x200 pixels. The original DOOM code is optimized for 32-bit CPUs without floating-point operations, aligning well with X-HEEP's architecture.

DOOM has become a cultural touchstone with the meme "Does it run DOOM?" inspiring communities to find new devices capable of running the game [6, 9].

Two notable ports include versions for the Nordic Semiconductor nRF5340 [7] and EmbeddedDOOM [3], a lightweight version optimized for low-RAM devices and running on Linux. The theoretical absolute minimum RAM requirement of EmbeddedDOOM is still 384kB, not counting the `.wad` file, which is still too much by a factor of 1.5 for HEEPocrates.

For this project, the Nordic DOOM variant served as the foundation for porting DOOM to X-HEEP. Leveraging features like storing WAD files in flash memory and loading data on demand, significant reductions in RAM usage were achieved. Notably, even the smallest public demo level occupies 4.2MB of space.

Chapter 3

Peripheral Implementation

3.1 Adafruit Bonnet

All inputs and outputs are managed through the Adafruit 1.3" Color TFT Bonnet for Raspberry Pi [1]. This bonnet includes a 240x240 pixel TFT display, a joystick, and two buttons. The button inputs are read through a standard GPIO interface, while the display is driven by the ST7789V chip connected via SPI.

To integrate the Adafruit Bonnet, an example program (`example_adafruit_bonnet`) [4] was created to test its functionality and ensure compatibility with the board. This program will potentially be integrated into the X-HEEP GitHub repository for future projects involving input/output operations with the Adafruit bonnet.

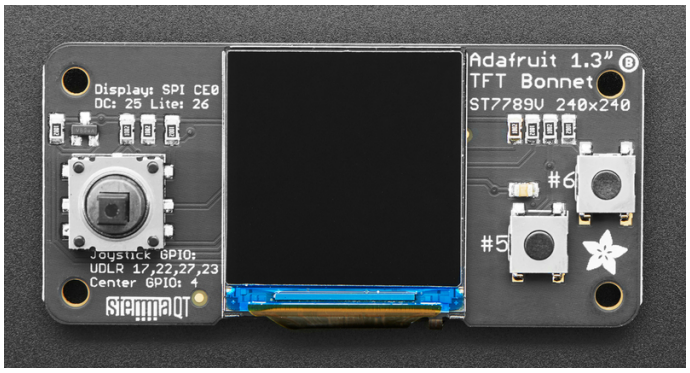


Figure 3.1: Adafruit 1.3" Color TFT Bonnet for Raspberry Pi [1]

3.1.1 Buttons

There are six buttons in total: a joystick on the left side of the display (UP, DOWN, LEFT, and RIGHT), and two buttons (A and B) on the right side of the display. They need to be initialized once and then read once at every loop in the game. The `example_adafruit_bonnet` implements them by reading the logic value of the six pins and filling in 3 arrays to determine their current state and if they are at positive or negative edge by comparing with their last state. Reading the buttons by polling aligns with DOOM's input handling. There is no hardware interrupt in DOOM.

Implementing button functionality was facilitated by the existing `example_gpio_intr` application from the x-heep repository [12].

3.1.2 Display Driver Development

The display module works by receiving an initializaiton sequence once after power on and then receiving information about the pictures to draw. To draw a picture it first requires to receive the X and Y coordinates of the pixels in the upper left and lower right corners of a rectangle and then the color of every pixel row-wise. This permits the user to chose, what portion of the display needs to be drawn. DOOM operates by rendering graphics to a screen buffer, which is then displayed on the screen. Therefore Doom redraws the whole display every time.

The integration of the Adafruit Bonnet's display proved more complex than expected. Originally designed for Raspberry Pi with a Python-based driver, the display uses the ST7789 chip, common in TFT displays. Several existing driver implementations exist for the ST7789 chip, including the Arduino ST7789 Fast library on GitHub [2], which served as a reference for this project.

