INDUSTRIAL PROJECT REPORT

Internship project report on

MICROROS SUPPORT ON DEVELOPMENT BOARDS

submitted in partial fulfilment of the
Requirements for the award of

Bachelor of Engineering
in
School of Electronics and Communication Engineering

Carried out at

Bosch Global Software Technologies Private Limited

Submitted By-
Aditya krishna vamsy Mudragada
01FE18BEC006

Under the guidance of

Prof. Prabha C N
College Guide
KLE Technological University

Ravikumar Nanjaiah
Industry Guide
Bosch global software technologies

SUBMITTED TO:
School of Electronics and Communication Engineering
KLE TECHNOLOGICAL UNIVERSITY
Hubballi

DECLARATION

I hereby declare that the Industrial Project Report entitled MICROROS SUPPORT ON DEVELOPMENT BOARDS is an authentic record of my own work as requirements of Industrial Project during the period from 15-02-2022 to 31-05-2022 for the award of the Degree of Bachelor of Engineering in Electronics and Communication.**KLE Technological University, Hubballi**, under the guidance of Prof. Prabha C Nissimagoudar.

Aditya krishna vamsy Mudragada
01FE18BEC006

Date:

K.L.E SOCIETY'S
KLE Technological University,
Hubballi-580031
2020-2021

SCHOOL OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the internship project entitled MICROROS SUPPORT ON DEVELOPMENT BOARDS is a bonafide work carried out by Mr. Aditya krishna vamsy Mudragada , bearing University Seat No.01fe18bec006,in Bosch global software Technologies company, in partial fulfillment for the award for Bachelor of Engineering in Electronics and Communication in the School of Electronics and Communication Engineering of KLE Technological University, Hubballi for the academic year 2021-2022.

Prof. Prabha C N	**Dr. Nalini C. Iyer**	**Dr.N.H.Ayachit**
Guide	**Head of School**	**Registrar**

External Viva:

Name of Examiners	**Signature with date**
1.	
2.	

ACKNOWLEDGMENT

It is a genuine pleasure to express our deep sense of thanks to Ravikumar Nanjaiah , Chhavinidhi Singh for guidance and constant supervision. The project wouldn't be possible without their support and timely advice. I am highly indebted to Dr. Ashok S Shettar , Vice Chancellor of KLE Technological University, Hubli , Head of School of Electronics and communication department for giving us an opportunity to implement a solution for such a noble cause.e.I am thankful to my university guide Prof. Prabha C Nissimagoudar for support and encouragement throughout the internship

- Aditya krishna vamsy Mudragada(01fe18bec006)

COMPANY PROFILE

Figure 1: Bosch

The Bosch Group is a leading global supplier of technology and services. It employs roughly 395,000 associates worldwide (as of December 31, 2020). The company generated sales of 71.5 billion euros in 2020. Its operations are divided into four business sectors: Mobility Solutions, Industrial Technology, Consumer Goods, and Energy and Building Technology. As a leading IoT provider, Bosch offers innovative solutions for smart homes, Industry 4.0, and connected mobility. Bosch is pursuing a vision of mobility that is sustainable, safe, and exciting. It uses its expertise in sensor technology, software, and services, as well as its own IoT cloud, to offer its customers connected, cross-domain solutions from a single source. The Bosch Group's strategic objective is to facilitate connected living with products and solutions that either contain artificial intelligence (AI) or have been developed or manufactured with its help. Bosch improves quality of life worldwide with products and services that are innovative and spark enthusiasm. In short, Bosch creates technology that is "Invented for life."

