AGSTU **Examensarbete/Thesis**

Author Freddy Avaria		Assignement FPGA & TEIS Thesis pre-study	
Email address freddy_avaria@hotmail.com	Controlled by	Version 1.0	File Freddy_Avaria_FPGA_&_TEIS_Thesis_A. docx

AGSTU VOCATIONAL UNIVERSITY

PXE-Based Remote Reboot for Speed Enforcement System

FPGA & TEIS (Applied Electronics in Embedded Systems) Thesis Pre-study - part A

Freddy Avaria

2025-08-10

ABSTRACT

This pre-study is the first part of a larger thesis project aimed at setting up and possibly implementing a PXE server, which should function as a remote terminal for rebooting the system without needing to be physically on-site. Continuing is a thorough explanation on the subject, including its history, how it works, real-world applications and an understanding of the project itself, including its goals, project scope, architecture, verification and validation methods and an activity plan.

Content

1 REQUIREMENTS

Table 1: Requirement list

Requirement	Description	Done
Pre-study/initial report (also to be included in the final report)		
1	**Initial report** The deliverable must be in Word format and written in either Swedish or English. The report should include the following: 1. **Summary / Abstract:** A brief overview of the project and its objectives. 2. **Project goals:** A clear description of what the project is intended to deliver. 3. **Specification:** A preliminary specification describing the system or IP component to be delivered (may be adjusted as needed). 4. **Component reuse:** Identify components that may be reused, including references. Also, review previous thesis projects as supporting material. 5. **Architecture:** Outline a proposed hardware (HW) and/or software (SW) architecture. 6. **Verification and Validation:** Describe how the project will be verified and validated. The methods can be revised during the project. 7. **Activity and Time Plan:** Create a timeline for the project in accordance with the course syllabus. Follow up on the plan and present the results in the final report, including a calculation of labor costs (800 SEK/hour). 8. **Role Distribution in Group Work:** If the project is conducted in a group, specify which part of the work you are responsible for. All group members must submit full copies of the project. 9. **Chapter Division and Structure:** Propose a chapter outline for the final report and a folder structure for the project delivery (e.g., source code, documentation, and test results). See Final Delivery, item 3. 10. **Scope:** Maximum six pages. **Note: If the project is more theoretical or focused on another type of embedded system, the initial report may differ.**	
2	The initial report must be a standard report in Word format. The deliverable must be submitted to Lennart (lennart.lindh@agstu.com).	

	The filename should be “firstname_lastname_thesis_A.zip”. Use "Examensarbete" as the subject line.	
End of pre-study (must be approved before proceeding to the next phase)		

2 INTRODUCTION

A PXE server (short for Preboot Execution Environment server) is a system that allows computers to boot over a network rather than from a local storage device like a hard drive or SSD. This capability is crucial in environments where managing physical boot media is inefficient or impractical.

3 HISTORY

PXE, or Preboot Execution Environment, was introduced by Intel in the late 1990s as part of its Wired for Management initiative. Its purpose was to let computers boot directly from a network rather than from a local drive, making it easier to deploy operating systems and manage diskless workstations. In a PXE boot, the network interface card's firmware uses DHCP to get an IP address and the location of a boot server, then downloads a bootloader via TFTP before the operating system starts. Through the 2000s, PXE became a standard feature in most network cards and BIOS firmware, widely used in enterprises for centralized OS installation, recovery, and thin client setups. With the shift to UEFI firmware, PXE evolved to support newer protocols like HTTP Boot for faster and more secure network starts, but its core role—network-based booting and deployment—remains the same today.

A basic PXE setup can be seen in the following figure.