Communication with the Display

The ST7789 communicates with the board via an SPI interface, utilizing the MOSI (Master Out Slave In), SCK (Serial Clock), and CS (Chip Select) pins. Notably, the ST7789 also requires a DC (Data/Command) line to distinguish between data and command transmissions, which poses a challenge on X-HEEP's implementations on FPGA and HEEPocrates:

In the X-HEEP implementation on the PYNQ-Z2 FPGA, the SPI interface is pre-implemented. It consists of the four standard signals MOSI, MISO (Master In Slave Out), SCK, and CS. However, the FPGA cannot directly control the DC signal through SPI. Thus the DC pin's behaviour needs to be manually emulated on a GPIO pin.
Initially, this didn't work and the display stayed black. After some experimentation with an Arduino, the problem was diagnosed to be the initialization sequence of the display. When the display was powered on, received the initialization sequence by pluging it into the Arduino and then switching the SPI cables to the FPGA, it was possible to display an image from the FPGA. A logic analyzer was used to debug the problem (see Figure 3.2).
In the end the problem turned out to be the DC pin:
When X-HEEP gives a command to the SPI module, it will execute it once the module is ready. The GPIO function, however, is instant. As a consequence, it can happen that the FPGA is sending a command, while the DC pin already signals, that it is sending data. Adding a delay between different initialization commands resolved the issue. Performance wise this is not a problem, as the initialization sequence is only played once.

This issue significantly delayed the project, compounded by an initial display unit with a bad contact on the display ribbon cable.

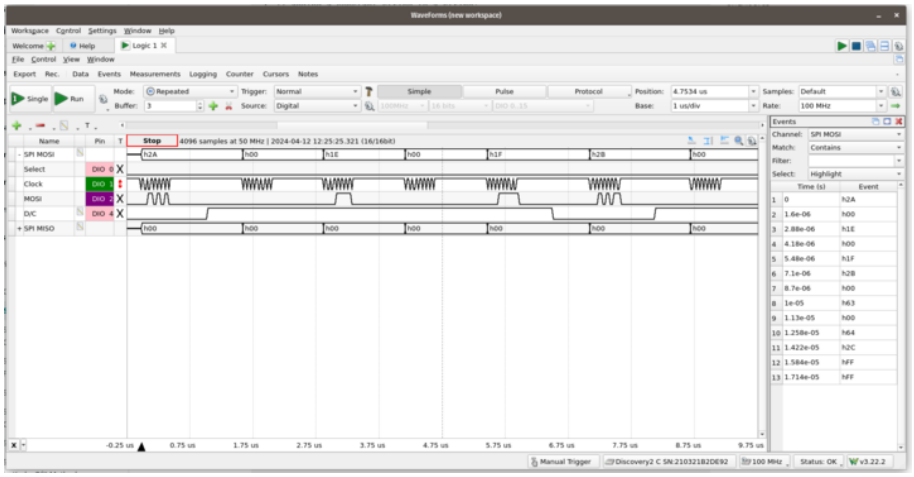


Figure 3.2: Detailed view of the correct display initialization sequence on a logic analyzer.

With the issues resolved, a new device BSP driver library (`sw/device/bsp/ST7789`) [4] has been created for the ST7789 chip. During the creation of that library, other issues arose. Copying functions from the test file to the library, suddenly made them perform differently. The initialization sequence still worked fine. This time, however, it was the image transmission that didn't work anymore. After some troubleshooting it was found out that the problem was again linked to the DC pin, which is not synched with the SPI signal. The initialization signal had no issue as there were already delays between the signals. To send an image, however, the device also starts by sending the start and end pixel of the rectangle to fill in as a command. Then it sends the color information for each pixel as data. It is unclear why this problem only arose when the functions were moved into the bsp library. The issue was solved by adding a delay to all functions that switch between command and data. Performance wist this doesn't affec performance greatly, as the bulk transfer of pixels doesn't need delays, as there is no switching between data and commands. The working bsp library will probably be added to the main X-HEEP repository.

The example program `example_adafruit_bonnet` shows the working of buttons and display by showing a 120x120 pixels image of the ESL Logo. When a button is pressed, the picture moves by two pixels in the direction of the pressed button and a message is sent on the UART terminal for each positive and negative edge of a button.