In India, Bosch is a leading supplier of technology and services in the areas of Mobility Solutions, Industrial Technology, Consumer Goods, and Energy and Building Technology. Additionally, Bosch has in India the largest development center outside Germany, for endto-end engineering and technology solutions. The Bosch Group operates in India through twelve companies: Bosch Limited – the flagship company of the Bosch Group in India – Bosch Chassis Systems India Private Limited, Bosch Rexroth (India) Private Limited, Bosch Global Software Technologies, Bosch Automotive Electronics India Private Limited, Bosch Electrical Drives India Private Limited, BSH Home Appliances Private Limited, ETAS Automotive India Private Limited, Robert Bosch Automotive Steering Private Limited, Automobility Services and Solutions Private Limited, Newtech Filter India Private Limited and Mivin Engg. Technologies Private Limited. In India, Bosch set-up its manufacturing operations in 1951, which has grown over the years to include 16 manufacturing sites, and seven development and application centers. The Bosch Group in India employs over 31,500 associates and generated consolidated sales of about .19,996 crores* (2.54 billion euros) in fiscal year 2020 of which . 14,011 crores*(1.78 billion euros) are from consolidated sales to third parties. The Bosch Group in India has close to 15,650 research and development associates.

ABSTRACT

Micro-ROS is a robotic framework designed for embedded and deep-embedded robot components that have limited processing resources.Micro-ROS allows traditional robots to communicate with IoT sensors and devices, resulting in truly dispersed robotic systems based on a common foundation.

This framework can be utilised in the automotive domain for a variety of purposes, resulting in a well-established application development framework for the embedded world based on standard communication middleware. It is critical to have support for this standard on several development boards that imitate the automotive environment, where it is typical to have ECUs with varied RTOS and hardware.

Contents

List of Figures

Chapter 1

Introduction

Micro-ROS is a robotic framework designed for embedded and deep-embedded robot components that have limited processing resources.Micro-ROS allows traditional robots to communicate with IoT sensors and devices, resulting in truly dispersed robotic systems based on a common foundation.Support for this standard is crucial on several development boards that simulate the automotive environment, where ECUs with various RTOS and hardware are common.

1.1 Motivation

- Support for this standard is essential on a number of development boards that imitate the automotive environment, where ECUs with a variety of RTOS and hardware are prevalent.
- Microros extremely resource-constrained but flexible middleware that can be used on microcontrollers
- Seamless integration with ROS 2 on ECU's.

1.2 Objectives

- Understand and compare the embedded build steps with microros build steps.
- Integrate microros with RTOSes on a target board from application perspective.

1.3 Literature survey

1.3.1 About ROS

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications and is an open-source robotics middleware suite. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

ROS provides functionality for hardware abstraction, device drivers, communication between processes over multiple machines, tools for testing and visualization, and much more. The key feature of ROS is the way the software is run and the way it communicates, allowing you to design complex software without knowing how certain hardware works. ROS provides a way to connect a network of processes (nodes) with a central hub. Nodes can be run on multiple devices, and they connect to that hub in various ways.

Ros was designed opensource with an aim intending that user would be able to choose the configuration of tools and libraries which interacted with the core of ROS so that users could shift their software stacks to fit their robot and application area.

ROS1 was initially created on 2007 by Willow Garage after year of using ROS1 the team came to know what important features are missing, and what could be improved but adding the changes to ros1 makes ros1 unstable so ros2 was developed the main aim of ros2 is to fulfill the requirements related to real-time,safety,certification the main aim for developing ros2 is to make ros compatible with industrial application.

Ros was designed opensource with an aim intending that user would be able to choose the configuration of tools and libraries which interacted with the core of ROS so that users could shift their software stacks to fit their robot and application area.

1.3.2 Micro-ROS

ROS has been developed for use on microprocessors, assuming several hundred megabytes of RAM at the minimum. However, robotic systems usually are networks of one or more microprocessors with several micro-controllers. This applies to both the classic ROS and the current version named ROS2 . Microros is a robotic framework the is made for extremely resource constrained devices such as embedded devices. Micro-ROS is compatible with Robot Operating System (ROS 2.0), the de facto standard for robot application development. In turn, it puts ROS 2 onto micro-controllers.