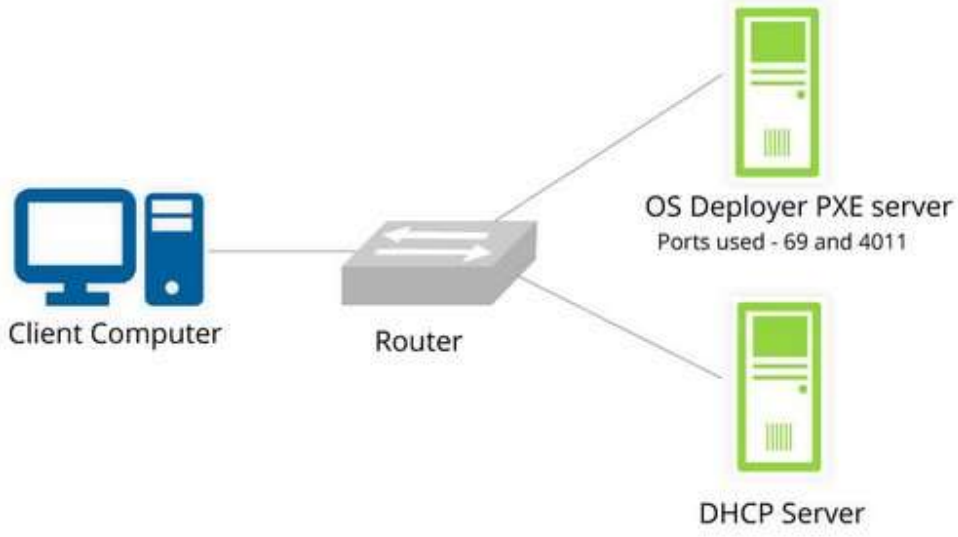

Figure 1: PXEserver setup

4 PROJECT GOALS

This project focuses on designing, implementing, and optimizing a network-based Preboot Execution Environment (PXE) specifically for embedded systems used in speeding systems. Using the Raspberry Pi 5 as the client and a 64-bit industrial computer as the PXE server, it enables reliable, efficient, and secure network booting without local physical storage. This allows embedded devices to reboot remotely and quickly by fetching boot images over the network, eliminating the need for physical drives or manual intervention.

By centralizing OS and boot files on a large external hard drive connected to the PXE server, the system supports diskless operation, streamlined deployment, and easier maintenance—especially useful for IoT and embedded devices in remote or inaccessible locations. Beyond basic PXE functionality, the project explores performance enhancements by testing different network protocols, evaluating boot time and reliability, and leveraging FPGA-based hardware acceleration to offload network processing tasks, reducing CPU load and improving transfer speeds.

Security is a key focus, with implementation of signed boot images and encrypted transfers to protect the boot process from network threats. The project aims to develop a scalable PXE infrastructure that supports multiple embedded clients simultaneously, enabling fast remote reboot and provisioning in resource-constrained environments.

This work falls under the categories of embedded systems and FPGA-based network systems, with significant aspects of firmware and bootloader design, integrating network protocols, hardware acceleration, and secure boot mechanisms to advance embedded device management in speeding systems.

5 SCOPE OF THE PROJECT

This chapter outlines the scope of the proposed project, defining its technical boundaries, objectives, and intended outcomes. The work centers on the design and optimization of a Preboot Execution Environment (PXE) for embedded systems, specifically using a Raspberry Pi 5 as the network-booted client and a 64-bit industrial computer as the PXE server. While PXE is a well-established technology in enterprise environments, its adaptation and optimization for embedded platforms—particularly when enhanced through FPGA-assisted network handling—presents both engineering challenges and opportunities for innovation.

I. **System design and architecture:** The project begins with the design of the PXE boot infrastructure. This will involve setting up a PXE server running on a 64-bit industrial computer equipped with PXE and TFTP services, connecting an external 6 TB hard drive for centralized OS image storage, and configuring a Raspberry Pi 5 to initiate a network boot and remote reboot through its firmware. The complete boot and reboot sequences, from power-on or reset to OS load, will be documented, including DHCP or proxyDHCP interaction, the use of TFTP or HTTP for image transfer, and kernel initialization procedures. Emphasis will be placed on enabling diskless operation and rapid, reliable remote rebooting without the need for physical storage media.

II. **PXE Boot Implementation:** The implementation phase will configure the PXE server software, such as *dnsmasq* or *tftpd-hpa*, and prepare PXE-compatible OS images for the Raspberry Pi 5. These images will include the kernel, *initramfs*, and necessary configuration files, all stored and managed on the external hard drive, optimized for fast remote deployment and reboot scenarios.