3.1.3 Pinout Configuration

On the PYNQ-Z2 FPGA, the buttons are connected to the following pins:

Description	PYNQ-Z2 PIN	Physical Location	Software PIN
Joystick UP	U8	Raspberry Pi 15	GPIO 9
Joystick DOWN	V7	Raspberry Pi 13	GPIO 10
Joystick LEFT	U7	Raspberry Pi 11	GPIO 11
Joystick RIGHT	V6	Raspberry Pi 8	GPIO 12
Button A	U13	Arduino AR2	GPIO 13
Button B	V13	Arduino AR3	GPIO 14
Display CLK	H15	Arduino SPI 3	spi_sck_o
Display MOSI	T12	Arduino SPI 4	spi_sd_io[0]
Display CS	F16	Arduino SPI 5	spi_csb_o
Display DC	V8	Raspberry Pi 19	GPIO 8

Table 3.1: Overview of the connections between the PYNQ-Z2 FPGA and the Adafruit Bonnet

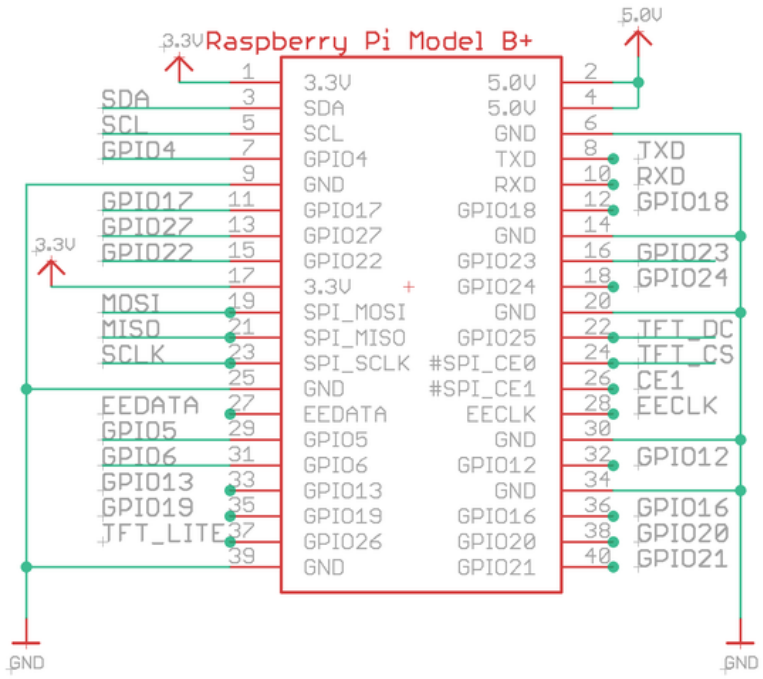


Figure 3.3: Adafruit Bonnet PINOUT diagram [1]