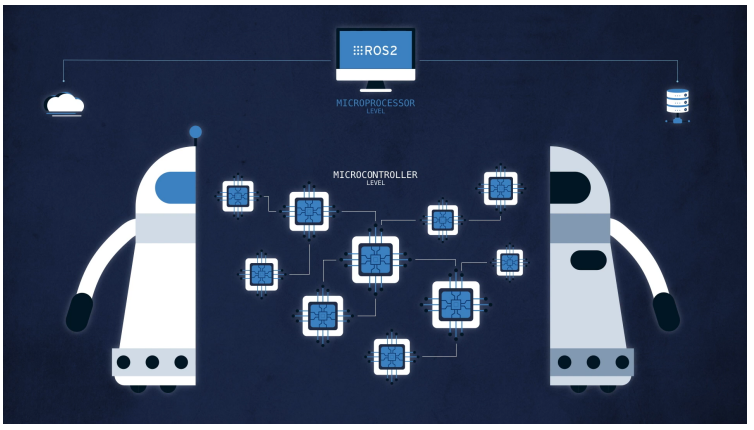

Figure 1.1: micro-ros

1.3.3 Middleware

Middleware is the software layer that sits between the operating system and the applications in a distributed system. It allows the many components of a system to communicate and share data more readily.It makes distributed system development easier by allowing software developers to concentrate on the specific goal of their applications rather than the mechanics of data transfer between apps and systems.

There are numerous standards and products for communications middleware. DDS is data-centric by design, making it suited for the Industrial Internet of Things.The majority of middleware functions by delivering data between applications and systems. Data centricity ensures that all communications contain the contextual information required by an application to comprehend the data it receives.

DDS contains security techniques for information dissemination that offer authentication, access control, secrecy, and integrity. DDS Security employs a decentralised peer-to-peer architecture that ensures security while preserving real-time performance.

DDS is the default middleware in ROS 2. It works with a variety of DDS or RTPS (the DDS wire protocol) manufacturers. eProsima's Fast DDS, RTI's Connext DDS, Eclipse Cyclone DDS, and GurumNetworks GurumDDS are currently supported. The default middleware implementation for micro-ROS' rmw layer is Micro XRCE-DDS. The DDS-XRCE protocol's goal is to give resource-constrained devices access to the DDS Global-Data-Space.

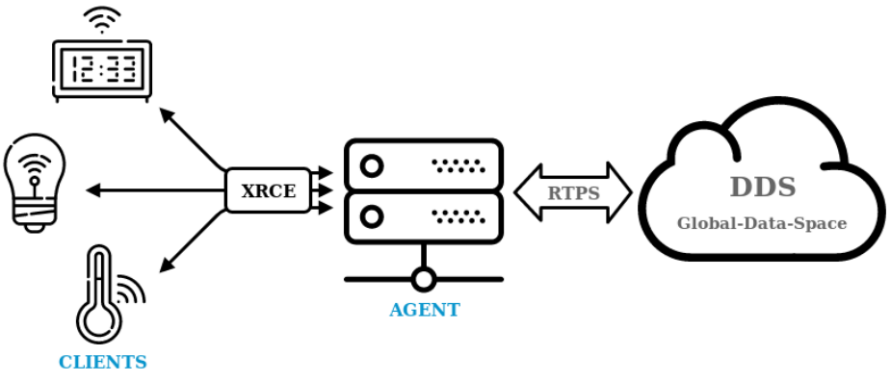

Figure 1.2: middleware

1.3.4 Architecture of ROS and Micro-ROS

The micro ros has the same architecture as ROS2 with distributed real-time system architecture . All the sensors , algorithms etc all components are called node in this distrubuted architecture . The microros applications access the ros2 features using the ROS client library. The client library is provided with interface with which it can exchange data using client-server etc approaches over ros2.

Due to its dynamic memory requirements, the ROS 2 C++ API (rclcpp) cannot be used on many microcontrollers. Instead, micro-ROS enriches the existing ROS 2 Client Support Library (rcl) by a small library rclc, which together offer a complete C API optimized for microcontrollers.

