Republic of Tunisia
Ministry of Higher Education
And Scientific Research

University of Sfax

National School of Electronics
And Telecommunications of sfax

ENET'COM
المدرسة الوطنية للإلكترونيك و الإتصالات بصفاقس
ECOLE NATIONALE D'ELECTRONIQUE ET DES TÉLÉCOMMUNICATIONS DE SFAX

Industrial computer Engineering

2024/2025

Internship Report

Presented at

National School of Electronics and Telecommunications of Sfax

Industrial Computer Engineering

by

Nawres Barhoumi

Adding Dotnet web server onto the Elco35 board

ELCO solutions

Internship completed within the Company: Elco-Solutions

Internship period: 08/07/2024-30/08/2024

Academic year: 2024-2025

Acknowledgment:

I would like to extend my deepest gratitude to everyone who supported me throughout this project.

I am especially thankful to Mr. Kamel Ghalgaoui and Mr. Ahmed Besbes for their invaluable guidance and encouragement, which significantly contributed to the successful completion of this work.

To Mr. Kamel Ghalgaoui, I am truly appreciative of the opportunity to be involved in your forward-thinking and inspiring projects. Your mentorship and vision have been instrumental in expanding my knowledge and fostering my passion for innovation.

I would also like to express my heartfelt thanks to the entire team at Elco-Solutions. Your warm reception and the innovative atmosphere you fostered made it possible for creativity to flourish.

The collaborative spirit and constant support from the team provided an enriching experience, which greatly enhanced my learning and professional growth.

Without your input, this project would not have been as successful. Finally, I would like to acknowledge everyone who contributed, directly or indirectly, to the completion of this project.

Your encouragement, assistance, and trust motivated me to push my limits and achieve this milestone.

Table of Contents:

List of figures:

General Introduction:

With the rapid evolution of embedded technologies and connected systems, it is becoming increasingly crucial to develop solutions for effective remote management and control of devices. This project addresses this need by integrating a .NET web server on the Elco35 board, an embedded platform used in industrial environments.

The primary goal is to provide an intuitive web interface that allows users to monitor and control the board's features in real time via a simple browser.

The integration of this web server offers a practical solution for remote management of electronic equipment. It not only centralizes control operations but also enhances the flexibility and efficiency of industrial processes by making devices more accessible and interactive.

This project aligns with the growing trend towards the Internet of Things (IoT) and industrial automation, where the ability to manage systems remotely has become a key requirement.

The implementation of this web server will enable users to optimize the management of embedded systems while facilitating innovation in areas such as automation, remote monitoring, and proactive maintenance.

Chapter 1: Pre-study

I. Introduction :

This chapter provides a detailed introduction to "Elco-Solutions," the company overseeing this project. We then identify the primary challenge that our project addresses and clearly define the goals we intend to accomplish.

Additionally, we describe the methodology we will employ throughout the project and outline the planned timeline for our activities.

II. Company Presentation :

1. Company Profile :

Elco Solutions is a privately owned company established in 2015 with a focus on software development and digitalization services.

Elco solutions covers the entire software development process, from the communication layers up to the application layers.

it provides a complete solution for embedded software, focusing on factory automation and industrial communication systems.

their software experts support the design and development of customer-specific projects subject to customer coding guidelines.

Figure 1:Elco-Solution Logo

III.Field of company:

Elco -solutions has three fields:

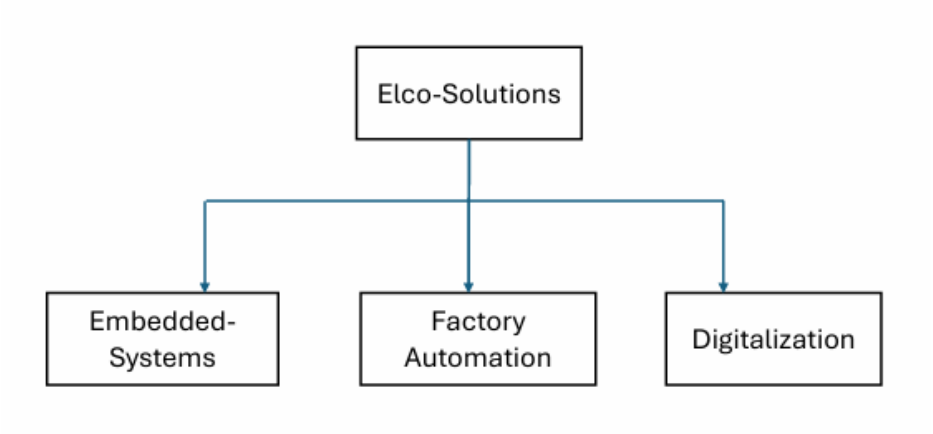


Figure 2:Field of company

1. Embedded Systems:

They offer fast, affordable, high-quality consulting and development services for any operating system, any processor, any application, any communication protocol — as well as specific expertise in industrial automation, automotive, and IoT. Find out how they can accelerate your development and meet your budget, schedule, and technical requirements.

2. Factory Automation:

Elco Solutions covers the entire software development process from the hardware abstraction layers up to the application layers. They provide a complete solution for factory automation and industrial communication systems. Their software experts support design and development of customer specific projects subject to customer coding guidelines.

3. Digitalization:

Elco Solutions provides production intelligence consultancy supporting manufacturing organizations in improving productivity and reducing production time and costs. Their Realtime factory floor monitoring and supervision tool features capability to measure key performance indicators enabling data driven decision and quantified improvement in performance.

IV. Elco-Solution's services:

The services offered by Elco-Solutions include consulting, development, testing and validation, as well as prototyping, to meet the specific needs of our clients.

They provide fast, affordable, and high-quality consulting and development services for any OS, processor, or application.

With extensive experience in operating systems, processors, and protocols—along with specific expertise in industrial automation, automotive, and IoT—Elco Solutions is your ISO-9001-certified consulting and development partner.

With over 15 years of embedded experience and the successful completion of more than 40 major contracts, discover how they can accelerate your development and meet your budget, schedule, and technical requirements.

1. Développent :

-Design and develop hardware and software for embedded systems.

-Augment staff with anywhere with their experienced software developers.

-Create a long-term outsourcing partnership that saves both time and money.

2. Testing and validation:

-Create optimum testing combination based on system requirements, technologies and infrastructure.

-Define and execute unit testing.

-Continuous integration and automated testing.

-Perform HIL/SIL test and validation.

3. Prototyping :

-Develop hardware and /or software prototypes for proof of concept.

-Evaluate feasibility and usability of new innovative electronics concepts.

-Develop and produce evaluation boards for new processors and microcontrollers.

4. Consulting:

-Overcome debug hurdles and solve design problems with consultants who have significant software development experience and technology expertise.

-Develop specifications, architecture and strategies as well as select OS and embedded processors.

V. Specific areas of expertise In Embedded Systems:

1. Automotive :

Elco Solutions offers a complete software delivery and architectural planning service from a single source.

Their proposal is combining core competencies in architectures, real-time and communication with cost-optimized development capacities in automotive software.

their AUTOSAR Experts will support accelerating your time to market.

2. Internet Of Things:

Elco Solutions adapts complex communications systems solutions into industries to solve their Digital Transformation challenges.

Their engineer the connectivity technologies that make sense for the industries and help our clients build, test and deploy new connectivity solutions that accelerate their business.

3. Factory Automation:

Elco Solutions covers the entire software development process from the hardware abstraction layers up to the application layers.

They provide a complete solution for factory automation and industrial communication systems.

Their software experts support design and development of customer specific projects.

VI. Specific areas of expertise In Factory Automation:

1. Embedded Linux:

Elco Solutions Team can develop complete Linux board support Packages allowing their customers to stay focused on the development for their specific applications. they can help in the following areas: bootloader and Linux kernel porting, development of custom device

drivers, system integration, boot time optimization, factory flashing and in-field system upgrade, support to application developers.

2. Application Development:

Embedded software development takes in consideration hardware restrictions (e.g. memory, processing power, NAND flash as a storage medium). Elco Solutions has many years of experience in developing application software for embedded systems and takes on partial or full responsibility for application development. Customers could also use our competences to support their internal development team.