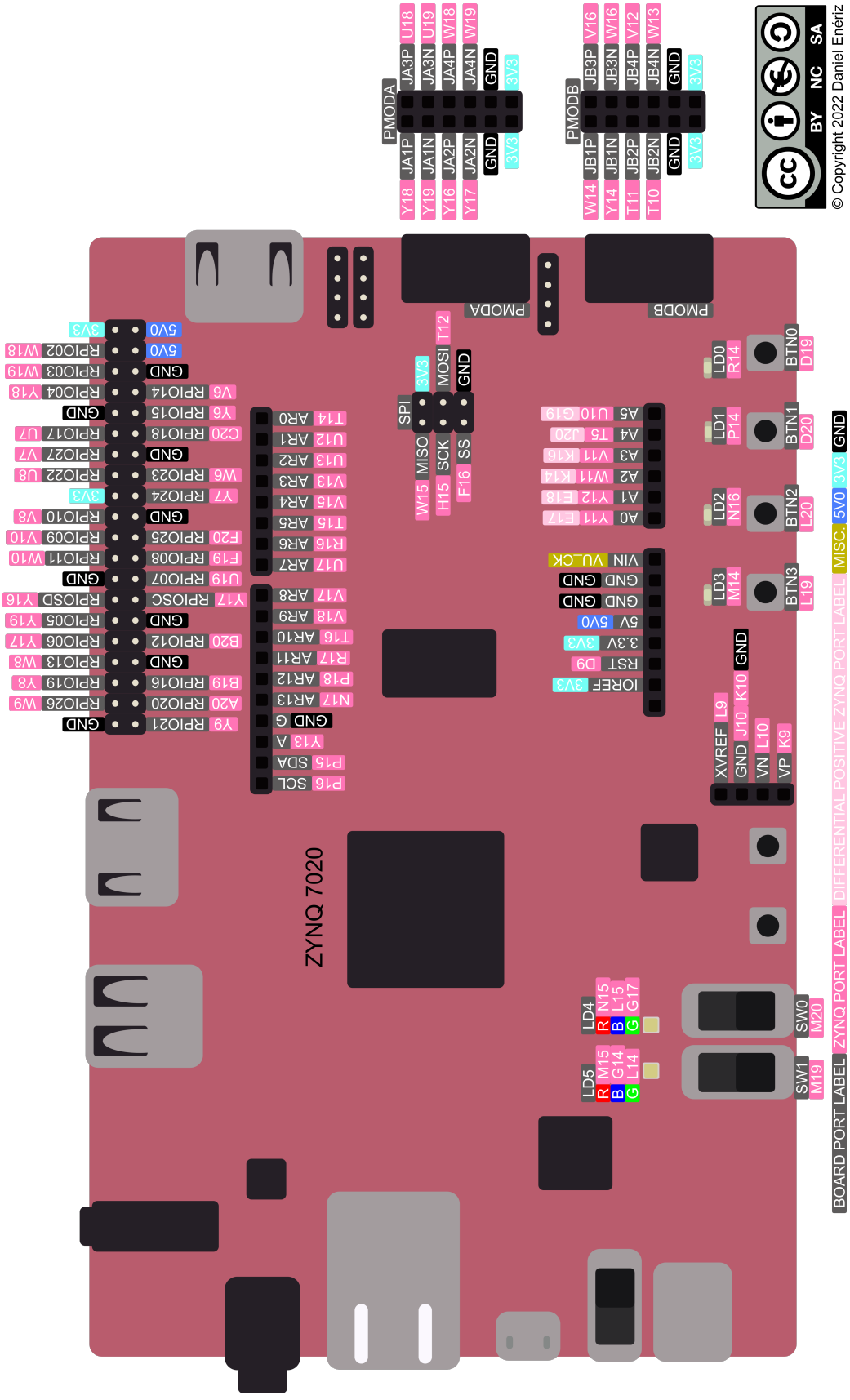


Figure 3.4: Pinout of PYNQ-Z2 FPGA [8]

Chapter 4

Porting DOOM to X-HEEP

This chapter will go over the process of porting DOOM over to the X-HEEP platform. It describes the creation of interface files for peripherals, strategies for RAM optimization, encountered obstacles on the way and the project's current status at the time of writing.

4.1 DOOM Requirements

There are two main metrics that can limit the gameplay of DOOM on X-HEEP (either for emulation on FPGA or for running on HEEPocrates): The CPU clock speed and the RAM size.

	PYNQ-Z2 FPGA	**HEEPocrates**	**Original requirements**
RAM	upto 512KB	256kB	8MB
CPU clock speed	15MHz	upto 470MHz	66MHz

Table 4.1: Comparison of CPU clock speed and RAM of different systems

While the original game was playable on slower hardware, it became fluid only at a clock speed of 66 MHz and 8MB of RAM. It was recommended to be played on an Intel 486, Pentium, or Athlon processor [10]. The hardware used on X-HEEP is significantly lacking in RAM. However, HEEPocrates has a very high clock speed compared to the original requirements. It was expected that DOOM would run very slowly on the FPGA. This is not a problem, as the FPGA is only used to test progress, and the real benchmark should be performed on HEEPocrates.

4.2 Interfacing Peripherals

This part of the project involved modifying interface files for the following peripherals:

- SPI Display
- GPIO buttons

- SPI Flash
- System Time
- Sound

All the interface files of the Nordic DOOM project were named `n_<name_of_interface>`. The Nordic project was already storing the `.wad` file on flash instead of loading it all on the RAM, which worked to this project's advantage.

