Integrating Linux with FPGA: A MicroBlaze Implementation on Nexys A7

Doluwamu Taiwo Kuye[1] Ashraf Siddiqi Mohammed[2]

Contents

1 Project Overview **3**

1.1 Integrating Linux with FPGA: A MicroBlaze Implementation on Nexys A7 3

1.2 The Nexys A7 Board and MicroBlaze Processor 3

1.3 Hardware Design and IP Core Utilization 3

1.4 Objective 3

1.5 Significance 4

1.6 Key Components and Tools 4

2 Background **6**

3 Motivation and Goals **6**

3.1 Motivation 6

3.2 Goals 6

4 Project Flow **7**

4.1 Hardware Design in Vivado 7

4.2 PetaLinux Development 10

4.2.1 Fundamentals and Tools of Petalinux Environment 10

[1] Doluwamu-taiwo.kuye@stud.hshl.de
[2] mohammad-ashrafuzzaman.siddiqi@stud.hshl.de

5 Implementation **12**

6 Evaluation **16**

6.1 Testing the Build with QEMU 16

6.2 FINAL TEST ON FPGA 17

7 Challenges Encountered and Overcome **20**

7.1 Compatibility Issues with Operating Systems and Petalinux Versions . 20

7.2 Limitations in Hardware Design: Ethernet PHY Integration 20

7.3 Device Tree Complications 20

7.4 Understanding the Embedded Linux Environment 20

7.5 Challenges with the Ubuntu Linux Work Environment 20

7.6 Concluding Thoughts 21

8 Future Directions **21**

9 Conclusion **22**

Abstract:

This document presents a comprehensive overview of our project, which entailed the development and implementation of a Petalinux-based system on a Microblaze processor. The project involved navigating through various challenges, including compatibility issues with different versions of Ubuntu and Petalinux, complexities in hardware design, particularly with Ethernet PHY integration, and intricacies within the embedded Linux environment. Our journey through these challenges has not only led to a successful implementation but has also provided us with invaluable insights into the nuances of embedded system design and development.

1 Project Overview

1.1 Integrating Linux with FPGA: A MicroBlaze Implementation on Nexys A7

This project focuses on the innovative application of running a Linux-based operating system on a Nexys A7 FPGA board utilizing the MicroBlaze soft processor core. It bridges the gap between high-level operating systems and low-level hardware control, showcasing the flexibility and capabilities of FPGA technology.

1.2 The Nexys A7 Board and MicroBlaze Processor

The Nexys A7 board, renowned for its robust features and FPGA capabilities, is the chosen platform for this endeavor. It comes equipped with a range of peripherals and interfaces, making it an ideal candidate for sophisticated hardware-software integration projects.

MicroBlaze, a soft processor core designed by Xilinx for FPGAs, serves as the project's heart. This versatile and configurable core allows the development of a processor tailored to specific needs, perfect for customized embedded systems.

1.3 Hardware Design and IP Core Utilization

A critical aspect of this project involves the hardware design in Vivado, leveraging IP cores and block design methodologies. This approach significantly enhances system efficiency and speeds up the development process. By utilizing pre-built IP cores, the project benefits from reliable, tested components, which are essential for integrating complex systems like Linux on an FPGA. The block design in Vivado allows for a visual and modular approach to constructing the hardware architecture, further simplifying the process of system integration.

1.4 Objective

The primary objective is to run a Linux operating system successfully on the MicroBlaze processor, interfacing with the Nexys A7 board's peripherals. This includes managing a range of GPIOs such as LEDs, switches, buttons, seven-segment anodes, USB UART, and a timer. The project also integrates a Quad SPI for Linux and configures the necessary interrupts.

1.5 Significance

This project demonstrates the practical application of merging FPGA hardware flexibility with the versatility of a Linux operating system. It caters to educational purposes, embedded system development, and experimentation in hardware-software co-design.

Running Linux on an FPGA using MicroBlaze is more than a technical achievement; it's an educational journey into computer engineering, encompassing hardware design, software engineering, and system integration.

1.6 Key Components and Tools

- **Nexys A7 FPGA Board:** The Nexys A7 is an FPGA board designed for educational purposes and prototyping. It offers a rich set of features including onboard I/O devices and extensive connectivity options. Known for its ease of use and flexibility, it is a popular choice for both learning and development in digital electronics and FPGA design.

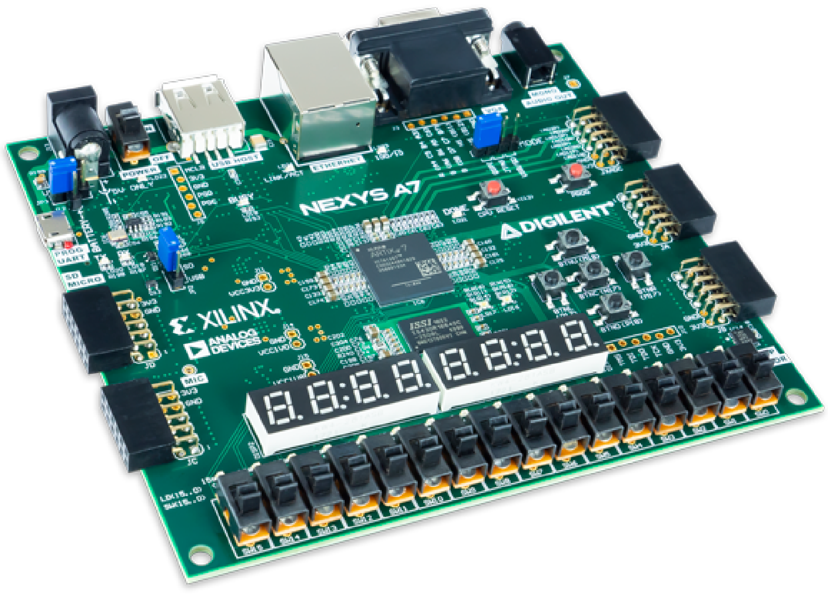


Fig. 1: Nexys A7 FPGA Board

- **MicroBlaze Processor:** MicroBlaze is a soft processor core designed for Xilinx FPGAs. It's configurable and adaptable, suitable for a wide range of applications. MicroBlaze can be tailored to specific needs, offering versatility for customized embedded systems. It supports various peripherals and memory interfaces, ideal for complex designs that require a custom processor solution.