III. **Network Protocol Evaluation:** Following the basic implementation, the project will evaluate network protocol performance by comparing PXE boot and reboot speeds using TFTP, HTTP, and NFS. This stage will analyze network latency, boot time, and reliability under varying loads, and it will include the development of protocol selection logic to achieve optimal performance for both initial boots and remote resets.

IV. **Scalability and Multi-Client Support:** Scalability will then be addressed by enabling the PXE server to support multiple Raspberry Pi clients booting and rebooting simultaneously. This will include monitoring server resource usage, identifying potential bottlenecks, and proposing strategies to maintain performance and reliability under increased remote management demand.

V. **Security Enhancements:** The next phase will focus on security enhancements, such as investigating UEFI Secure Boot and implementing signed boot image validation for PXE in embedded devices. Secure transfer methods, including HTTPS boot or

encrypted NFS, will be explored to improve resilience against network-based attacks during both boot and remote reboot operations.

VI. **FPGA-Based Acceleration (Specialization Tie-In):** To tie the project directly to the FPGA specialization, certain network processing tasks—such as TFTP block handling and packet parsing—will be offloaded to an FPGA module. Performance benchmarks will compare this hardware-accelerated approach with a purely software-based PXE implementation, measuring improvements in transfer speed, CPU load, and boot and reboot times.

VII. **Testing and Performance Analysis:** Testing and performance analysis will be conducted throughout the project. Key metrics will include boot and reboot times, network throughput, error rates, and resource utilization. Experiments will be performed under varying network conditions, and results will be analyzed statistically to identify performance trends and validate the effectiveness of proposed optimizations.

VIII. **Documentation and Deliverables:** The final stage will focus on delivering complete documentation, including system architecture diagrams, configuration scripts, image preparation guides, performance reports, and a set of recommendations for deploying PXE solutions with remote reboot capabilities in embedded system environments.

The scope described herein moves beyond the basic implementation of PXE booting. It incorporates performance benchmarking, protocol evaluation, security enhancements, scalability testing, and hardware acceleration through FPGA integration. By defining these elements, this chapter sets the framework for a structured and methodical approach to developing a high-performance, secure, and scalable network booting solution suitable for resource-constrained or specialized embedded deployments.

6 ARCHITECTURE

This chapter describes the architecture of the PXE-based network boot and remote reboot system designed for embedded applications, with a focus on speeding systems. The architecture integrates hardware and software components to enable a Raspberry Pi 5 client to boot entirely over the network from a centralized PXE server. The PXE server runs on an LP-173 industrial computer equipped with an Intel® processor and Windows 10 IoT Core. This server hosts PXE and TFTP services via Windows-compatible server software and is connected to a 6 TB external hard drive used for storing all operating system images and boot files.

Upon receiving a PXE boot request from the Raspberry Pi 5, the PXE server retrieves the required boot image from the hard drive and transfers it to the client using network protocols such as TFTP or HTTP. To optimize this process, an FPGA module is placed in-line between the PXE server and the network switch. The FPGA transparently intercepts PXE-related packets, performing hardware-accelerated operations such as packet parsing, checksum verification, TFTP block prefetching, and data buffering. By offloading these repetitive network tasks from the server CPU, the FPGA reduces transfer latency, improves boot throughput, and maintains performance when multiple clients boot simultaneously.

Because the FPGA operates as a transparent inline accelerator, no custom driver development is required for Windows 10 IoT Core. This allows the server's PXE and TFTP software to operate normally, while still benefiting from FPGA-level network acceleration. The resulting architecture combines centralized diskless boot management, secure and scalable network operation, and hardware-assisted performance enhancements, forming the basis for the system's verification and validation stages.