Display: A driver for the ST7789 SPI display had already been created in previous steps. This made the interfacing straightforward. The display needs to be initialized once and then filled from a screen buffer with RGB565 colors in every display update. The current screen buffers hold paletted color values. Therefore, an ST7789 screen buffer was created, filled with converted RGB565 values, and then sent to the display.

Buttons: For this module, the Nordic interface (`n_buttons.c`) was used as a template. The GPIO initialization and read functions were replaced by X-HEEP functions created in previous steps.

SPI Flash: The SPI flash interface was inspired by the `example_spi_read` program in the X-HEEP repository [12]. The program is stored at address 0, whereas the `.wad` file is stored at an offset of 1MB. The offset is added to all addresses of data read calls to the flash storage. A new Makefile command was created to load the `.wad` file as a `.hex` file in the flash at an offset of 1MB: `make flash-prog-doom`

Time: The system time function needed to get the current number of clock cycles performed. This had already been implemented for performance benchmarks in other example projects and was copied to the interface file. There are no timer interrupts in DOOM.

Sound: Since no sound output was planned for this project, all the files and mentions to sound were deleted or commented out from the code.

To maintain a coherent naming scheme, all the interface files for X-HEEP are called `x_<name_of_interface>`:

Module	Interface File	Header File
Display	`x_display.c`	`x_display.h`
Buttons	`x_buttons.c`	`x_buttons.h`
SPI Flash	`x_spi.c`	`x_spi.h`
Time	`x_time.c`	`x_time.h`
Sound	–	–

Table 4.2: List of all the added interface files for X-HEEP

4.3 Problems, Solutions, and Optimization

The initial objective was to get the program to compile for the FPGA, even if it used too much RAM for HEEPocrates. From there, the code can still be optimized further.

4.3.1 RAM Usage

Even with reduced RAM usage by loading the level data to flash, the FPGA's RAM size was still too small (see Figure 4.1).

```
In file included from /home/ismael/Desktop/X_HEEP_DOOM/x-heep/sw/device/lib/runtime/syscalls.c:30:
/home/ismael/Desktop/X_HEEP_DOOM/x-heep/sw/device/target/pynq-z2/x-heep.h:8:9: note: '#pragma message: the x-heep.h for PYNQ-Z2 is used'
    8 | #pragma message ( "the x-heep.h for PYNQ-Z2 is used" )
      |         ^~~~~~~
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: main.elf section `.text' will not fit in region `ram0'
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: main.elf section `.text' will not fit in region `FLASH0'
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: main.elf section `.bss' will not fit in region `ram1'
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: main.elf section `.bss' will not fit in region `FLASH1'
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: section .data LMA [0000000040040000,000000004004042d] overlaps section
 LMA [0000000040000350,000000004004f6a7]
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: section .eh_frame VMA [000000000004f6a8,000000000004f6e3] overlaps sect
bss VMA [0000000000040530,000000000008807f]
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: region `ram0' overflowed by 63204 bytes
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: region `FLASH0' overflowed by 63204 bytes
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: region `ram1' overflowed by 36992 bytes
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: region `FLASH1' overflowed by 32896 bytes
/home/ismael/tools/riscv/lib/gcc/riscv32-unknown-elf/11.1.0/../../../../riscv32-unknown-elf/bin/ld: /tmp/cc3OtBGs.ltrans2.ltrans.o: in function `.L0 ':
/home/ismael/Desktop/X_HEEP_DOOM/x-heep/sw/applications/0_DOOM/doom/s_sound.c:468: undefined reference to `S_sfx'
```

Figure 4.1: Error message of RAM overflow before it was optimized

Screen Buffers: The first iteration of DOOM adapted for X-HEEP had three screen buffers. Two of them were the video buffer and back buffer from the original DOOM, each 320x200 pixels and paletted 8 bits per pixel. While the video buffer is written on the display, the back buffer is filled with the next frame. When both operations are finished, the two buffers switch role. The third buffer was for the ST7789 with a size of 200x200 pixels but 16 bits per pixel for an RGB565 color format. These three screen buffers alone used 203 kB of RAM. The first optimization was to delete the back buffer. In the current version of the X-HEEP implementation, the SPI transfer to the display is a blocking operation, meaning the CPU sends pixels one by one. Hence, there is no need for a back buffer, as no new image can be drawn into a buffer while another image is being drawn on the display. The second optimization was to delete the ST7789 screen buffer and convert paletted colors to RGB565 colors on the fly. As the CPU is already blocked anyway, it can also do the conversion before sending a pixel. This reduced the RAM usage of screen buffers by 141 kB, leaving 63 kB.