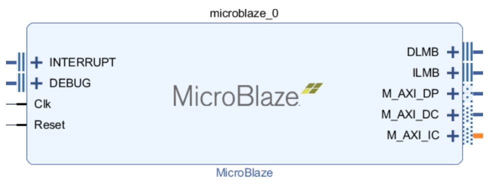


Fig. 2: MicroBlaze Architecture

- **Vivado 2022.2:** Vivado, from Xilinx, is a software development environment for FPGA, SoC, and MPSoC. Used for higher-level synthesis, design analysis, and implementation of FPGA designs, Vivado supports the latest high-capacity FPGAs and provides a comprehensive design environment with advanced synthesis and analysis tools. It is known for its IP integrator, which enables designers to create complex systems using IP cores.
- **PetaLinux 2019.2:** PetaLinux is a set of software tools developed by Xilinx for streamlined embedded Linux development. Not a standalone solution, but a package that simplifies customizing, building, and deploying embedded Linux solutions on Xilinx processing systems. Based on the Yocto Linux distribution, it works closely with other Xilinx software like Vivado and Xilinx SDK. Its features include command-line interfaces, application and device driver generators, system image builder, debug agents, GCC tools, an integrated QEMU full system simulator, and automated tools. PetaLinux tools also enable developers to synchronize the software platform with the hardware design, automatically generating a custom Linux Board Support Package, including device drivers for embedded processing IP cores, kernel, and bootloader configurations.

This section introduces the project, providing a clear understanding of its context, the involved hardware and software, and the overarching goals and significance. Each following section will delve into specific details of the project, from hardware configuration to software implementation and testing.

2 Background

In the realm of embedded systems, the fusion of adaptable hardware and sophisticated software heralds a new era of innovation. Our project embarks on this journey, harnessing the synergy of FPGA technology with the versatility of Linux, embodied in the MicroBlaze soft processor and the Nexys A7 FPGA board. This convergence is not merely a technical experiment but a bold exploration into creating more efficient, intelligent, and versatile systems.

Imagine a world where the boundaries of hardware limitations are redefined, where an FPGA board does not just perform pre-defined tasks but evolves with the demands of its environment. This is the vision driving our project - to transform the Nexys A7 FPGA board from a static entity into a dynamic and intelligent system, capable of running Linux, a bastion of modern computing.

3 Motivation and Goals

3.1 Motivation

Our decision to embark on this project was fueled by a deep-rooted passion for challenges and the intrigue of Linux's complexity. The world of embedded Linux is a labyrinth of endless possibilities, and navigating this maze requires both skill and tenacity. As students of an advanced embedded class, we yearn to push the boundaries of what's possible, merging disparate systems to forge something truly groundbreaking. This project is more than an academic endeavor; it is a testament to our love for innovation, our resilience in the face of technical complexity, and our commitment to pioneering new frontiers in embedded systems.

3.2 Goals

- **Creation of a Miniature Computer:** Our aim is to morph the Nexys A7 FPGA board into a mini-computer, one that not only runs Linux but is also capable of executing complex tasks like Python scripting and other computational functions.
- **Low-Power, High-Performance Computing:** We aspire to demonstrate that power efficiency and high performance are not mutually exclusive. This project is our foray into creating a system that is both energy-efficient and robust in capabilities.
- **Innovation in Embedded Systems:** At the heart of our project lies the goal to innovate. We want to leave a mark in the field of embedded systems, showing that through creativity and technical prowess, we can redefine the norms of what small-scale systems can achieve.

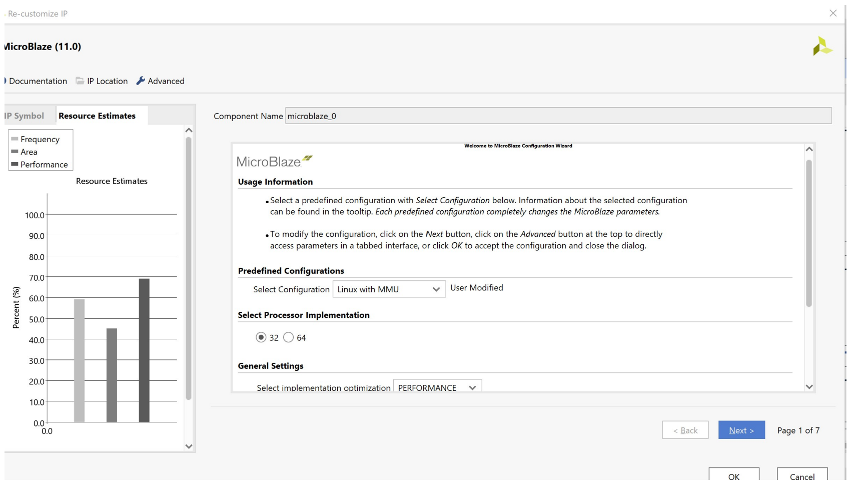
In essence, our project is a blend of ambition, technical skill, and a daring spirit to venture into uncharted territories of embedded systems. We stand at the cusp of creating something not just functional but transformative.

4 Project Flow

4.1 Hardware Design in Vivado

The hardware design phase in Vivado is the foundation upon which our Linux-capable system is built. This process is meticulously engineered to ensure optimal performance and compatibility with the Linux operating system on the MicroBlaze processor.