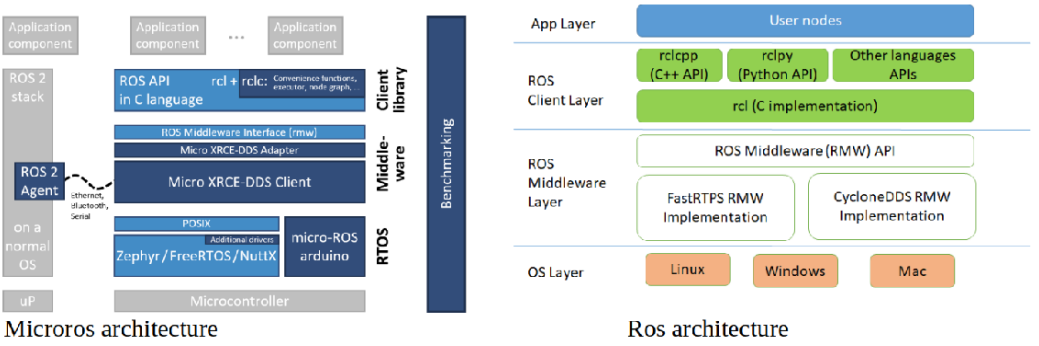

Figure 1.3: Architecture of ros and micro-ros

All these components communicate with each other using DDS middleware in the distributed environment. Microros has a middleware interface which is written in C language which procides package having middleware implementation for XRCE-DDS(rmw layer) it discribes the interface for upper layers of the microros stack and is implemented using XRCE-DDS middleware in the lower layers the middleware interface layer(rmw) helps in abstracting the information from the user.

Micro-XRCE-DDS couples with the standard DDS middleware used in ROS 2 by the micro-ROS agent. The agent takes over discovery and QoS mechanisms that are too computationally expensive for the microcontroller.

As the embedded devices are resource constrained, we need and operating system to function as a resource manager which occupies less space, fewer resources with high response time there are different RTOSes supported by microros architecture such as freertos etc. in this following way the user application accesses the ros2 features and resources using rtoses.micro-ROS requires the RTOS to provide a POSIX interface.

1.3.5 Micro-ros Supported platforms

Supported hardware platforms related information can be found here

Supported RTOS related information can be found here

Micro-ROS seeks to introduce ROS 2 to a wide range of microcontrollers, allowing the embedded world to have first-class ROS 2 entities. Micro-ROS is primarily aimed at mid-range 32-bit microcontroller families. Memory limits are usually the bare minimum for operating micro-ROS on an embedded architecture.

1.4 Problem statement

Document the software build steps of microROS and to Identify and document the steps specific to integrating RTOSes on development boards on microROS. Demonstrate the know-how gathered by integrating microROS on specific development board.

	FreeRTOS	Zephyr	Mbed
Olimex STM32-E407	UART, Network	USB, UART	-
ST B-L475E-IOT01A	-	USB, UART, Network	UART
Crazyflie 2.1	Custom Radio Link	-	-
Espressif ESP32	UART, WiFI UDP	-	-
ST Nucleo F446RE [1]	UART	-	-
ST Nucleo F446ZE [1]	UART	-	-
ST Nucleo H743ZI [1]	-	UART	-
ST Nucleo F746ZG [1]	UART	UART	-
ST Nucleo F767ZI [1]	UART	-	-

[1] *Community supported, may have lack of official support*

Figure 1.4: Supported boards

1.5 Application in Societal Context

Integration of microros onto automotive helps in bringing the robotic framework onto the vehicles and provides a simple integration of a variety of tools, including as visualisation of robot kinematics and sensor data, path planning and perception algorithms, and low-level drivers for commonly used sensors.The level of complexity in software development required in robotics projects can be considerably decreased with the help of ROS. ROS has an integrated framework and toolsets for robotics development, which speeds up software creation and aids in redistribution there by leading to new and faster innovations.