3. Network / Field Buses:

Embedded systems often need to communicate with the outside world. they have already designed a variety of solutions for customers that are based on real-time Ethernet communication such as EtherCat, Profinet etc. their expertise includes TCP/IP based protocols such as Modbus and peer to peer communication such as IO-Link.

4. Internet of Things:

Communication is nowadays a must-have feature for most embedded systems in the industrial and home automation. Elco Solutions expertise in wireless and wired communication protocols ensures proper and efficient implementation of robust data collection systems. From Sensor and actuators nodes to the cloud Elco solutions offers a complete service from a single source for your IoT projects.

VII. Professional Team:

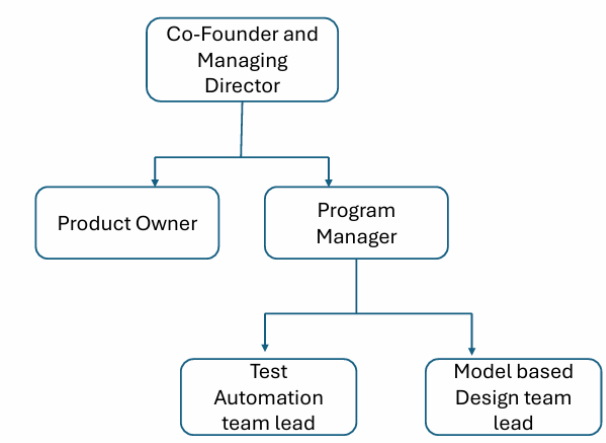


Figure 3:Professional Team

1. Locations Worldwide:

Elco-Solution operates across four countries, with locations in Tunisia, France, Germany, and USA.

2. Quality & Management Information System:

The Integrated Management System (IMS) at Elco Solutions is certified according to the international standard ISO 9001:2015 & ISO 27001/2013.

IMS covers the entire operations for marketing, sales, design, development, and support of embedded software.

Providing high quality software products and software development services has always been a top priority for Elco Solutions.

This formal certification of their extensive IMS, covering all aspects of their operations as per ISO9001:2015 & ISO27001:2013 is an additional step to assure customer satisfaction and continuous improvement by conducting both internal and external audits periodically.

VIII. Project presentation :

1. Problem statement:

As part of this project, several issues have been identified in the industry, particularly in embedded environments. First, the lack of a remote management interface for embedded systems like the Elco35(PhyGATE Tauri-S) board complicates monitoring and maintenance without direct intervention. Additionally, integrating web applications into these environments is often challenging due to resource limitations, so to address these issues, the proposed solution involves adding a Dotnet web server to the Elco35 board.

IX. Project Overview and Objectives:

The goal of this project is to integrate a Dotnet web server into the Elco35 board using the Yocto Project framework. The process includes several steps, ranging from fetching and building the Yocto environment to configuring the web server to operate efficiently on the embedded system. Once the project is completed, the Elco35 board will be able to host a web-based interface, enabling remote control and interaction with the system via a web application.

1. Objectives:

Fetch and Build the Elco35 Yocto Project:

Download and compile the Elco35 Yocto project to establish the necessary foundation for development.

Create a Build Environment for Applications:

Set up an environment that allows building applications without needing to rebuild the entire image each time, utilizing the Extensible SDK for efficiency.

Write a Yocto Recipe for the Web Server:

Develop a Yocto recipe that integrates the Dotnet web server into the Elco35 environment, ensuring compatibility and functionality.

Add a System Service File for the Dotnet Web Server:

Create and configure a systemd service file to manage the Dotnet web server, ensuring it can be easily started, stopped, and monitored.

Ensure Automatic Start of the Web Server on Boot:

Configure the Dotnet web server to start automatically when the system boots, providing seamless operation.

Use Nginx as a Reverse Proxy:

Implement the Nginx server to act as a reverse proxy, routing requests to the Dotnet web server, improving performance and security.

Create a Recipe to Compile a Web Application Using Dotnet:

Develop a Yocto recipe to compile a web application using Dotnet, ensuring it can be deployed and run on the Elco35 board.

X. Conclusion:

Through this introductory chapter, we had a general vision of the company. In addition, we defined our problem in order to find the solution appropriate. The next chapter will focus on the analysis and specification of requirements, as well as on the design of our future system.

Chapter 2: Building an Embedded Operating System for the Elco35 Board

I. Introduction :

Building an embedded operating system is a critical step in ensuring that the hardware platform, such as the Elco35 board, can perform optimally for its intended applications. Embedded systems often require a tailored operating system that balances performance, resource efficiency, and functionality. The Elco35 board, which is designed for embedded applications, necessitates a customized Linux-based operating system to manage its components and run specific applications, such as the Dotnet web server.

This chapter focuses on the process of building an embedded operating system for the Elco35 board, utilizing tools such as the Yocto Project, which provides a powerful framework for creating a customized OS.

in this chapter we will show you how to build and to boot the Linux image by Yocto project.

II. Installing Linux with a Dual Boot Configuration (Ubuntu 20.04):

Here are the steps for installing Linux with a dual boot configuration and reserving 150GB of disk space:

Create a Bootable USB Drive: Start by creating a bootable USB drive with Ubuntu 20.04 using a tool like Rufus.

Figure 4:Ubuntu LOGO (20.04 version)

Ubuntu is a Linux distribution, or rather, an operating system based on the Linux kernel.

Reserve Disk Space: Before installing Ubuntu, shrink your existing partition and make sure to reserve at least 150GB of unallocated space for the new operating system.

And then, restart your computer and boot from the USB drive by accessing the boot menu. Select the option to "Install Ubuntu alongside your existing OS" and allocate 150GB of reserved space for Ubuntu. After the installation, the bootloader (GRUB) will be configured automatically, allowing you to choose between Ubuntu and your current OS upon reboot.

III.Tools for Building an Embedded Linux System:

To build an embedded Linux distribution, developers can use a variety of tools, each with its strengths and applications. For the Elco35 board, the Yocto Project was selected due to its flexibility and scalability.

Other common tools in the embedded space include Buildroot and OpenWRT, but Yocto stands out for its ability to create highly customizable images suited to complex hardware environments.

Yocto Project: A powerful framework that provides flexibility and customization for building embedded Linux systems. Yocto allows the creation of highly specialized Linux distributions for various hardware platforms.

Buildroot: A simple, efficient, and fast tool to generate embedded Linux systems. It is often used for smaller systems with fewer customization needs.

OpenWRT: Primarily used for network devices, OpenWRT is a Linux distribution designed for routers and embedded devices, providing strong networking support.

IV. What is Yocto Project ?

The Yocto Project is an open-source collaboration project that helps developers create custom Linux-based systems that are designed for embedded products regardless of the product's hardware architecture. Yocto Project provides a flexible toolset and a development environment that allows embedded device developers across the world to collaborate through shared technologies, software stacks, configurations, and best practices used to create these tailored Linux images.

1. Principles of the Yocto Project:

The Yocto build system simplifies creating custom Linux distributions for embedded systems. It uses source code, defined by recipes, and user configurations tailored to the target hardware.

The output includes a complete Linux image, consisting of the OS, kernel, device tree, bootloader, and tools like the Software Development Kit (SDK), ready for deployment.