- **Clock Configuration:** The MicroBlaze and all peripherals are clocked using a UI clock at 81MHz, derived from the MIG (Memory Interface Generator), which takes the system clock at 100MHz from the Nexys A7 board. This specific clock configuration is chosen for its balance of performance and power efficiency, crucial for Linux-based systems.
- **MicroBlaze Configuration:** Configured specifically for Linux with the MMU (Memory Management Unit) option enabled and 64KB of instruction and data cache. This setup provides the necessary resources for Linux, allowing efficient handling of memory-intensive processes. This can be seen in 3



Component Name microblaze_0

Welcome to MicroBlaze Configuration Wizard

MicroBlaze

Usage Information

- Select a predefined configuration with *Select Configuration* below. Information about the selected configuration can be found in the tooltip. *Each predefined configuration completely changes the MicroBlaze parameters.*
- To modify the configuration, click on the *Next* button, click on the *Advanced* button at the top to directly access parameters in a tabbed interface, or click *OK* to accept the configuration and close the dialog.

Predefined Configurations

Select Configuration Linux with MMU User Modified

Select Processor Implementation

[✓ 32] 64

General Settings

Select implementation optimization PERFORMANCE

Fig. 3: Microblaze Linux Settings

- **UART Configuration:** The UART is crucial for serial communication, particularly for debugging and data transfer purposes. It is configured with a baud rate of 115200, ensuring reliable and swift data transmission.
- **Timer and Interrupts:** The inclusion of a timer and the configuration of interrupts are essential for task management and scheduling in the Linux environment. These components help in maintaining system responsiveness and efficiency.
- **Quad SPI Flash:** The Quad SPI flash memory is used for storage purposes, crucial for booting Linux and storing system configurations and data.
- **DDR2 Memory and GPIOs:** DDR2 memory provides the necessary storage for running Linux and its applications, while GPIOs like Leds, RGB, seven-segment, switches, and buttons are used for interacting with the external environment and user inputs.

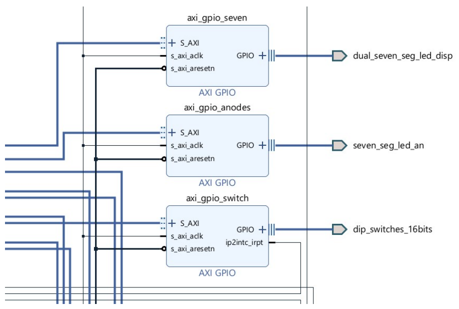


Fig. 4: GPIO in Block diagram

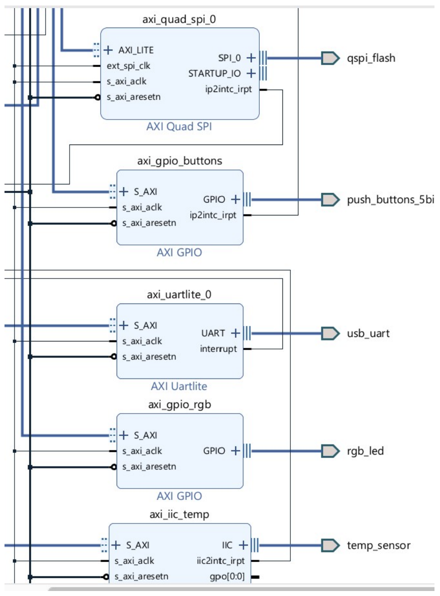
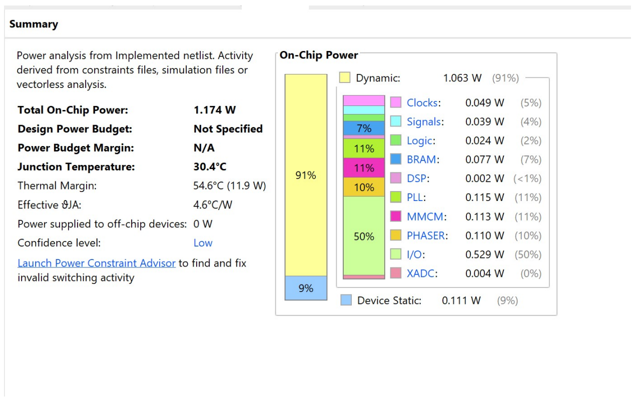


Fig. 5: GPIO in Block diagram

The final steps involve creating the wrapper, generating the bitstream, and exporting the hardware XSA file, which is then used in the PetaLinux project. This process marks the transition from hardware configuration to software implementation, setting the stage for running Linux on the MicroBlaze processor.

This carefully orchestrated design not only ensures compatibility with Linux but also opens the door to a realm of applications, demonstrating the power of combining FPGA flexibility with GPOS.



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.174 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**30.4°C**
Thermal Margin:	54.6°C (11.9 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:		1.063 W	(91%)
	Clocks:	0.049 W	(5%)
	Signals:	0.039 W	(4%)
	Logic:	0.024 W	(2%)
	BRAM:	0.077 W	(7%)
	DSP:	0.002 W	(<1%)
	PLL:	0.115 W	(11%)
	MMCM:	0.113 W	(11%)
	PHASER:	0.110 W	(10%)
	I/O:	0.529 W	(50%)
	XADC:	0.004 W	(0%)
Device Static:		0.111 W	(9%)