Chapter 2

PREREUISTES/REQUIREMENTS

2.1 Build steps of embedded system

The host computer used to develop the code and the target embedded platform have different architecture, so embedded systems have different build processes.

Build steps of embedded system:

The build process is to compile the c code to machine understandable code for the target system. An Embedded system would also use tools such as a Compiler, Linker, Locater and Debugger to perform the entire build process. When build tools run on the same system as the program generate on the same system there are a lot of assumptions about the system but in embedded system case this can't be done as target system is not present in the host system, so we need to provide different information to the host system about to the target system.

The different tools used for the build process in embedded systems:

- Toolchain: It is the set of tools that compiles source code into executable that can run on your target device, and includes a compiler, a linker, and run-time libraries. All these tools are required for a specific architecture of an embedded system to generate the code from a host pc. In this way the code generation is automated in the embedded system.
- Target system: The target system may be having an architecture either 8-bit or 16-bit ... which is basically the bit size of the microcontroller or processor which is supported by the target system where we need to provide information regarding the memory interfaces which are available in the target system.
- Cross compilers: Compilation of the code in the language mostly in C which is platform independent it uses the configuration files information. It takes all source files and compiles it to the object file. These object files will be linked together to generate executable code.
- Linker script: During linking there will be another automation script called linker script which has information of all the physical memory addresses where the program will be relocated in the target system.

- Flashing to the target hardware: The result of the final step is a file called executable binary image which is ready to be run on the embedded system. After generation of the executable file, it needs to be downloaded to the target board to make it run.
- The need for such a procedure is because the host pc we are working on may be using windows which has a separate architecture like intel etc but the embedded system has a different architecture so, the host system cannot directly run the program on the target architecture.

2.2 Build steps of micro

We start with setting up a workspace with a ready-to-use micro-ros build system. The build system oversees downloading the necessary cross-compilation tools and building the apps for the required platforms.

Workflow of the build procedure: We begin by setting up a workspace with a micro-ros build system that is ready to use. The build system is in charge of getting the necessary cross-compilation tools and creating programs for the platforms that are required.

Workflow of the build procedure:

- Create step: This step is responsible for downloading the relevant code repositories and cross-compilation toolchains for the hardware platform in question. It will also download a library of ready-to-use micro-ROS apps from these sources.

  The following steps are required for implementing the step:

  - Retrieving RTOS and Platform
  - Checking if firmware exists and checking folders.
  - Setting common environment and Check generic build.
  - Creating development directory mcu directory .
  - build the development workspace.
  - Installing toolchain and Import repos related to the board.

  Configure step: The user can choose which app will be cross-compiled by the toolchain at this stage. In this phase, you'll also choose transport, the agent's IP address/port (for UDP transport), and the device ID (for serial connections). This configuration stage has the following options:

  - —transport or -t: udp, serial, or any other transport designation unique to the hardware
  - In a serial-like transport, —dev or -d is the agent string descriptor.
  - –ip or -i: agent IP in a network-like transport
  - –port or -p: agent port in a network-like transport

  Build step: This is where cross-compilation and platform-specific binaries are generated. To build the firmware, you'll need to do the following.:

- Parse pli arguments which provide options during build Check if the firmware exists
- Clean paths to clean the environment setup
- Building the specific firmware folder.

Flash step:The binaries generated in the preceding step are flashed onto the hardware platform memory to allow the micro-ROS application to run.Steps involved are as follows.

- Checking if firmware exists.
- Flash specific firmware folder if needed
- Flashing

2.3 Understanding RTOS integration on target hardware

2.3.1 Zephyr

Procedure:

Understanding the target hardware The examination of the board to be worked on begins with the schematic of the board, which aids in making conclusions regarding gpio pins, on-board buttons, and sensors.

Main apis needed for integration Using api's to access os and board-related functions In order to take advantage of all of Zephyr RTOS's features, we must be aware of the api's available.