Augment RAM on FPGA: The next step was to unblock all available memory banks on the FPGA. There are 16 memory banks with 32 kB each, totaling 512 kB of available memory. By default only two memory banks are activated for a total of 64kB. The available memory is divided into `ram0` for code and `ram1` for data. Activating all 16 memory banks on the FPGA gave acess to the whole 512kB of memory. The correct division between the two memories was found by adjusting settings in `configs/general.hjson` on lines 17 and 21. The final value that made it compile is `0x000053000`. This divides the available space into two parts of 332 kB for `ram0`, and 180 kB for `ram1`.

Chapter 5

Current Status of the Project

5.1 Adafruit Bonnet Driver

The Adafruit 1.3" Color TFT Bonnet for Raspberry Pi [1] is fully functional on X-HEEP. A BSP (Board Support Package) library has been written to draw images on the display. Added functions include:

- **Initialization:** Initialize the display by setting up SPI and GPIO for communication, and sending an initializaiton sequence to configure the display.
- **Write Pixel:** Write a single pixel color at specified X and Y coordinates
- **Fill Screen:** Fill the screen with one specified color.
- **Fill Picture:** Fill the screen with a 120x120 pixel picture provided in a C-array
- **Fill Picture with Shift:** Fill screen with a picture provided in a C-array but with an added shift in X or/and Y coordinates.
- **Manual:** Provided functions to manually set a window and functions and fill it in. This can be used to draw pixels, vertical and horizontal lines, or rectangles.

The example program `example_adafruit_bonnet` was created to demonstrate the functionality of the display (see Figure 5.1). The library, together with the example program provide a framework and guidance for future projects requiring a display. Note that the library, the example program, and the code discussed in the next section are only available in the Github repository [4] of the project's author. Integration in the official X-HEEP repository [12] is expected.

5.2 DOOM on X-HEEP

On June 17, 2024, the DOOM for X-HEEP ran for the first time. Although the display did not show any output, console messages were observed, indicating partial success. Several issues remain unresolved as of today, June 21, 2024.

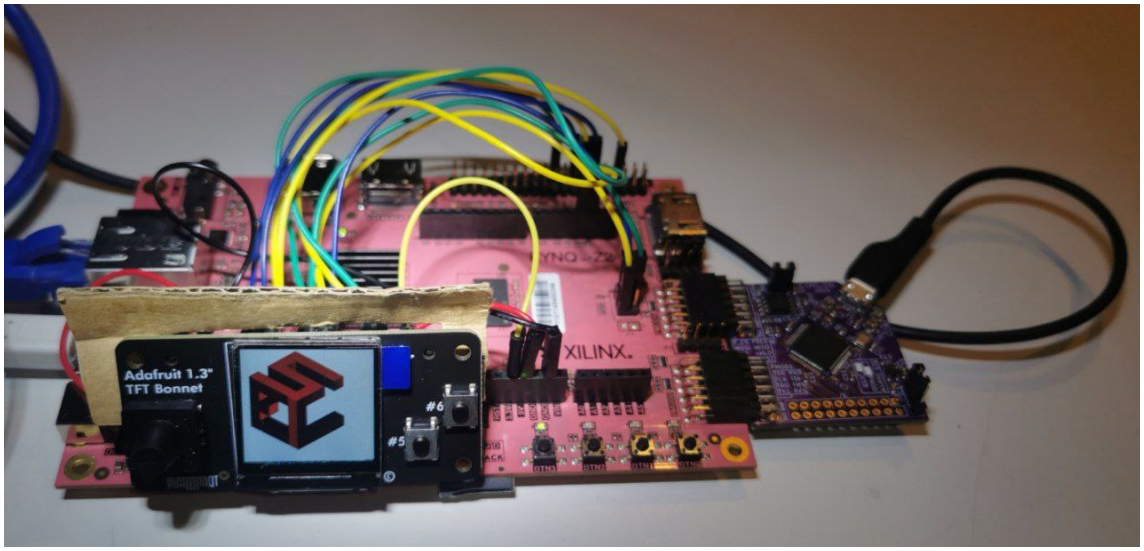


Figure 5.1: Image of the working `example_adafruit_bonnet` program executed by X-HEEP on FPGA