Fig. 6: Power Summary

Resource Utilization The resource utilization metrics for the FPGA indicate a healthy margin for scalability within the design. Utilization of LUTs and Registers at 20.05% and 10.91%, respectively, suggests that the design is neither resource-starved nor nearing capacity, allowing for future expansion. Memory utilization through LUTs at 9.12% affirms efficient use with ample remaining headroom. Minimal use of F7 and F8 Muxes reflects a straightforward routing structure, thus maintaining optimal performance. Overall, the FPGA has sufficient resources for the current Linux-based application with room for additional complexity if required.

```
+----------------------------+-------+-------+------------+-----------+-------+
|          Site Type         |  Used | Fixed | Prohibited | Available | Util% |
+----------------------------+-------+-------+------------+-----------+-------+
| Slice LUTs                 | 12709 |     0 |          0 |     63400 | 20.05 |
|   LUT as Logic             | 10977 |     0 |          0 |     63400 | 17.31 |
|   LUT as Memory            |  1732 |     0 |          0 |     19000 |  9.12 |
|     LUT as Distributed RAM |  1258 |     0 |            |           |       |
|     LUT as Shift Register  |   474 |     0 |            |           |       |
| Slice Registers            | 13832 |     0 |          0 |    126800 | 10.91 |
|   Register as Flip Flop    | 13828 |     0 |          0 |    126800 | 10.91 |
|   Register as Latch        |     0 |     0 |          0 |    126800 |  0.00 |
|   Register as AND/OR       |     4 |     0 |          0 |    126800 | <0.01 |
| F7 Muxes                   |   194 |     0 |          0 |     31700 |  0.61 |
| F8 Muxes                   |     8 |     0 |          0 |     15850 |  0.05 |
+----------------------------+-------+-------+------------+-----------+-------+
* Warning! LUT value is adjusted to account for LUT combining.
```

Fig. 7: Resource Utilization

4.2 PetaLinux Development

The PetaLinux development phase commenced on a strategically chosen Ubuntu 16.04 LTS workstation. This environment was selected for its compatibility and stability with the PetaLinux 2019.2 development tools. Later Versions like Ubuntu 20 LTS and Ubuntu 22 were also tested but the build process failed due to dependency issues.

4.2.1 Fundamentals and Tools of Petalinux Environment

Integration of U-Boot, Yocto, and Device Trees in Petalinux Development

Introduction to Petalinux Development Petalinux, developed by Xilinx, is a comprehensive toolkit designed to streamline the development of embedded Linux systems. Based on the Yocto Linux distribution, it provides a suite of tools for customizing, building, and deploying embedded Linux solutions on Xilinx processing systems.

U-Boot in Petalinux U-Boot (Universal Boot Loader) is an open-source, primary bootloader used in embedded systems. In the Petalinux environment, U-Boot is responsible for the initial hardware setup, including initializing the processor, memory, and peripherals. It also prepares the system to load and execute the Linux kernel.

Role in Petalinux: Within Petalinux, U-Boot is customized to suit specific hardware configurations. This customization includes setting up boot arguments, and memory layouts, and defining which devices to initialize. U-Boot thus acts as a bridge between the hardware initialization and the launching of the Linux kernel, ensuring a seamless boot process.

Yocto Project and Petalinux The Yocto Project is an open-source collaboration project that provides templates, tools, and methods to create custom Linux-based systems for embedded products.

Integration with Petalinux: Petalinux leverages the Yocto Project's tools for building custom Linux distributions. This integration allows developers to use Yocto recipes, layers, and configurations to create a tailored Linux image that meets the specific needs of their embedded system. The Yocto framework within Petalinux ensures a flexible and reproducible build process, essential for embedded system development.

Device Trees in Petalinux *Understanding Device Trees*: A device tree is a data structure used to describe the hardware components of a system in a format that the Linux kernel can understand. It provides information about the hardware, such as the memory layout, the available peripherals, and their configuration.

Importance in Petalinux: In Petalinux, the device tree plays a critical role in informing the Linux kernel about the hardware specifics of the embedded system. This allows the kernel to correctly initialize and manage the hardware. Modifying the device tree in Petalinux enables developers to ensure that the Linux kernel is accurately configured for their specific hardware setup, enhancing the system's stability and performance.

Practical Implications *Customization and Configuration*: Developers can customize U-Boot settings and device tree files in Petalinux to align the boot process and kernel initialization with their hardware design.

Building and Running the Project: The combination of U-Boot, Yocto-based tools, and device tree customizations results in an extensive build and execution environment in Petalinux. This integration ensures that the final embedded system is optimized for performance, and reliability, and aligned with the specific requirements of the project.

Conclusion The integration of U-Boot, Yocto, and device trees in Petalinux is fundamental to building and running successful embedded Linux projects. These components work in harmony to provide a customizable, efficient, and reliable development environment, crucial for the unique demands of embedded systems.

5 Implementation

- **Installing PetaLinux:** Initial steps involved setting up the development environment by installing necessary dependencies and the PetaLinux toolkit, ensuring all tools and dependency were available for streamlined development.
  - Firstly we run the downloaded petalinux installer

    ```
    ./petalinux-v2019.2-final-installer.run /tools/Xilinx/PetaLinux
    ```

  - Then we install dependencies for our ubuntu workstation:

    ```
    sudo apt-get install -y gcc git make net-tools libncursesw5-dev tftpd
    sudo apt-get install -y zlib1g-dev libssl-dev flex bison libselinux1 gnupg
    sudo apt-get install -y wget diffstat chrpath socat xterm autoconf libtool
    sudo apt-get install -y tar unzip texinfo gcc-multilib build-essential libsdl1.2-dev
    sudo apt-get install -y libglib2.0-dev zlib1g:i386 screen pax gzip gawk
    sudo apt-get install -y glibc-doc:i386 locales:i386 ncurses-dev qemu-system-arm:i386
    sudo apt-get install -y ncurses-dev:i386 libstdc++6:i386 libselinux1:i386 lib32ncurses5-d
    ```

```
sudo apt-get install -y package-name
```