- Device driver API: The Zephyr kernel supports a variety of device drivers. Whether a driver is available depends on the board and the driver.
- Interrupt controller: This device driver is used by the kernel's interrupt management subsystem.
- Timer: This device driver is used by the kernel's system clock and hardware clock subsystem.
- Serial communication: This device driver is used by the kernel's system console subsystem.
- Entropy: This device driver provides a source of entropy numbers for the random number generator subsystem.

for more information related to the api's refer to

Creating the application

The following are the components of a Zephyr application directory:

- The build settings configuration information this tells west (which is essentially a cmake build system) where to look for the files it needs to build the firmware. It's also used for debugging, emulation, and other capabilities in more advanced applications.
- Information about the kernel's setup It instructs Zephyr whether to incorporate specific capabilities in your application code for most projects - whether GPIO, PWM, or a serial monitor are utilized, it must first enable them in this file.
- The magic happens in the bespoke application code! It's a good idea to place all of the custom source code in a src/ directory like this to keep it separate from the settings files.
- In some circumstances, the board you want to use, or even the individual MCU chip you wish to use, may not be supported in Zephyr yet. Create your own definitions in special projects if this is the case.

Toolchains used

Gcc , NXP MCUXpresso SDK , Renode , Synopsys Designware , Nordic nrfx

For detailed explanation refer click here

2.4 Freertos

About

Freertos provides Board support packages to mainly Atmel, stm, and NXP in order to integrate RTOS to a given board.

STM32Cube is free embedded software from ST that includes all of the drivers and middleware components needed to get STM32 ARM Cortex-M microcontrollers up and running quickly. FreeRTOS is included with STM32Cube, however it is not required.

Main apis needed for integration A FreeRTOS application will start and run as a non-RTOS application until the vTaskStartScheduler() API is used, which starts the RTOS scheduler. vTaskStartScheduler() is commonly called from an application's main() function. The RTOS regulates the execution sequence after vTaskStartScheduler() is called.
FreeRTOS is distributed as standard C source files that are built alongside the rest of the project's C files.

The following items must be included in the project at a minimum in the source to handle things related to tasks , list , queue , compiler architecture , memory management.

Configuration

A file called FreeRTOSConfig.h is also required for every project. The RTOS kernel is tailored to the application using FreeRTOSConfig.h. It should be in an application directory, not one of the RTOS kernel source code directories, because it is exclusive to the application, not RTOS.

Toolchains used

A toolchain is the set of tools that compiles source code into executables that can run on target device, and includes a compiler, a linker, and run-time libraries. Toolchains supported for ST Hardware are : IAR, Atollic TrueStudio, GCC, Keil, Rowley CrossWorks. For more details refer click here

Chapter 3

Integrate development boards with microROS on Freertos and zephyr RTOSes

3.1 Zephyr

Create step

- Retrieving RTOS and Platform: Validation occurs by examining the directories, depending on the amount of arguments given by the user and the combination of rtos and board. In comparison to freertos, Zephyr takes a more broad approach to the platform.
- Checking if firmware exists: Checking if there is any firmware setup already established is part of the validation stage to ensure that the firmware folder is not overwritten.
- Checking folders: Following the validation process, the files relevant to rtos's unique platform are reviewed to see if they are there in order to create firmware for a certain platform and rtos combination.
- Setting common environment: Set up a common environment for DDS (middleware) for microcontroller applications that require access to the publisher/subscriber architecture for information sharing.
- Check generic build: There is a generic build implementation for zephyr that provides a common approach for all boards. Zephyr's SDK handles the creation of the application from its supported boards.
- Creating development directory and mcu directory :Make a dev ws directory with ros2 packages and apps, as well as a mcu directory (having micro ros related repos and ros2 packages)
- Build the development workspace: If there is an existing build environment, clean it and construct the workspace using the colcon command, then install the setup.
- Install dependencies for specific platform: Install platform-specific dependencies among all platform dependencies present in a given ros distribution, such as "foxy,"

while ignoring a few keys. Creating specialized firmware folders with all relevant data.