`flash_load` instead of `flash_exec`: The program compiles for both linker options `flash_load` and `flash_exec`. The difference between the two is that `flash_load` copies the entire program into RAM and then executes operations from RAM. This option is faster but uses more RAM. The `flash_exec` option leaves the program in flash and copies one operation at a time to the CPU for execution. This option saves RAM but has worse performance as every signle operation requires a read transaction from the flash. Only the `flash_exec` option, however, provides some output when launched. The `flash_load` option shows no sign of life. This issue requires further investigation.

SPI Flash Interface: The interface to get level data from the flash does not seem to work, see comments of Figure 5.2. An example program (`0_DOOM_TEST`)[4] has been created to investigate this issue further. One reason could be that DOOM only reacts in `flash_exec` mode, but the `example_spi_read` program explicitly states that the read function does not work in this mode. Therefore, either the previous problem needs to be fixed or a workaround has to be found to make it work.

```
---------------------------------
UART Initialized
---------------------------------
X_SPI: init flash complete
                            NRF Doom 0.1.0
Z_Init: Init zone memory allocation daemon.
zone memory: Using native C allocator.
V_Init: allocate screens.
M_LoadDefaults: Load system defaults.
NRFD-TODO: I_BindInputVariables
NRFD-TODO: I_BindJoystickVariables
NRFD-TODO: M_BindStrifeControls
NRFD-TODO: M_BindWeaponControls
NRFD-TODO: M_BindMapControls
NRFD-TODO: M_BindMenuControls
NRFD-TODO: M_BindChatControls
NRFD-TODO: NET_BindVariables
NRFD-TODO: LoadDefaultCollection
NRFD-TODO: LoadDefaultCollection
WAD File: doom.wad
W_Init: Init WADfiles.
 adding doom.wad
W_CheckCorrectIWAD
D_IdentifyVersion
InitGameVersion
NRFD-TODO: game detect
Game version:
Emulating the behavior of the 'Doom 1.9' executable.
NRDF-TODO? W_GenerateHashTable
D_SetGameDescription
NRFD-TODO: M_GetSaveGameDir
===========================================================================
                            DOOM Shareware
===========================================================================
 NRF Doom is free software, covered by the GNU General Public
 License.  There is NO warranty; not even for MERCHANTABILITY or FITNESS
 FOR A PARTICULAR PURPOSE. You are welcome to change and distribute
 copies under certain conditions. See the source for more information.
===========================================================================
I_Init: Setting up machine state.
M_Init: Init miscellaneous info.
R_Init: Init DOOM refresh daemon
R_InitTextures
W_GetNumForName: PNAMES not found!
```

Figure 5.2: Output when DOOM is run on X-HEEP prototyped into the FPGA in mode `flash_exec`

A few things to note about the output:

- NRF-TODO: NRF is a tag used for by the author of the Nordic Semiconductor implementation. These are references to possible next steps and can be ignored for this project.
- A few initialization functions seem to be executed before the program crashes.

- The error where the program crashes is at the function W_GetNumForName. PNAMES is an asset supposed to be found in the `.wad` file on the flash. As there is a problem with the reading of the flash, this asset is not available.

Chapter 6

Future Work

As DOOM is not yet fully functional on either FPGA emulation or HEEPocrates, the project remains incomplete. The following is a step-by-step plan outlining the next phases of the project:

1. **Resolve Current Errors:** Address and fix the errors mentioned at the end of the previous chapter (5).

2. **Debug New Issues on FPGA:** Identify and troubleshoot any new problems that arise during the FPGA testing phase.

3. **Test Performance on FPGA:** Once operational, evaluate the performance of DOOM on the FPGA. Due to the low clock frequency and the CPU-driven pixel-by-pixel screen update, performance is expected to be suboptimal.