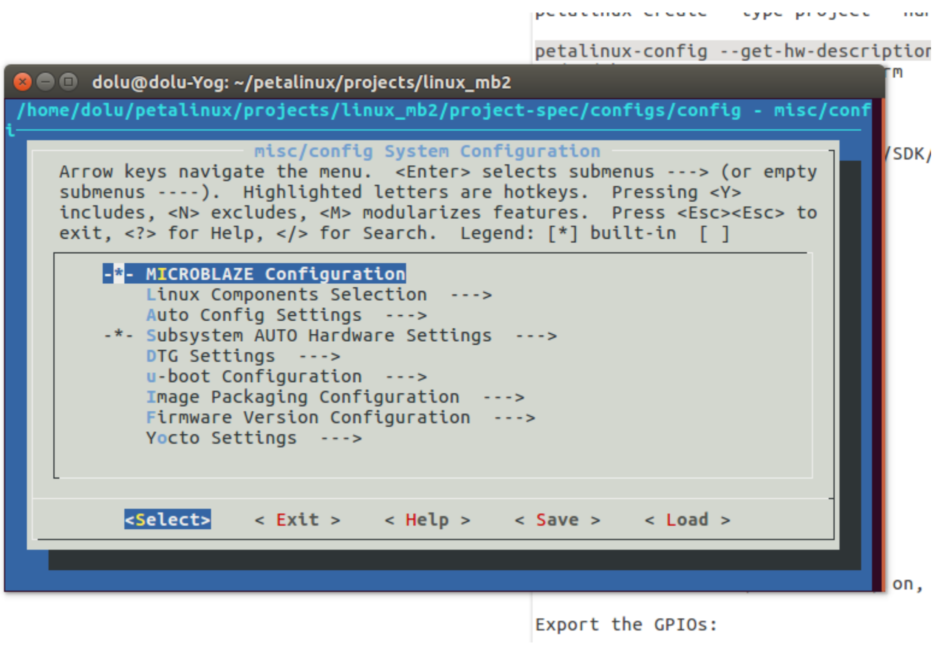
- **Creating a New PetaLinux Project:** A project named ‘linux_mb‘ was initiated, incorporating the hardware description exported from Vivado. This crucial step ensures that the software aligns perfectly with our FPGA’s hardware configuration.
  - Firstly we source the petalinux settings with:

    ```
    source ~/petalinux/settings.sh
    ```

    This step must be repeated at every new terminal session. A good practice is keeping the code in bash so we can call bash at the beginning of every session and get the environment ready.
  - Here we create a petalinux generated folder for our project called linux_mb :

    ```
    petalinux-create --type project --template microblaze --name linux_mb
    ```
  - Then we create the project using our hardware XSA file generated from Vivado with this command :

    ```
    petalinux-config --get-hw-description=/home/dolu/Documents/GitHub/Linux_Microblaze_V1.0/L
    ```



```
petalinux-config --get-hw-descriptior
dolu@dolu-Yog: ~/petalinux/projects/linux_mb2
/home/dolu/petalinux/projects/linux_mb2/project-spec/configs/config - misc/conf
                 misc/config System Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

      -*- MICROBLAZE Configuration
          Linux Components Selection  --->
          Auto Config Settings  --->
      -*- Subsystem AUTO Hardware Settings  --->
          DTG Settings  --->
          u-boot Configuration  --->
          Image Packaging Configuration  --->
          Firmware Version Configuration  --->
          Yocto Settings  --->

       <Select>    < Exit >    < Help >    < Save >    < Load >
/SDK/
on,
Export the GPIOs:
```

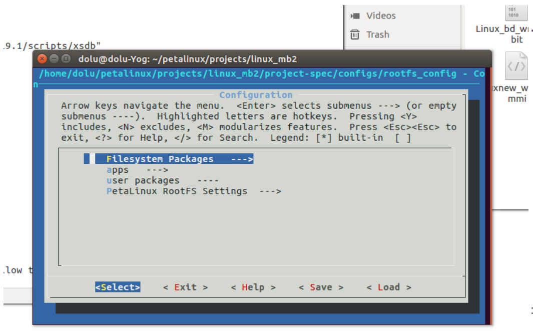
Fig. 8: GUI for petalinux config

Here we can make configuration settings according to our needs.

- **Configuration of PetaLinux Project:** Here we configure the petalinux, the rootfs and the kernel. The root filesystem (rootfs) configuration step is where we add additional functionalities to the already existing petalinux configuration. Rootfs configuration involves customizing this filesystem to meet specific requirements. This can include adding or removing software packages, configuring user accounts, setting up permissions, and tailoring the system to suit the intended application or environment. The configuration process ensures that the Linux system functions correctly and securely when it boots up. It's an integral part of building a customized Linux distribution or embedded system.
  - We Configure the petalinux using the following command:

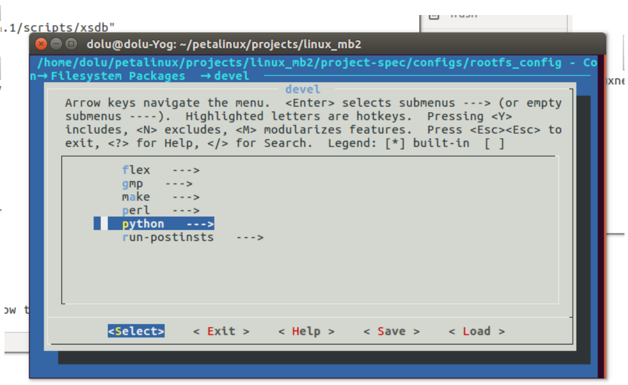
    ```
    petalinux-config -c rootfs
    ```

  A GUI where we can make selections based on our requirements is presented after exiting the configuration is saved.



```
dolu@dolu-Yog: ~/petalinux/projects/linux_mb2
/home/dolu/petalinux/projects/linux_mb2/project-spec/configs/rootfs_config - Co
n
                              Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

        Filesystem Packages   --->
        apps   --->
        user packages   ----
        PetaLinux RootFS Settings  --->

     <Select>    < Exit >    < Help >    < Save >    < Load >
```