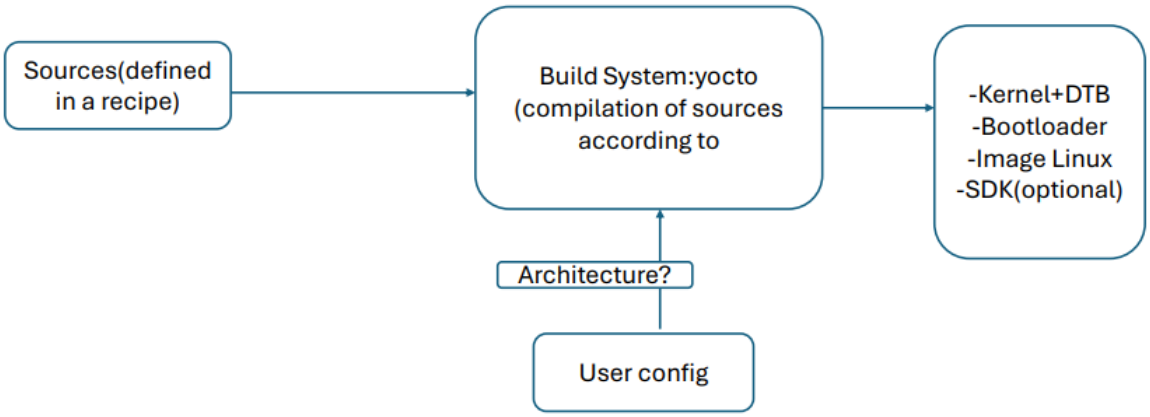
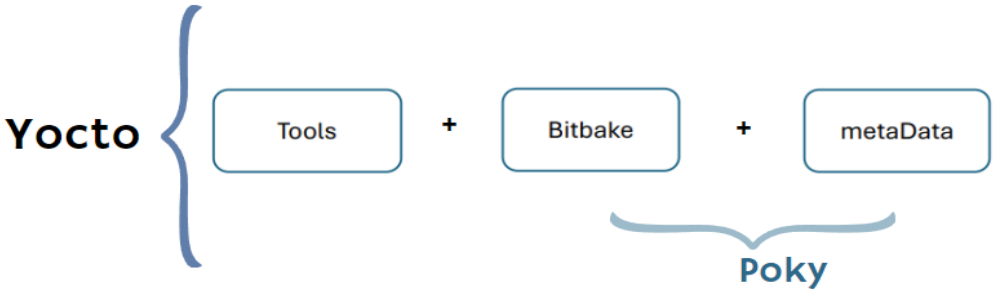


Figure 5:principle of Yocto project

The yocto project contains Tools +poky (It contains the build system (BitBake and OpenEmbedded Core))



V. - Components and Tools:

The Yocto Project employs a collection of components and tools used by the project itself, by project developers, and by those using the Yocto Project.

These components and tools are open-source projects and metadata that are separate from the reference distribution (Poky) and the Open Embedded Build System. Most of the components and tools are downloaded separately.

VI. Needed Components :

We need the following components to get started quickly:

- PhyGATE Tauri-S device (Elco35 board)
- 1x MicroSD Card with prepared prebuilt image
- 1x 24 VDC Mains-Adapter with a mating connector to device supply connector
- 1x LAN cable

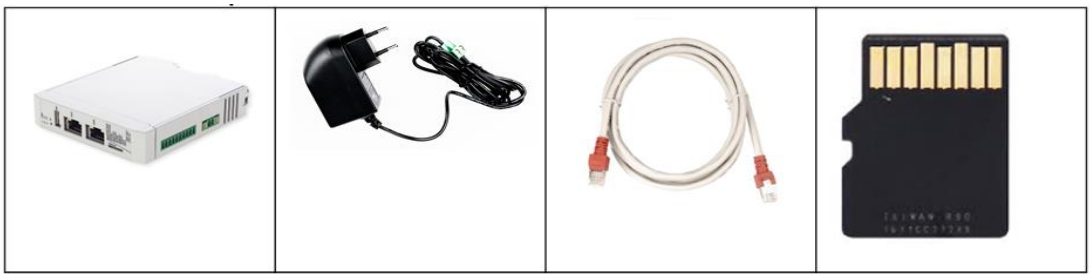


Figure 6:Needed Components

VII. Hardware Components :

1. Elco35 board (phyGATE Tauri-S):

The Elco35 board developed by PHYTEC company is based on the NXP i.MX 6ULL Cortex-A7 processor, which features a single-core architecture optimized for embedded applications. This processor, built on the ARM Cortex-A7 architecture, is designed to provide a balance between performance and energy efficiency, making it ideal for embedded systems that require effective resource management.

The operating system used for this board is Linux, built with the Yocto Project. The Yocto Project enables the creation of a customized Linux distribution tailored to the specific needs of the Elco35 board.

Figure 7:Elco35 board (phyGATE Tauri-S (based IMX6))

VIII. Electrical Connection:

The phyGATE Tauri-S(Elco35) has several interfaces and controls/displays which are shown in the following figure.

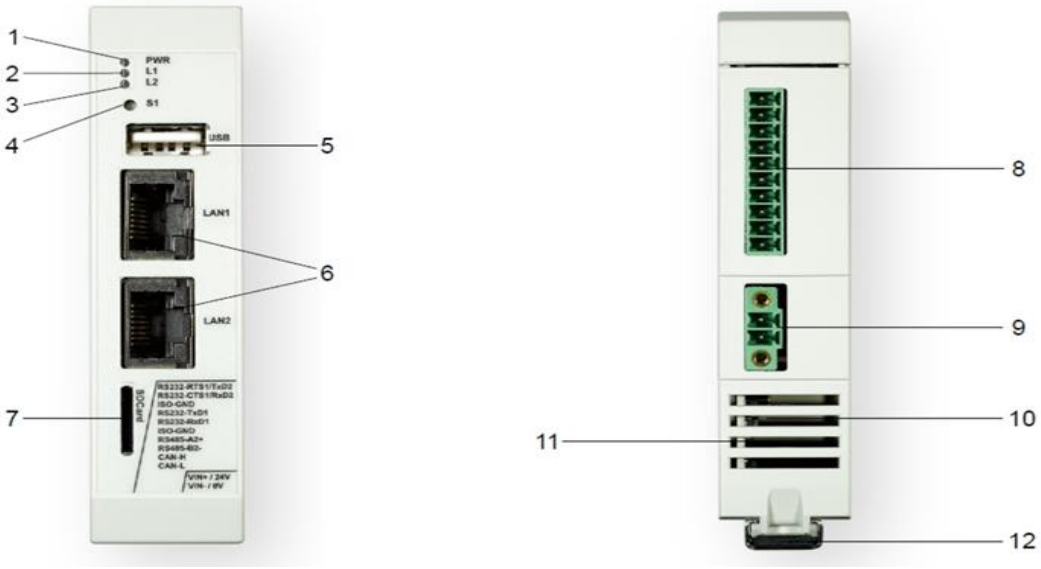


Figure 8:board Connection

IX. phyGATE Tauri-S Interfaces :

PB-03513-001.Ax (25mm housing)	PB-03513-002.Ax (50mm housing)	Controls/Displays
X	X	1. Power-LED green
X	X	2. User-LED red (freely configurable)
X	X	3. User-LED yellow (freely configurable)
X	X	4. Button (freely configurable)
X	X	5. USB interface (Typ A)
X	X	6. Ethernet interface (RJ45)
X	X	7. Micro-SD Slot
X	X	8. RS232, RS485, CAN interface (configurable)
X	X	9. Supply connection
X	X	10. Boot-Switch (Concealed, on opposite housing side)
X	X	11. Air vents
X	X	12. Extension interface under a top-hat rail and top-hat rail clip (35mm)
	X	(13.) SMA antenna connectors
	X	(14.) Internal miniPCIe slot (USB only)
	X	(15.) SIM card socket

Figure 9:phyGATE Tauri-S Interface List

X. Technical Data:

Hardware Specification	
CPU type	NXP i.MX 6ULL Cortex-A7
RAM	512 MB
eMMC	8 GB

Ethernet	2x 10/100 Mbit/s
USB	USB 2.0
CAN (optional)	max. 1 Mbit/s, isolated
Serial (optional)	1. 1x RS232+RTS/CTS isolated or 2. 2x RS232 (RxD/TxD) isolated or 3. 1x RS232+ 1x RS485 isolated
Mass storage	microSD card slot storage size max 128 GB, class10
Additional features	TPM chip, Temperature sensor
RTC	GoldCap for real-time functionality
User control elements	1x user button 2x configurable user LED 1x power status LED
Software Specification	
Operating system	Linux (Yocto)
Security Features	Device-Management / Cloud-Update concept

Environmental Data	
Storage temperature	-20 °C - +70 °C
Operating temperature	-20 °C - +60 °C
Humidity	10% - 95% non condensing
IP protection class control cabinet	min. IP44

Figure 10:PhyGATE Tauri-S Technical Data

XI. Powering the Board:

The phyGATE Tauri-S has a 2 pin Phoenix Contact MINI COMBICON power connector (counterpart Phoenix Contact FMC 1,5/ 2-STF-3,81). The permissible input voltage is 12 VDC to 36 VDC. A 24 VDC adapter with a minimum current rating of 1A is recommended to supply the board.