The whole setup is shown in the figure below:

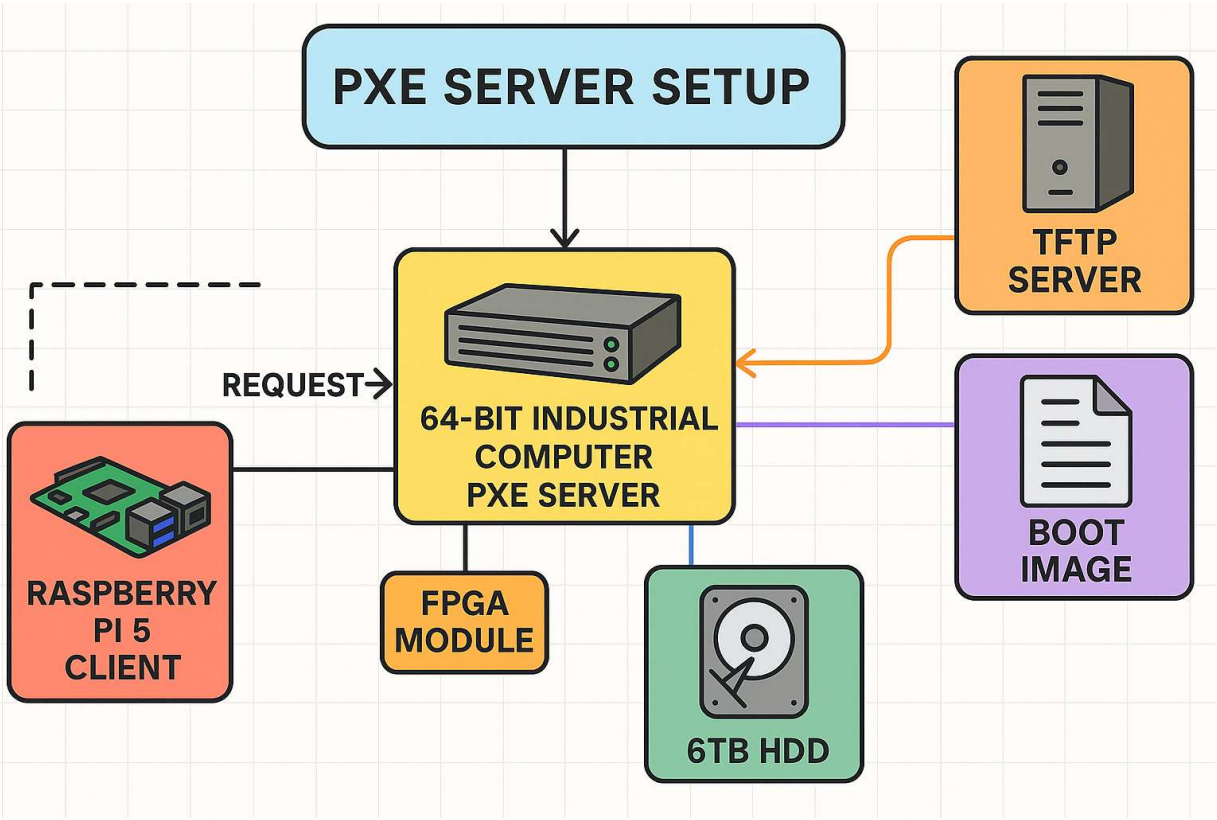

Figure 2: Preliminary architecture

7 VERIFICATION AND VALIDATION

This chapter outlines the methods used to verify that system components meet design requirements and validate the overall PXE-based network boot and remote reboot system's performance and reliability under real conditions. It includes testing, measurement, and analysis of the PXE infrastructure, FPGA acceleration, scalability, and security features.

7.1 Verification

7.1.1 Hardware Setup

Raspberry Pi 5:

Confirm that the Raspberry Pi 5 firmware supports PXE/network boot. Access the bootloader settings and enable network boot if needed. Test initial PXE boot requests in isolation to ensure the device sends correct DHCP and PXE broadcast packets.

64-bit Industrial Computer (PXE Server):

Verify installation of PXE server components (e.g., dnsmasq or isc-dhcp-server and tftpd-hpa). Check that DHCP and TFTP services start automatically and are configured to listen on the correct network interface. Confirm server responds to PXE client requests with proper boot files.