Fig. 9: Filesystem GUI selection menu



```
dolu@dolu-Yog: ~/petalinux/projects/linux_mb2
/home/dolu/petalinux/projects/linux_mb2/project-spec/configs/rootfs_config - Co
n→ Filesystem Packages → devel
                              devel
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

        flex   --->
        gmp   --->
        make   --->
        perl   --->
        python   --->
        run-postinsts   --->

     <Select>    < Exit >    < Help >    < Save >    < Load >
```

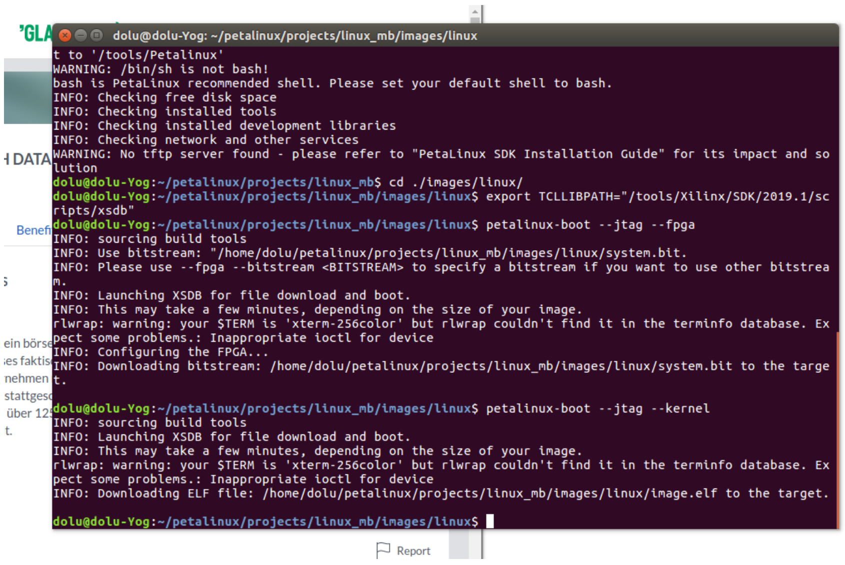
Fig. 10: Added Python

- **Building the PetaLinux Project:** Despite its time-intensive nature, the build process was successful, compiling the essential components to bring our Linux system to life on the FPGA.

  ```
  petalinux-build
  ```

- **Programming the FPGA:** Lastly, the use of XSDB commands to program the Nexys A7 board and load the kernel is explored, a testament to the versatility and capability of command-line based FPGA programming. The successful execution of the following commands marks the culmination of the PetaLinux development phase, bringing the Linux-powered FPGA system to life

  ```
  petalinux-boot --jtag --fpga
  petalinux-boot --jtag --kernel
  ```



```
dolu@dolu-Yog: ~/petalinux/projects/linux_mb/images/linux
t to '/tools/Petalinux'
WARNING: /bin/sh is not bash!
bash is PetaLinux recommended shell. Please set your default shell to bash.
INFO: Checking free disk space
INFO: Checking installed tools
INFO: Checking installed development libraries
INFO: Checking network and other services
WARNING: No tftp server found - please refer to "PetaLinux SDK Installation Guide" for its impact and so
lution
dolu@dolu-Yog:~/petalinux/projects/linux_mb$ cd ./images/linux/
dolu@dolu-Yog:~/petalinux/projects/linux_mb/images/linux$ export TCLLIBPATH="/tools/Xilinx/SDK/2019.1/sc
ripts/xsdb"
dolu@dolu-Yog:~/petalinux/projects/linux_mb/images/linux$ petalinux-boot --jtag --fpga
INFO: sourcing build tools
INFO: Use bitstream: "/home/dolu/petalinux/projects/linux_mb/images/linux/system.bit.
INFO: Please use --fpga --bitstream <BITSTREAM> to specify a bitstream if you want to use other bitstrea
m.
INFO: Launching XSDB for file download and boot.
INFO: This may take a few minutes, depending on the size of your image.
rlwrap: warning: your $TERM is 'xterm-256color' but rlwrap couldn't find it in the terminfo database. Ex
pect some problems.: Inappropriate ioctl for device
INFO: Configuring the FPGA...
INFO: Downloading bitstream: /home/dolu/petalinux/projects/linux_mb/images/linux/system.bit to the targe
t.

dolu@dolu-Yog:~/petalinux/projects/linux_mb/images/linux$ petalinux-boot --jtag --kernel
INFO: sourcing build tools
INFO: Launching XSDB for file download and boot.
INFO: This may take a few minutes, depending on the size of your image.
rlwrap: warning: your $TERM is 'xterm-256color' but rlwrap couldn't find it in the terminfo database. Ex
pect some problems.: Inappropriate ioctl for device
INFO: Downloading ELF file: /home/dolu/petalinux/projects/linux_mb/images/linux/image.elf to the target.

dolu@dolu-Yog:~/petalinux/projects/linux_mb/images/linux$
```

Fig. 11: Programming the board

6 Evaluation

6.1 Testing the Build with QEMU

QEMU, which stands for "Quick Emulator,ïs essential to our development process. It allows us to test our PetaLinux build without requiring physical hardware, and it provides a vital environment for validation and debugging.

What exactly is QEMU?

QEMU is an open-source emulator that allows us to run software written for one architecture (for example, ARM) on another (for example, x86). On our Ubuntu workstation, we utilise QEMU to emulate the MicroBlaze processor and the Linux environment.