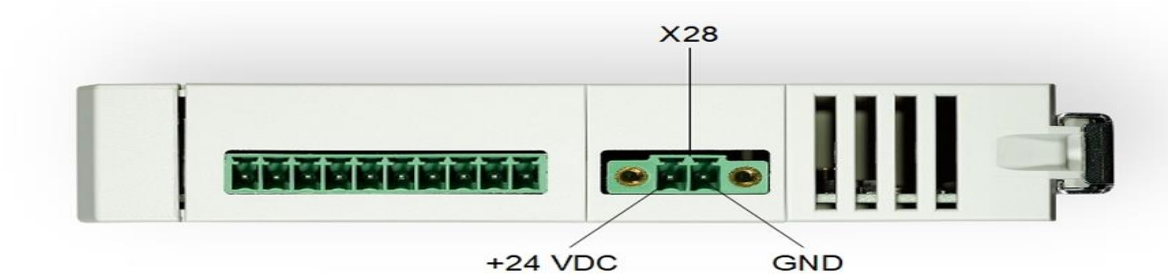


Figure 11:Power Connector Location

- After turning the power supply on, the green LED D9 in the device front will light up and the boards will start booting.

XII. -SD card:

An SD card, short for secure digital card, is a type of removable storage device used to store and transfer digital data.

If we are using your own SD Card, we 'll have to download the prebuilt image file and burn it to the SD card.

Figure 12:SD card Device

XIII. -Ethernet Cable (LAN):

Ethernet is a network protocol primarily used with RJ45 LAN connections to link multiple devices within a local network, enabling them to exchange information.

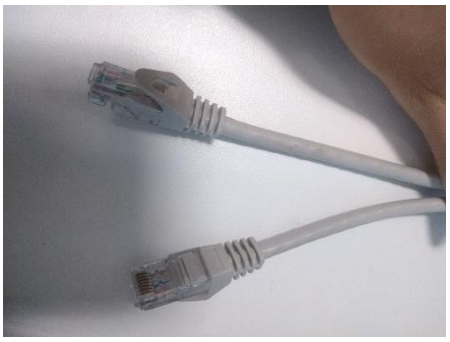


Figure 13:Ethernet cable

XIV. Yocto project workflow :

The Yocto workflow involves several key steps, including:

1. Setting Up the Build Environment:

This involves configuring the environment and dependencies required for the project.

The Yocto Project source code for your project is hosted in a Git repository on the Azure DevOps platform, specifically within the "elco" repository. To interact with this repository, Git is used as the version control system, allowing you to clone, pull, push, and manage changes to the project. The SSH (Secure Shell) protocol is employed for secure communication between your local environment and the Azure DevOps server. This protocol ensures that all data transferred, such as code or configuration files, is encrypted and protected, making the interaction secure and efficient when collaborating on the Yocto project.

Figure 14:Git LOGO

Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

2. Managing Layers:

Yocto's layered architecture allows for modular builds, making it easy to add or remove components.

3. Building the Image:

Using Bitbake, Yocto compiles and assembles the desired Linux distribution for the target hardware which are shown in the following figure.

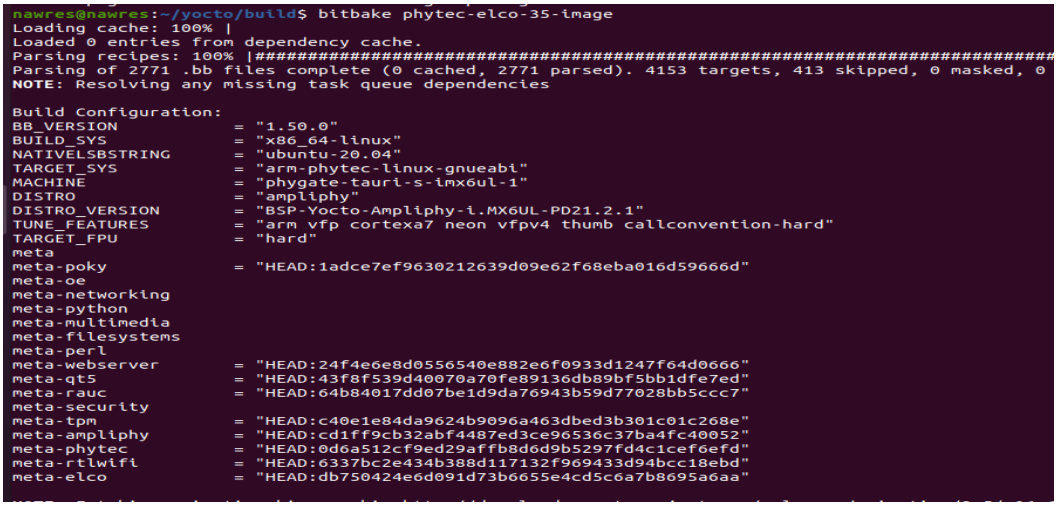


Figure 15:Build image linux

-All images generated by Bitbake are deployed to yocto/build/deploy/images/<machine>.so we will find those files as shown the following figure:

-the following list shows all files generated for the i.MX 6ULL SOC, phygate-tauri-imx6ul-1 machine: Barebox: barebox.bin

Barebox configuration: barebox-defconfig

Kernel: zImage

Kernel device tree file: imx6ull-phygate-tauri-nand.dtb

Kernel configuration: zImage.config

Root filesystem: phytec-headless-image-phygate-tauri-imx6ul-1.tar.gz,

phytec-headless-image-phygate-tauri-imx6ul-1. ubifs

SD card image: phytec-headless-image-phygate-tauri-imx6ul-1.sdcard

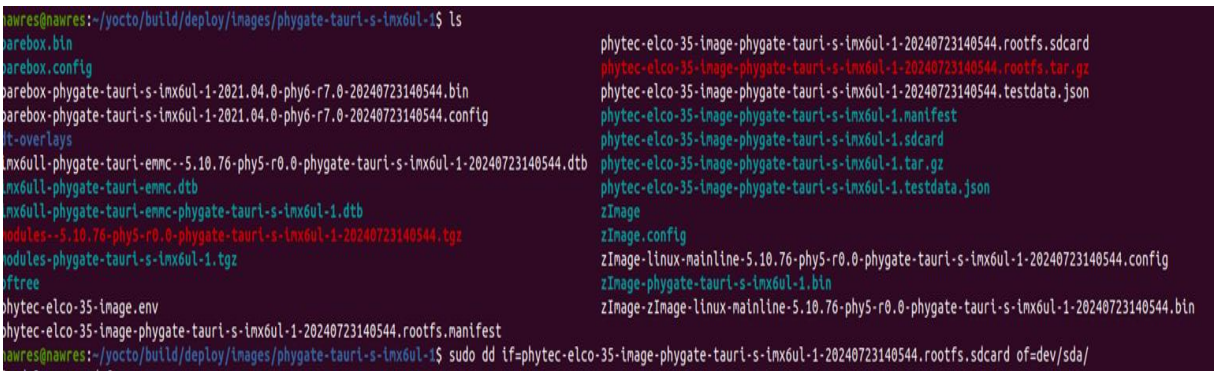


Figure 16:BSP image

4. Deploying on the Elco35 Board:

Once built, the OS image is deployed onto the Elco35(phytec) board for testing and implementation.

The default boot source for the i.MX 6UL/6ULL modules like phyCORE-i.MX 6UL or phyCORE-i.MX 6ULL is the NAND flash. The easiest way to get started with your freshly created images is by writing them to an SD card and setting the boot configuration accordingly. For information on how to set the correct boot configuration, refer to the corresponding hardware manual for your PHYTEC board,so we need to copy our image file (.sdcard) in SD card by command Linux .

-the steps:

-Unmount all partitions.