External 6 TB Hard Drive:

Confirm that the hard drive is mounted correctly on the PXE server. Perform read/write tests on the directory containing boot images. Ensure that the drive is accessible by the PXE/TFTP services and has appropriate permissions.

FPGA Development Board:

Verify that the FPGA development tools (Vivado, Quartus) successfully synthesize the hardware design. Program the FPGA and test basic functionality (e.g., LED indicators, register reads). Confirm the FPGA correctly interfaces with the PXE server's network stack and processes packets as expected.

Ethernet Cables and Switch:

Test physical connectivity using link lights and ping tests. Use cable testers if available to check wiring quality. Confirm that the switch routes traffic properly between PXE server and Raspberry Pi(s) on the same subnet.

7.1.2 Software Setup

PXE Server Software:

Verify configuration files for DHCP and TFTP services are syntactically correct. Use system logs to check for errors during service startup. Run manual tests by initiating PXE requests from the client and confirming the server responds appropriately.

Boot Images Preparation Tools:

Verify that kernel images, initramfs, and config files are correctly generated and placed in the PXE server's boot directory. Use checksums (e.g., SHA256) to confirm file integrity after transfers or modifications.

FPGA Development Environment:

Verify the build process completes without errors. Use simulation tools to validate timing and logic. Confirm the deployed bitstream enables the intended packet processing offload functions.

Security Tools:

Test creation of signed boot images using available signing tools. Confirm that signature verification passes on boot. Verify encryption and decryption mechanisms work correctly on boot files.

7.2 Validation

7.2.1 Hardware Setup

Raspberry Pi 5

Perform full network boot cycles from power-on and remote reboot commands. Validate that the Pi boots correctly into the OS using PXE each time without falling back to local storage. Log boot times and check for any failures or retries.

PXE Server

Simulate multiple simultaneous PXE boot requests (e.g., from multiple Raspberry Pis) and monitor server CPU, memory, and network utilization. Validate the server can handle load without timeouts or dropped packets.

External 6 TB Hard Drive

Measure data throughput during simultaneous boot image transfers. Confirm no degradation in speed or data corruption under load. Perform stress tests simulating frequent read/write operations during client boots.

FPGA Module

Benchmark boot and reboot times with and without FPGA offload enabled. Measure CPU usage on the PXE server to confirm workload reduction. Validate packet processing latency improvements.

Network Infrastructure (Switch, Cables)

Use tools like iperf and Wireshark to measure network latency, packet loss, and throughput during boot/reboot operations. Confirm network stability over extended testing periods.

7.2.2 Software Setup

PXE Server Software

Validate the full PXE boot sequence including DHCP lease, TFTP or HTTP transfer, and kernel initialization by observing logs and client behavior. Perform tests under different network conditions (e.g., congestion, packet loss).

Boot Images

Validate that the booted OS initializes correctly, mounts necessary filesystems, and runs startup scripts. Perform functional tests of the OS to confirm system readiness post-boot.

Network Monitoring and Testing Tools

Use Wireshark to capture and analyze PXE and TFTP traffic to confirm protocol compliance. Use iperf to measure network throughput and latency during boot operations. Log and analyze error rates.

Security Implementations

Conduct penetration testing attempts to access or tamper with boot images. Verify that unauthorized images are rejected during boot. Test encrypted transfers to confirm data confidentiality and integrity.

8 ACTIVITY AND TIME PLAN

This chapter outlines the week-by-week activity plan for the development of the PXE-based network boot and remote reboot system for embedded speeding applications. The plan defines the sequence of tasks, estimated effort, and time allocation required to progress from initial research and setup through implementation, testing, and documentation. By structuring the work into clear phases, the activity plan ensures steady progress, accommodates learning curves in networking and FPGA integration, and provides a framework for meeting project milestones within the allotted timeframe. This is depicted in the table and list below.