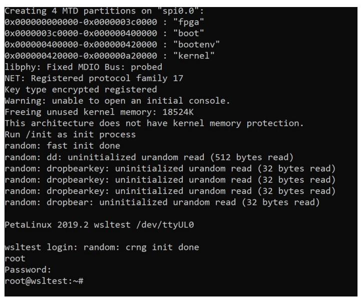
How QEMU Benefits Our Project

QEMU provides several advantages:

- Rapid Development: It accelerates the development cycle by allowing us to test software on the host machine before deploying it on the target FPGA.
- Debugging: QEMU provides debugging capabilities, helping us identify and resolve issues more efficiently.
- Portability: It enhances project portability, as software can be tested independently of the target hardware.

We initiate the testing process by running the following command:

```
petalinux-boot --qemu --kernel
```



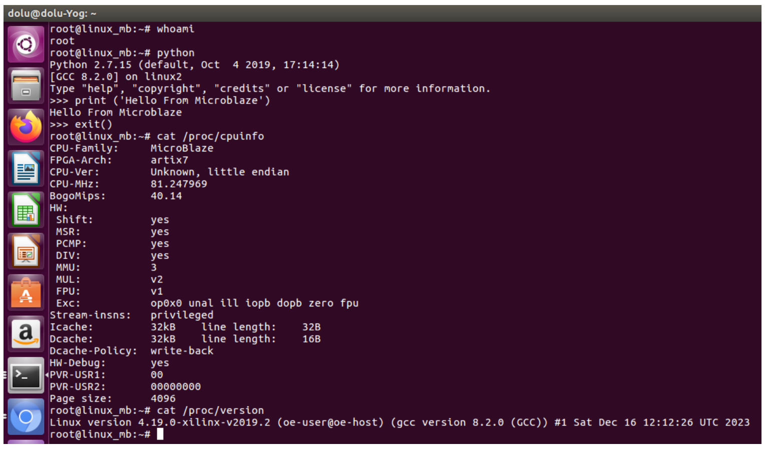
```
Creating 4 MTD partitions on "spi0.0":
0x000000000000-0x0000003c0000 : "fpga"
0x0000003c0000-0x000000400000 : "boot"
0x000000400000-0x000000420000 : "bootenv"
0x000000420000-0x000000a20000 : "kernel"
libphy: Fixed MDIO Bus: probed
NET: Registered protocol family 17
Key type encrypted registered
Warning: unable to open an initial console.
Freeing unused kernel memory: 18524K
This architecture does not have kernel memory protection.
Run /init as init process
random: fast init done
random: dd: uninitialized urandom read (512 bytes read)
random: dropbearkey: uninitialized urandom read (32 bytes read)
random: dropbearkey: uninitialized urandom read (32 bytes read)
random: dropbearkey: uninitialized urandom read (32 bytes read)
random: dropbear: uninitialized urandom read (32 bytes read)

PetaLinux 2019.2 wsltest /dev/ttyUL0

wsltest login: random: crng init done
root
Password:
root@wsltest:~#
```

Fig. 12: Login successful on QEMU

6.2 FINAL TEST ON FPGA

After simulation on QEMU, we proceeded to program the FPGA board directly using the commands stated in the Petalinux implementation section. We then set up a miniterm terminal and configured it for serial communication with the FPGA at a baud rate of 9600. This setup provided the first indication of a functioning system, as evidenced by the UART IP module configured during the Hardware design stage.

Figure 13 shows the Petalinux kernel booting successfully and a successful user login.



```
dolu@dolu-Yog: ~
root@linux_mb:~# whoami
root
root@linux_mb:~# python
Python 2.7.15 (default, Oct  4 2019, 17:14:14)
[GCC 8.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print ('Hello From Microblaze')
Hello From Microblaze
>>> exit()
root@linux_mb:~# cat /proc/cpuinfo
CPU-Family:     MicroBlaze
FPGA-Arch:      artix7
CPU-Ver:        Unknown, little endian
CPU-MHz:        81.247969
BogoMips:       40.14
HW:
 Shift:         yes
 MSR:           yes
 PCMP:          yes
 DIV:           yes
 MMU:           3
 MUL:           v2
 FPU:           v1
 Exc:           op0x0 unal ill iopb dopb zero fpu
Stream-insns:   privileged
Icache:         32kB    line length:    32B
Dcache:         32kB    line length:    16B
Dcache-Policy:  write-back
HW-Debug:       yes
PVR-USR1:       00
PVR-USR2:       00000000
Page size:      4096
root@linux_mb:~# cat /proc/version
Linux version 4.19.0-xilinx-v2019.2 (oe-user@oe-host) (gcc version 8.2.0 (GCC)) #1 Sat Dec 16 12:12:26 UTC 2023
root@linux_mb:~#
```

Fig. 13: Petalinux kernel booting and user login on NexysA7 FPGA.

Testing Different Functionalities

- **Python:** Testing began with running Python to execute a simple print statement, verifying the software's operational status:

  ```
  print('Hello from Microblaze')
  ```

  The output is shown in Figure 13.

- **CPU and Linux Information:** To corroborate the integration results, we ran commands that display information about the processor and the Linux Kernel version:

  ```
  cat /proc/cpuinfo
  cat /proc/version
  ```

  These results are documented in Figure 13.