-After having unmounted all devices with an appended number (<*our_device*><*number*>), we can create your bootable SD card as shown in the flowing figure:

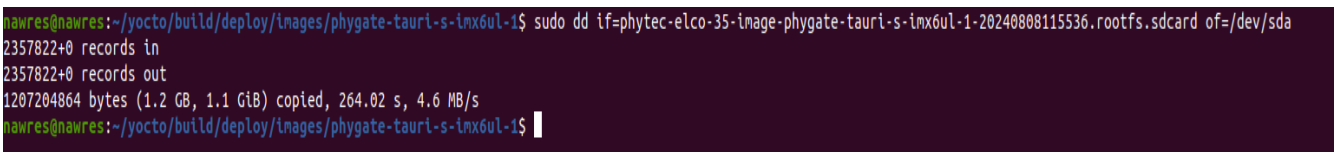


Figure 17:Copy Image in SD card

5. Boot process:

The figure below illustrates the boot process on a high level:

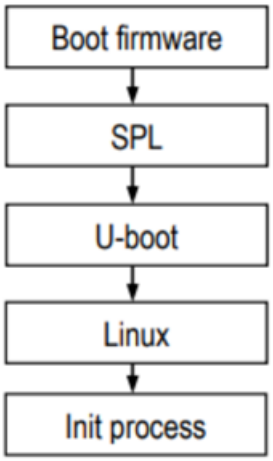


Figure 18:Boot Process

1.Boot firmware in processor starts to execute and loads SPL

2.SPL initializes SDRAM and loads U-BOOT.

3.U-BOOT initializes basic hardware, loads device tree and Linux

4.Linux activates peripherals, mounts root file system and runs init .

5.Init process starts services and main application.

This process ensures that the system is ready to perform its embedded tasks.

XV. Connecting the OS via SSH:

Once the board is done booting, we can connect to the board via Ethernet and SSH. Therefore, we need to connect an RJ45 Ethernet cable between the PhyGATE Tauri-S and our computer. We need to Make sure that the IP configuration of our computer is configured as follows:

- IP address: 192.168.3.10
- Netmask: 255.255.255.0

The use of SSH (Secure Shell) and Ethernet protocols in this context serves distinct but complementary purposes:

1. Ethernet Protocol:

Ethernet is the standard for connecting devices within a local area network (LAN). In this case, the RJ45 Ethernet cable creates a physical link between the board (phyGATE Tauri-S) and your computer. Ethernet provides a reliable, high-speed connection necessary for communication between the board and the computer. It serves as the underlying transport layer that allows network communication to take place.

2. SSH Protocol:

SSH is a network protocol that allows for secure, encrypted communication between devices. It is typically used to remotely log into a system and execute commands securely. In this setup, after the Ethernet connection is established, SSH enables secure remote access to the OS on the board. This means you can control the board, run commands, and manage configurations without physically accessing the board.

Why both are used:

-Ethernet establishes the physical network connection.

-SSH provides a secure method to access and control the board over that connection.

By combining both, you ensure that while the data travels over Ethernet, it remains encrypted and secure due to SSH.

This is critical for maintaining the integrity and confidentiality of the communication between your computer and the embedded board.

So, we can now connect via SSH. there are two ways to connect:

First way: if we are using Windows so can putty to connect with our board as shown in the following figure:

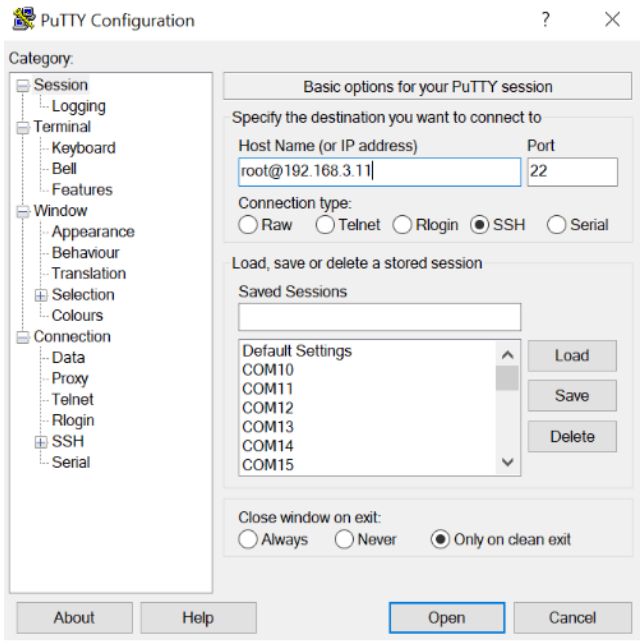


Figure 19:: PuTTY interface

Second way: if we are using Linux we need to use Linux commands as shown in the following figure and then we will get our image Linux (PHYTEC IMAGE) .

```
nawres@nawres:~/yocto/build/deploy/images/phygate-tauri-s-imx6ul-1$ ssh root@192.168.3.11
The authenticity of host '192.168.3.11 (192.168.3.11)' can't be established.
ECDSA key fingerprint is SHA256:JIBJTZqFWDGuOoFpBGTNJdKafSusAtmGLdZQRDU8rJo.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.3.11' (ECDSA) to the list of known hosts.
 ____   _   _ __   __ _____  _____   ____
|  _ \ | | | |\ \ / /|_   _||  ___| / ___|
| |_) || |_| | \ V /   | |  | |_   | |
|  __/ |  _  |  | |    | |  |  _|  | |___
|_|    |_| |_|  |_|    |_|  |_____| \____|

   _      __  __  ____    _      ___  ____    _    _ __   __
  / \     |  \/  ||  _ \  | |    |_ _||  _ \  | |  | |\ \ / /
 / _ \    | |\/| || |_) | | |     | | | |_) | | |  | | \ V /
/ ___ \   | |  | ||  __/  | |___  | | |  __/  | |_ | |  | |
/_/   \_\|_|  |_||_|     |_____||___||_|     |_| |_|  |_|

ampliPHY (Phytec Base Distribution) BSP-Yocto-Ampliphy-i.MX6UL-PD21.2.1

root@phygate-tauri-s-imx6ul-1:~# client_loop: send disconnect: Broken pipe
nawres@nawres:~/yocto/build/deploy/images/phygate-tauri-s-imx6ul-1$ cd
```

Figure 20:PHYTEC image

In this case, our environment is ready to add our web servers and to test our project.

XVI. Conclusion:

In this chapter, we have walked through the essential steps of building and deploying a customized Linux operating system for the Elco35 board using the Yocto Project. Starting with setting up a dual-boot environment for development, we explored the tools and workflow needed to create an optimized embedded system.

The chapter detailed the process of configuring and building a tailored Linux image, successfully deploying it to the Elco35 board, and establishing a secure connection via SSH for further project testing.

Chapter 3: Realization of the project

I. Introduction:

This chapter describes the practical implementation of our project, focusing on the integration and configuration of essential components. We will examine the Extensible SDK ,the Dotnet SDK architecture, selecting the ARM32 version compatible with our Elco35 board, and the integration of Systemd for automatic service management, and the setup of Nginx as a reverse proxy to enhance security and traffic management . Finally, we will discuss the compilation and testing of our .NET web application.

II. Extensible SDK (eSDK):

1. Definition of the Extensible SDK:

The Extensible SDK (eSDK) is a flexible and comprehensive version of the software development kit (SDK) within the Yocto Project. It allows developers to integrate, develop, and test applications or libraries independently, without having to rebuild the entire Yocto system. Unlike a standard SDK, the eSDK offers the ability to manipulate Yocto layers and recipes, add new recipes, and rebuild specific packages or images within the project.

2. Why use the Extensible SDK and What is in It?

The extensible SDK provides a cross-development toolchain, and libraries tailored to the contents of a specific image. You would use the Extensible SDK if you want a toolchain experience supplemented with the powerful set of devtool commands tailored for the Yocto Project environment.

The installed extensible SDK consists of several files and directories. Basically, it contains an SDK environment setup script, some configuration files, an internal build system, and the devtool functionality.

devtool subcommands provide entry-points into development:

- ***devtool add***: Assists in adding new software to be built.
- ***devtool modify***: Sets up an environment to enable you to modify the source of an existing component.
- ***devtool ide-sdk***: Generates a configuration for an IDE.

- ***devtool upgrade***: Updates an existing recipe so that you can build it for an updated set of source files.