4. **Optimize RAM Usage:** Further optimize RAM usage to ensure compatibility with HEEPocrates. If the problem with `flash_load` mode hasn't been solved, explore the `flash_exec` mode. If `flash_exec` mode encounters issues with data and code residing on the same flash, consider adding a separate flash memory for data storage, provided that HEEPocrates supports three SPI devices.

5. **Validate DOOM on HEEPocrates:** Once DOOM is operational on HEEPocrates, conduct thorough testing and enjoy a playthrough of the demo level.

6. **Develop a Testbench:** Create a testbench to evaluate code performance. Identify bottlenecks and explore potential optimizations. It is likely that DOOM will still not run at its maximal framerate at this stage, as the CPU is responsible for transferring each pixel individually to the display.

7. **Optimize for `flash_load` Mode:** If not previously achieved, implement `flash_load` mode instead of `flash_exec`. This transition is expected to significantly enhance performance, as HEEPocrates will no longer need to load each instruction from flash memory.

8. **Explore Paletted Mode on ST7789:** Investigate whether the ST7789 display supports a paletted mode. If available, this could improve performance

by eliminating the need for on-the-fly conversion from 8-bit paletted colors to RGB565. This would also enable the use of DMA for screen transfers, freeing up valuable CPU cycles. However, this change would necessitate reintroducing the back buffer, increasing RAM usage.

9. **Contribute to Pop Culture:** Create an internet meme showcasing DOOM running natively on an unexpected device. This contributes to the ongoing pop culture trend of running DOOM on unconventional hardware and positions ESL and EPFL as leaders, inspiring other researchers and enthusiasts to accept similar challenges.

Chapter 7

Conclusion

In conclusion, the project to port DOOM to the X-HEEP platform has made significant strides, though it remains incomplete. Key achievements include:

- The creation of a software driver for the Adafruit Bonnet for input and display
- Its integration into the DOOM source code alongside other peripherals (SPI, system time)
- Removing unnecessary parts of the DOOM source code (sound, network options)
- Substantial optimization of RAM usage

However, challenges such as resolving flash memory read issues and further optimizing the code remain. The project has provided valuable insights into the capabilities and limitations of the X-HEEP platform and outlines a roadmap for future work. By addressing these remaining challenges, it is anticipated that DOOM can eventually run on X-HEEP, contributing to the broader effort of demonstrating the flexibility and performance of open-source RISC-V microcontrollers.

Bibliography

[1] *Adafruit TFT Display Bonnet*. June 20th, 2024. URL: https://learn.adafruit.com/adafruit-1-3-color-tft-bonnet-for-raspberry-pi.

[2] *Arduino ST7789 Fast Library*. June 20th, 2024. URL: https://github.com/cbm80amiga/Arduino_ST7789_Fast/tree/b2782a381d61511b87df6cd6b20b71276d072a6d.

[3] *embeddedDOOM Github Repository*. June 20th, 2024. URL: https://github.com/cnlohr/embeddedDOOM.

[4] *Github Repository of the project*. June 20th, 2024. URL: https://github.com/isidebisi/x-heep.

[5] *HEEPocrates EPFL research*. June 20th, 2024. URL: https://www.epfl.ch/labs/esl/research/systems-on-chip/heepocrates/.

[6] *Homepage of Can It Run DOOM ?* June 20th, 2024. URL: https://canitrundoom.org/.

[7] *nrf-doom Github Repository*. June 20th, 2024. URL: https://github.com/NordicPlayground/nrf-doom.

[8] *PYNQ-Z2 Pinout Diagram*. June 20th, 2024. URL: https://discuss.pynq.io/t/pynq-z2-pinout/4256.

[9] *Reddit R/itrunsdoom Community*. June 20th, 2024. URL: https://www.reddit.com/r/itrunsdoom/.

[10] *System Requirements for the original DOOM*. June 20th, 2024. URL: https://gamesystemrequirements.com/game/doom.

[11] *X-HEEP Homepage*. June 20th, 2024. URL: https://x-heep.epfl.ch/.

[12] *X-Heep Repository on Github*. June 20th, 2024. URL: https://github.com/esl-epfl/x-heep.