Further Test- GPIO To control peripherals like LEDs and the RGB LED on our FPGA, we needed to identify the available GPIOs on our system. This was done with the following command, which lists all GPIOs, allowing us to locate and determine the specific ones we intended to control:

```
ls /sys/class/gpio/
```

Upon identifying the relevant GPIOs, we proceeded to export them to user space with these commands:

```
echo <GPIO_NUMBER> > /sys/class/gpio/export
```

For instance, to export GPIO number 483 and 453, we used:

```
echo 483 > /sys/class/gpio/export
echo 453 > /sys/class/gpio/export
```

After exporting the GPIOs, we set their direction to 'out', which enables us to send signals to these pins:

```
echo out > /sys/class/gpio/gpio<GPIO_NUMBER>/direction
```

For our specific GPIOs, the commands looked like this:

```
echo out > /sys/class/gpio/gpio483/direction
echo out > /sys/class/gpio/gpio453/direction
```

To turn on the GPIOs, we wrote a '1' to their value file:

```
echo 1 > /sys/class/gpio/gpio<GPIO_NUMBER>/value
```

Specifically, for GPIOs 483 and 453, we executed:

```
echo 1 > /sys/class/gpio/gpio483/value
echo 1 > /sys/class/gpio/gpio453/value
```

With the GPIOs now accessible, we created a script to automate their control. We used the ‘vi‘ editor to write this script, granting executable permissions, and then tested it:

```
vi rgb.gpio
```

To facilitate the control of an RGB LED via GPIO, we created the ‘rgb.gpio‘ script. This script allows us to turn the RGB LED on or off by passing an argument to the script. Below is the content of the ‘rgb.gpio‘ script:

```
#!/bin/bash
# Script to control an RGB LED via GPIO

if [ "$#" -ne 1 ]; then
  echo "Usage: $0 <0 or 1>"
  exit 1
fi

GPIO_NUMBER=<YOUR_GPIO_NUMBER> # Replace with your GPIO number

if [ "$1" -eq 0 ]; then
  echo 0 > /sys/class/gpio/gpio${GPIO_NUMBER}/value
elif [ "$1" -eq 1 ]; then
  echo 1 > /sys/class/gpio/gpio${GPIO_NUMBER}/value
else
  echo "Invalid argument. Use 0 or 1."
  exit 1
fi
```

The script checks if an argument is provided and if it is either 0 (off) or 1 (on). Based on the argument, it sets the GPIO value accordingly. Make sure to replace ‘<YOUR_GPIO_NUMBER>‘ with the actual GPIO number you are controlling.

After creating the script, we granted executable permissions and then tested it:

```
chmod +x rgb.gpio
./rgb.gpio 1  # Turn on the RGB LED
./rgb.gpio 0  # Turn off the RGB LED
```

This script provided a convenient and efficient way to test the GPIO functionality and the integration of our software with the FPGA hardware.

7 Challenges Encountered and Overcome

7.1 Compatibility Issues with Operating Systems and Petalinux Versions

One significant hurdle we faced was finding compatible combinations of Ubuntu and Petalinux versions. After extensive testing with four Ubuntu versions (20, 22, 16, 18) and three different Petalinux releases, each pairing resulted in unique errors. Our solution, borne out of intensive research and experimentation, was the use of Ubuntu 16 LTS and Petalinux 2019.2.

7.2 Limitations in Hardware Design: Ethernet PHY Integration

Our goal to enhance the system with an Ethernet PHY was challenged by the removal of the necessary IP in Vivado versions post-2019.1. Time constraints hindered our attempts to integrate this IP across different versions, leading to a compromise in our quest for a robust system.

7.3 Device Tree Complications

Navigating device trees, especially for components like Quad SPI, proved complex. In later Petalinux versions, manual editing of device trees was essential, as incorrect configurations often resulted in boot failures. This challenge directed our choice towards Ubuntu 16 LTS, which offered smoother integration.

7.4 Understanding the Embedded Linux Environment

The embedded Linux environment, encompassing elements like U-Boot and device trees, presented a continual learning curve. Our journey involved assembling a complex puzzle of new concepts and technical details.

7.5 Challenges with the Ubuntu Linux Work Environment

The Ubuntu Linux work environment, while powerful, posed its own challenges, especially in tool installation and dependency management. This experience highlighted the intricacies and steep learning curve associated with embedded system development compared to more user-friendly GPOS like Windows.

7.6 Concluding Thoughts

Despite the daunting challenges, the project was instrumental in our learning and growth, pushing us into uncharted territories of adaptation and deeper understanding of embedded systems. It was a journey marked not just by technical endeavor but also by persistence, innovation, and discovery.

8 Future Directions

As we look forward to the future of this project, there are numerous exciting directions and enhancements to consider. While the immediate focus remains on integrating Ethernet capabilities to enhance connectivity, the potential applications of Petalinux on FPGA extend far beyond this.

Advanced Networking and Communication

The inclusion of advanced networking features such as support for 5G technology and IoT protocols can transform the system into a powerful tool for edge computing and IoT applications. This enhancement would enable real-time data processing and management in various sectors like smart cities, industrial automation, and healthcare monitoring.

Artificial Intelligence and Machine Learning

Integrating AI and machine learning capabilities into the Petalinux system could open avenues in areas such as predictive maintenance, image and speech recognition, and autonomous systems. Leveraging FPGA's parallel processing capabilities, AI algorithms can be executed efficiently, offering accelerated computing for complex tasks.

Energy-Efficient and Sustainable Computing

Exploring energy-efficient design methodologies within the FPGA environment is crucial for sustainable computing. This includes optimizing power consumption, implementing dynamic power management techniques, and focusing on green computing practices.

Educational and Research Applications

Finally, the project can pivot towards educational and research applications, serving as a valuable tool for teaching embedded system design and facilitating cutting-edge research in universities and institutions.

In conclusion, the potential advancements for the Petalinux on FPGA project are vast and varied. Each of these future directions not only enhances the system's capabilities but also contributes to the broader field of embedded systems, paving the way for innovative solutions to complex problems.

9 Conclusion

In conclusion, this project has been a profound learning experience, significantly enhancing our understanding and skills in the realm of embedded systems. Despite the encountered challenges, the successful completion of the project stands as a testament to the efficacy of our approach and our resilience in the face of technical adversities. The project not only achieved its initial goals but also laid a strong foundation for future advancements. The knowledge and experience gained will undoubtedly serve as invaluable assets in our ongoing pursuit of excellence in embedded system development.