3. Commands associated with the eSDK:

devtool: A powerful tool for managing recipes in the eSDK, it allows for the rapid development, testing, and deployment of applications or libraries outside the full Yocto build process.

bitbake -c populate_sdk_ext <image>: Used to generate the Extensible SDK for a specific Yocto image.

4. Advantages:

Reduced development time: Allows developers to work without having to generate a complete image for every change.

Isolation: Separates application development from the rest of the system, making the process more modular and easier to maintain.

Modularity: Easily add new features, layers, or recipes to the SDK without affecting the main build.

III.Dotnet web server:

1. What is Dotnet SDK:

The .NET SDK is a set of libraries and tools that developers use to create .NET applications and libraries. It contains the following components that are used to build and run applications.

Figure 21:Dotnet LOGO

2. Explanation of .NET SDK Architectures and Project Choice:

The .NET SDK is available for multiple hardware architectures, allowing .NET applications to be deployed on a variety of platforms.

These architectures include variants such as x86, x64, ARM32, and ARM64. Each architecture is optimized to run on specific processors, providing suitable compatibility and performance.

.NET SDK Architectures:

x86 (32-bit) and x64 (64-bit): These architectures are commonly used on desktop PCs and servers. They are designed for Intel and AMD processors, which are based on x86/x64 architectures.

ARM32: This architecture is designed for 32-bit ARM processors, often used in embedded systems and mobile devices.

ARM64: Intended for 64-bit ARM processors, this architecture is common in modern devices such as certain smartphones and tablets.

SDK 8.0.201

SE	Installateurs	Fichiers binaires
Linux	Instructions du gestionnaire de paquets	Arm32 | Arm32 Alpine | Arm64 | Arm64 Alpine | x64 | x64 Alpine
macOS	Arm64 | x64	Arm64 | x64
Windows	Arm64 | x64 | x86 | instructions winget	Arm64 | x64 | x86
Tous	dotnet-install scripts	

Runtimes inclus
8.0.2runtime .NET
Exécution ASP.NET Core 8.0.2
Exécution de bureau .NET 8.0.2

Prise en charge linguistique
C# 12.0
F# 8.0
Visual Basic 16.9

Figure 22:Dotnet Architecture

3. Project Choice:

For our project, we require the .NET SDK for the ARM32 architecture. The electronic board we are using, the Elco35, is equipped with an NXP i.MX 6ULL Cortex-A7 processor.

This processor is based on the ARM32 architecture, which means that to ensure compatibility and optimal performance, we need to use a version of the .NET SDK that is suited for this architecture.

We should include both of these links in our recipe (as a source in our recipe Dotnet), as we want to install it by the recipe not manually.

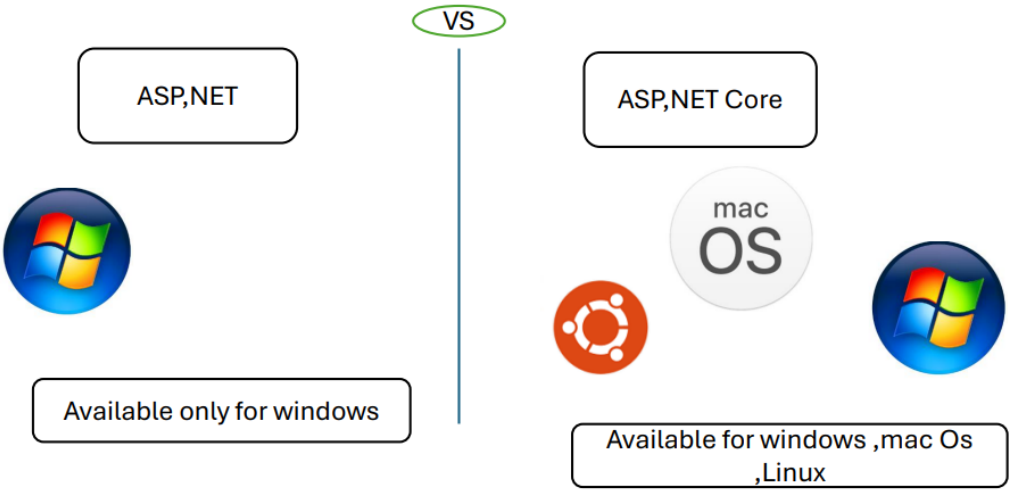
Figure 23: .NET SDK Download Links

Runtime: ASP.NET CORE 8.0.2:

We are using ASP.NET Core 8.0.2 as the runtime for this project, which provides the essential environment for running .NET web applications.

4. The difference between ASP.NET CORE and ASP.NET:

ASP.NNET core is an open-source version of ASP.NET which runs on windows ,Linux an mac OS ,the first version of ASP.net core was released in 2016 to provide some more features than ASP.NET.



The result that we found after building our recipe for the Dotnet Web -server:

```
awres@nawres:~/yocto/build/tmp/work/cortexa7t2hf-neon-vfpv4-phytec-linux-gnueabi/webserver/1.0-r0/webserver-1.0$ ls
dotnet  host  LICENSE.txt  packs  sdk  sdk-manifests  shared  templates  ThirdPartyNotices.txt
awres@nawres:~/yocto/build/tmp/work/cortexa7t2hf-neon-vfpv4-phytec-linux-gnueabi/webserver/1.0-r0/webserver-1.0$ ls -l
otal 168
rwxr-xr-x 1 nawres nawres 44844 Nov 30  2023 dotnet
rwxr-xr-x 3 nawres nawres  4096 Nov 30  2023 host
rw-r--r-- 1 nawres nawres  1116 Nov 30  2023 LICENSE.txt
rwxr-xr-x 6 nawres nawres  4096 Nov 30  2023 packs
rwxr-xr-x 3 nawres nawres  4096 Nov 30  2023 sdk
rwxr-xr-x 3 nawres nawres  4096 Nov 30  2023 sdk-manifests
rwxr-xr-x 4 nawres nawres  4096 Nov 30  2023 shared
rwxr-xr-x 3 nawres nawres  4096 Nov 30  2023 templates
rw-r--r-- 1 nawres nawres 94355 Nov 30  2023 ThirdPartyNotices.txt
awres@nawres:~/yocto/build/tmp/work/cortexa7t2hf-neon-vfpv4-phytec-linux-gnueabi/webserver/1.0-r0/webserver-1.0$
```

Figure 24:The Dotnet web server

IV. Systemd Yocto :

Adding a Systemd service file for the Dotnet web server in Yocto allows the web server to start automatically on boot, making it more reliable and easier to manage. This service file defines how and when the web server should be started, stopped, or restarted. It ensures that the web server can recover automatically if it crashes and allows administrators to control it with simple commands.

Using Systemd also enhances security by running the server with restricted permissions, isolating it from critical system processes. Overall, the integration of Systemd streamlines the deployment and improves the robustness of the Dotnet web server in an embedded environment.

```
1 [Unit]
2 Description=Continuos webserver Dotnet service
3
4 [Service]
5 ExecStart=/usr/bin/webserver-1.0 /var/www/app/Elco35.WebConfigurator.dll
6 WorkingDirectory=/var/www/app
7 Restart=on-failure
8 RestartSec=10
9 KillSignal=SIGINT
10 User=root
11 Environment=ASPNETCORE_ENVIRONMENT=Production
12
13 [Install]
14 WantedBy=multi-user.target
```

Figure 25:SystemD service file

We need to add it by the recipe (file. bbappend).

V. Nginx web server :

Nginx is a high-performance web server known for its stability, low resource consumption, and ability to serve static content efficiently. It is widely used to handle a large number of concurrent connections, making it an excellent choice for web hosting, streaming, and content delivery.

Figure 26:Nginx web server Logo

We have different types of Nginx configurations:

Feature/Use Case	Forward Proxy	Reverse Proxy
Client Visibility	The server sees the proxy, not the client	The client sees the proxy, not the server
Usage Direction	Client-side (outbound requests)	Server-side (inbound requests)
Use Cases	Anonymity, caching, access control	Load balancing, security, caching, SSL termination
Examples	Accessing blocked websites, hiding IP	Protecting backend servers, distributing traffic

Figure 27:Nginx proxy Types

So, we need to use Nginx as a reverse proxy with the provided configuration:

- The IP address and port of the .NET server are not exposed to external clients.
- Clients interact solely with Nginx.
- Nginx handles communication between clients and the .NET server in a secure and transparent manner.