Table 2: Activity plan

Week	Activity	Time (h)
1 - 3	Background Research & Requirements Definition	10h/week
4 - 5	Network Fundamentals & Lab Setup	20h/week
6 - 7	PXE Server Hardware Setup	25h/week
8 - 9	PXE Service Installation	25h/week
10 - 11	Raspberry Pi 5 PXE Boot Configuration	25h/week
12 - 13	OS Image Preparation & Testing	25h/week
14 - 15	Performance Evaluation of Network Protocols	25h/week
16 - 17	FPGA Offloading Implementation	25h/week
18	Remote Reboot Feature for Speeding Systems	25h/week
19 - 20	Security Enhancements	25h/week
21	Scalability Testing	25h/week
22 - 23	Verification, Validation & Refinement	25h/week
24	Documentation & Final Report	20h/week

Week 1–3: Background Research & Requirements Definition

Learn PXE basics, firmware design, embedded boot processes, FPGA acceleration.

Difficulty: Medium — networking is new, but most is theoretical at this stage.

Week 4–5: Network Fundamentals & Lab Setup

Set up isolated PXE test network with VLAN or dedicated switch.

Difficulty: High — network conflicts and VLAN setup can be tricky for beginners.

Week 6–7: PXE Server Hardware Setup

Assemble 64-bit PXE server with 6 TB HDD, install base OS.

Difficulty: Medium — standard hardware setup, unless industrial hardware has quirks.

Week 8–9: PXE Service Installation

Install DHCP/proxyDHCP, TFTP, and PXE services.

Difficulty: High — DHCP/TFTP conflicts are common and frustrating.

Week 10–11: Raspberry Pi 5 PXE Boot Configuration

Update firmware, configure network boot, verify first boot from PXE.

Difficulty: Medium — Pi firmware setup is well-documented, but network boot can still fail unexpectedly.

Week 12–13: OS Image Preparation & Testing

Build PXE-compatible OS images, ensure full boot to desktop/CLI.

Difficulty: Medium — kernel/initramfs alignment requires attention to detail.

Week 14–15: Performance Evaluation of Network Protocols

Benchmark TFTP, HTTP, and NFS boot speeds.

Difficulty: Low — mostly repetitive measurement and comparison.

Week 16–17: FPGA Offloading Implementation

Integrate FPGA for packet handling, measure CPU load and speed improvements.

Difficulty: High — HDL dev + network integration is complex.

Week 18: Remote Reboot Feature for Speeding Systems

Implement and test remote reboot trigger with watchdog fallback.

Difficulty: Medium — logic is manageable, but reliability across reboots is tricky.

Week 19–20: Security Enhancements

Add signed images, HTTPS/NFS encryption, and Secure Boot.

Difficulty: High — cryptographic boot requires toolchain familiarity and PXE extensions.

Week 21: Scalability Testing

Boot multiple Raspberry Pi clients at once, stress-test PXE server.

Difficulty: Medium — requires extra hardware but not conceptually hard.

Week 22–23: Verification, Validation & Refinement

Difficulty: Medium — systematic but may reveal regressions. nfirm requirements met, fine-tune configs, retest performance.

Week 24: Documentation & Final Report

Complete thesis writing, final diagrams, and presentation prep.

Difficulty: Low — labor-intensive but straightforward if you've documented as you go.

9 REVISION HISTORY

Revision	Comments	Responsible	Date
1.0	First release	Freddy A.	2025-08-10

10 REFERENCES

Wikipedia – PXE (Preboot Execution Environment) (Jun.25, 2025): *https://en.wikipedia.org/wiki/Preboot_Execution_Environment*

Heimdal - What Is PXE Boot and How Does It Work? (Jun.27, 2025): *https://heimdalsecurity.com/blog/what-is-pxe-boot/*

Chat GPT - AI helper (Jun.25, 2025): *https://chatgpt.com/*

GitHub - PXE (Jun.30, 2025): *https://github.com/tianocore/tianocore.github.io/wiki/PXE*

Runtime Recruitment - Optimizing FPGA Designs for High-Speed Networking Applications (Jul.30, 2025): *https://runtimerec.com/optimizing-fpga-designs-for-high-speed-networking-applications/?utm_source=chatgpt.com*

The PXE Playbook: Transform Your OS Deployment Strategy (Aug.2, 2025): *https://www.youtube.com/watch?v=5qyTLfJrQM4&t=17s*