This setup helps protect the backend infrastructure and centralizes request management and security.

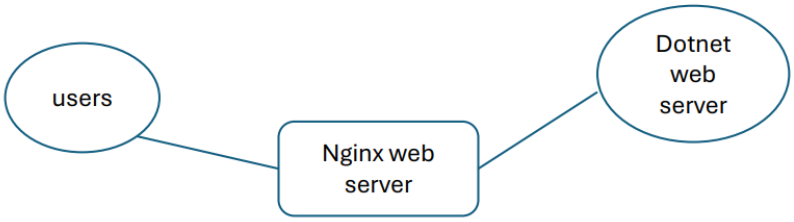


Figure 28:Nginx web server architecture

VI. Compiling the .NET Web Application:

Before testing our web servers, we need to compile our .NET web application.

There are two ways to do this:

The first way is to compile it on Windows and then simply add the compiled files.

The second way is to create a Yocto recipe to compile our web application using Dotnet (X86-64 Architecture our computer installed manually),we will get those fils(.dll/**Dynamic Link Library)** of our application in our image Linux as shown in the following figure.

```
root@phygate-tauri-s-imx6ul-1:~# cd /var/www/app/
root@phygate-tauri-s-imx6ul-1:app# ls -l
total 584
drwxr-xr-x 6 root root   4096 Aug 15 09:46 wwwroot/
-rw-r--r-- 1 root root  22016 Aug 15 09:46 Elco35.ConfiguratorCore.dll
-rw-r--r-- 1 root root  27280 Aug 15 09:46 Elco35.ConfiguratorCore.pdb
-rw-r--r-- 1 root root   1410 Aug 15 09:46 Elco35.IotGatewayConfig.deps.json
-rw-r--r-- 1 root root  18432 Aug 15 09:46 Elco35.IotGatewayConfig.dll
-rw-r--r-- 1 root root  26448 Aug 15 09:46 Elco35.IotGatewayConfig.pdb
-rwxr-xr-x 1 root root  48044 Aug 15 09:46 Elco35.WebConfigurator*
-rw-r--r-- 1 root root   1407 Aug 15 09:46 Elco35.WebConfigurator.deps.json
-rwxr-xr-x 1 root root 159232 Aug 15 09:46 Elco35.WebConfigurator.dll*
-rw-r--r-- 1 root root  68420 Aug 15 09:46 Elco35.WebConfigurator.pdb
-rw-r--r-- 1 root root    488 Aug 15 09:46 Elco35.WebConfigurator.runtimeconfig.json
-rw-r--r-- 1 root root 137216 Aug 15 09:46 Tomlyn.dll
-rwxr-xr-x 1 root root    506 Mar  9  2018 WebApplicationLinux.deps.json*
-rwxr-xr-x 1 root root  43008 Mar  9  2018 WebApplicationLinux.dll*
-rwxr-xr-x 1 root root    488 Mar  9  2018 WebApplicationLinux.runtimeconfig.json*
-rw-r--r-- 1 root root    127 Aug 15 09:46 appsettings.Development.json
-rw-r--r-- 1 root root    151 Aug 15 09:46 appsettings.json
```

Figure 29:compiled Fils for the Web-application

1. The web application we have is for configuring our board (Elco35).

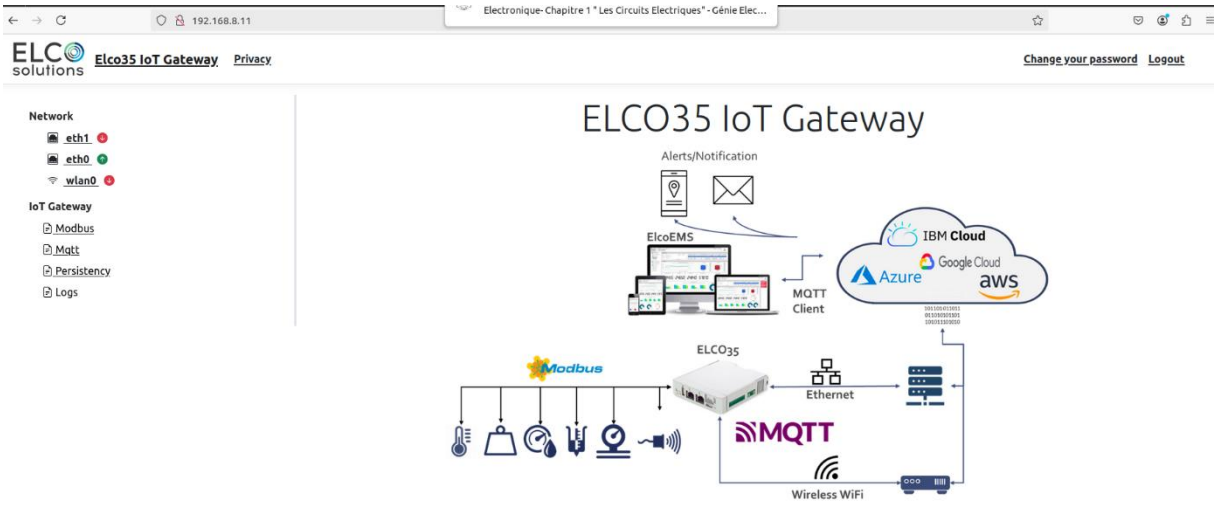


Figure 30:Web-Application

VII. Testing the Application with and without Nginx:

1. Direct Access to the .NET Server:

When we access the .NET server directly, we use the local address and the specific port on which the application is hosted.

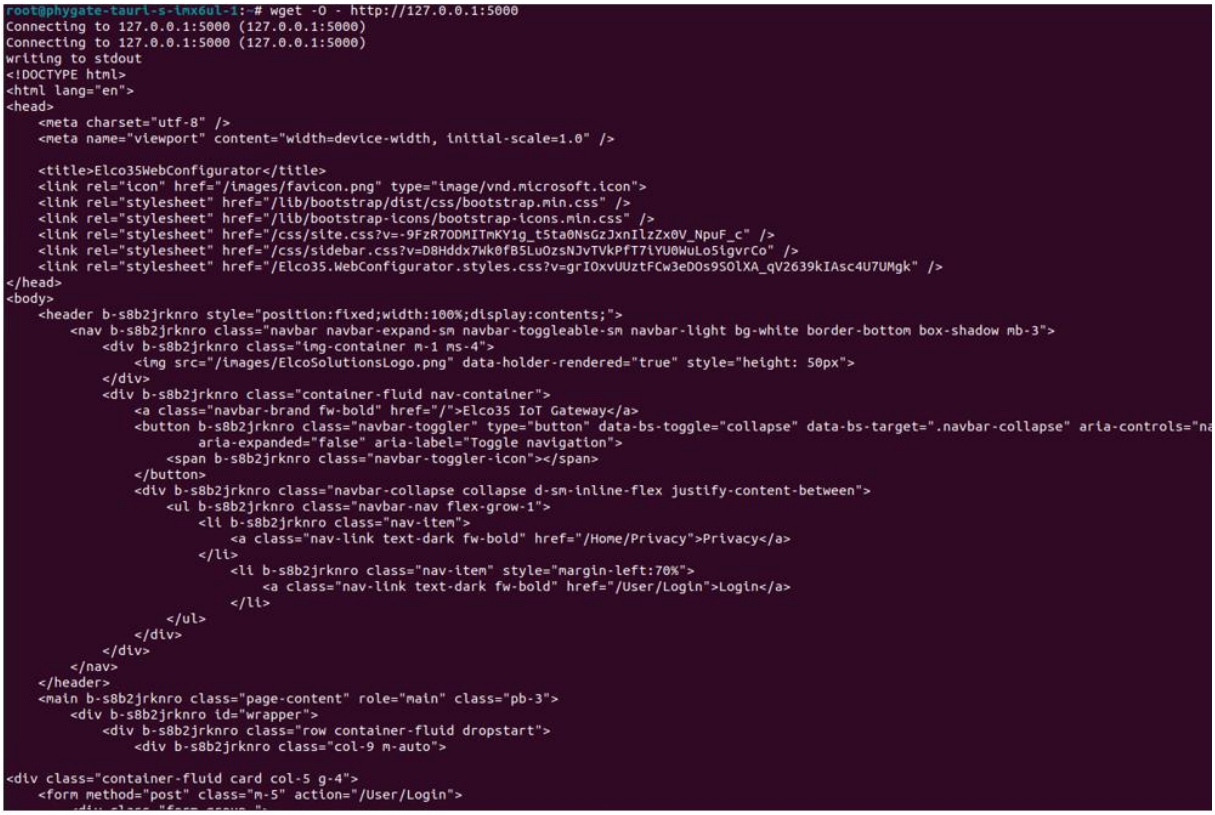


Figure 31:Direct Access Test Web-application

So, we receive a direct response from the .NET server, showing the “THE code HTML of our web-application" This means that the .NET server is up and running and can respond to HTTP requests on port 8080 without any intermediary.

This type of access is useful for basic connectivity tests and to check the application's functionality. However, it directly exposes the .NET server to the network without any layer of protection.

2. Access via Nginx as a Reverse Proxy:

we configured Nginx to act as a reverse proxy. This means that Nginx receives client requests and forwards them to the .NET server while hiding the server's actual address and port.

```
root@phygate-tauri-s-imx6ul-1:~# wget -O - http://localhost
Connecting to localhost (127.0.0.1:80)
Connecting to localhost (127.0.0.1:80)
writing to stdout
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Elco35WebConfigurator</title>
    <link rel="icon" href="/images/favicon.png" type="image/vnd.microsoft.icon">
    <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="/lib/bootstrap-icons/bootstrap-icons.min.css" />
    <link rel="stylesheet" href="/css/site.css?v=-9FzR7ODMITmKY1g_t5ta0NsGzJxnIlzZx0V_NpuF_c" />
    <link rel="stylesheet" href="/css/sidebar.css?v=D8Hddx7Wk0fB5LuOzsNJvTVkPfT7iYU0WuLo5igvrCo" />
    <link rel="stylesheet" href="/Elco35.WebConfigurator.styles.css?v=grIOxvUUztFCw3eDOs9SOlXA_qV2639kIAsc4U7UMgk" />
</head>
<body>
    <header b-s8b2jrknro style="position:fixed;width:100%;display:contents;">
        <nav b-s8b2jrknro class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div b-s8b2jrknro class="img-container m-1 ms-4">
                <img src="/images/ElcoSolutionsLogo.png" data-holder-rendered="true" style="height: 50px">
            </div>
            <div b-s8b2jrknro class="container-fluid nav-container">
                <a class="navbar-brand fw-bold" href="/">Elco35 IoT Gateway</a>
                <button b-s8b2jrknro class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-cont
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span b-s8b2jrknro class="navbar-toggler-icon"></span>
                </button>
                <div b-s8b2jrknro class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
                    <ul b-s8b2jrknro class="navbar-nav flex-grow-1">
                        <li b-s8b2jrknro class="nav-item">
                            <a class="nav-link text-dark fw-bold" href="/Home/Privacy">Privacy</a>
                        </li>
                            <li b-s8b2jrknro class="nav-item" style="margin-left:70%">
                                <a class="nav-link text-dark fw-bold" href="/User/Login">Login</a>
                            </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <main b-s8b2jrknro class="page-content" role="main" class="pb-3">
        <div b-s8b2jrknro id="wrapper">
            <div b-s8b2jrknro class="row container-fluid dropstart">
                <div b-s8b2jrknro class="col-9 m-auto">

<div class="container-fluid card col-5 g-4">
    <form method="post" class="m-5" action="/User/Login">
```

Figure 32:Access via nginx

we also see "the HTML code of web-application " but this time the request passes through Nginx. Nginx manages the incoming connections, forwards the traffic to the .NET server, and protects the backend infrastructure.

The advantage of this setup is that the .NET server is not directly exposed. Nginx can filter requests and handle other aspects such as caching, load balancing, and security.

VIII. Conclusion:

In conclusion, the implementation chapter was a crucial step in our project, highlighting the addition of web servers ,Extensible SDK(eSDK) and compiling the Web-application in Yocto. These improvements have generated significant added value, reinforcing our belief that this project is scalable and adaptable to meet future needs.

General Conclusion:

This project successfully addressed the challenges of integrating a web-based management interface into an embedded system, specifically the Elco35 (PhyGATE Tauri-S) board. By incorporating a Dotnet web server using the Yocto Project framework, the solution enables remote control and monitoring of the system, offering a more efficient and user-friendly approach to managing embedded environments.

Key objectives such as creating a flexible build environment, automating the server start-up process, and optimizing communication through Nginx as a reverse proxy were accomplished.

This project not only demonstrates the feasibility of integrating web applications in resource-constrained environments but also provides a robust foundation for future enhancements, including expanding functionalities and improving security.

Overall, the solution contributes to modernizing embedded system management, promoting scalability, and ensuring ease of use for industry applications.

Bibliography:

[1]https://www.phytec.de/cdocuments/?doc=SIBaHw#HardwareandBSPManualphyGATETauriSi-MX6ULULLKitL1012e-A3-ConnectingtheOSviaSSH (19/09/2024)

[2]file:///home/nawres/Downloads/Hardware%20and%20BSP%20Manual%20-%20phyGATE-Tauri-S%20i.MX%206UL_ULL%20Kit%20(L-1012e.A3)-v3-20240711_105959.pdf (20/.09/2024)

[3] https://dotnet.microsoft.com/en-us/download/dotnet/8.0 (30/08/2024)

[4] https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-publish (01/09/2024)

[5] https://docs.yoctoproject.org/ref-manual/variables.html (17/09/2024)

[6] https://wiki.koansoftware.com/index.php/Add_a_systemd_service_file_into_a_Yocto_image (02/09/2024)

[7] https://stackoverflow.com/(27/08/2024)

[8]https://www.youtube.com/watch?v=5fj05BWryhM&list=PLwqS94HTEwpQmgL1UsSwNk_2tQdzq3eVJ (29/08/2024)

[9] https://www.elco-solutions.de/fr/ (15/09/2024)

[10] https://www.microsoft.com/net/core (02/08/2024)

[11] https://wiki.koansoftware.com/index.php/Add_a_systemd_service_file_into_a_Yocto_image (09/08/2024)

[12] https://nginx.org/en/(18/08/2024)

Abstract:

The project aims to integrate a Dotnet web server on the Elco35 (PhyGATE Tauri-S) embedded board using the Yocto framework. This integration enables the creation of a remote management interface to monitor and control the system without the need for direct intervention. The main steps of the project include configuring the Yocto environment, writing a recipe for the web server, adding a systemd service file to automate the server start-up, and using Nginx as a reverse proxy to improve performance and security. This project provides a modern solution for managing embedded systems and lays the groundwork for future expansions and improvements in security and functionality.

Key-words: Yocto project ,Embedded Linux ,Recipe ,Elco35 board ,Systemd ,Dotnet ,Nginx ,Extebsible SDK ,Web-Application .

Résumé :

Le projet vise à intégrer un serveur web Dotnet sur la carte embarquée Elco35 (PhyGATE Tauri-S) en utilisant le framework Yocto. Cette intégration permet de créer une interface de gestion à distance pour surveiller et contrôler le système sans nécessiter d'intervention directe. Les principales étapes du projet incluent la configuration de l'environnement Yocto, l'écriture d'une recette pour le serveur web, l'ajout d'un fichier de service systemd pour automatiser le démarrage du serveur, et l'utilisation de Nginx en tant que proxy inverse pour améliorer les performances et la sécurité. Ce projet offre une solution moderne pour la gestion des systèmes embarqués et prépare le terrain pour des extensions futures et des améliorations en matière de sécurité et de fonctionnalités.

Mots-clés : Projet Yocto ,Linux embarqué ,recette ,Systemd ,Dotnet,Nginx ,Applications-Web,Extensible SDK .