- installing toolchain: GNU Arm Embedded Toolchain is a ready-to-use, open-source collection of tools for C, C++, and assembly development. Download the arm compiler. The toolchain differs depending on the architecture; if using cmake, the version must be higher than 3.13.1. Depending on whether the platform is a host or a board, the features change.
- Initializing the Zephyr project: The Zephyr project provides the west command line tool, which is a Swiss army knife command line tool. West uses a single file called the west manifest file, or manifest for short, to handle many Git repositories under a single directory. Set up this directory with west init, then use west update to fetch and/or update the repositories listed in the manifest.
- Installing Zephyr SDK: A toolchain consists of a compiler, an assembler, a linker, and other programs needed to create applications. Each of Zephyr's supported architectures has its own toolchain in the Zephyr Software Development Kit (SDK). Additional host tools, such as custom QEMU and OpenOCD builds, are also included.
- Import repos related to the board: Import the freertos-related apps and configure them to run on target boards.
- Installing the ros dependencies for the board in the newly established mcu workspace.

Configure the firmware

- Checking if firmware exists and configure script exist: Part of the validation process is to see if the firmware and configuration scripts are present before moving on to the next step.
- Parsing micro-ROS arguments: Checks for the availability of the provided app and handles the configuration relevant to how data is transported:

  The options available for this configuration is:

  - transport or -t: udp, serial or any hardware-specific transport label
  - dev or -d: agent string descriptor in a serial-like transport
  - ip or -i: agent IP in a network-like transport
  - port or -p: agent port in a network-like transport

  Configure specific firmware folder if needed: In this phase, based on the transport type, the specific firmware folder related to a combination of RTOS and board is configured, and the generic implementation of DDS middleware configuration is also set.

  Set the DDS and Transport: DDS-related parameters are taken care of, and transport is determined based on the information provided by the user.

Building the firmware

- Parse cli arguments: The arguments are related to fast build, verbose mode etc .Command line arguments are simple arguments that are stated on the system's command line following the program's name, and the values of these arguments are passed on to the program during program execution.
- Checking if firmware exists: A stage in the validation process that checks if the firmware is present.
- Clean paths : Setup a clean atmosphere Before creating a specific firmware folder, it is preferable to source it rather than run it.
- Building the specific firmware folder: By configuring the utilities, cleaning the microros build, generating a microros stack, and lastly building the specific firmware, you can create the firmware folder that is supported by RTOS.
- Initialize the environment
- Retrieve the app to be built
- Choose configuration based on transport and host.
- Build the app using west command line tool.

Flashing the firmware

- Checking if firmware exists: Checking if there is any firmware setup already established is part of the validation stage to ensure that the firmware folder is not
- Flash specific firmware folder if needed: Depending on the RTOS and its approach to implementation for a certain firmware, the firmware folder may be customized or generic.
- Open On-Chip Debugger, such as openocd, supports flashing by providing an interface and a specific configuration file for the board.
- Dealing with boards that have special openocd rules.
- Using the west command line tool, get information about the zephyr SDK and Flash environment.

3.2 Freertos

Create step

- Retrieving RTOS and Platform: Validation occurs by examining the directories, depending on the amount of arguments given by the user and the combination of rtos and board.
- Checking if firmware exists: Checking if there is any firmware setup already established is part of the validation stage to ensure that the firmware folder is not overwritten.

- Checking folders: Following the validation stage, the files pertaining to the rtos's specific platform are examined to see if they are there before producing the firmware for that platform and rtos combination.
- Setting common environment: Set up a shared environment for DDS (middleware) for microcontroller applications that require access to the publisher/subscriber architecture for information sharing.
- Check generic build: Check to see if a generic implementation exists for all platforms (check it deep once)
- Creating development directory and mcu directory :Make a dev ws directory with ros2 packages and apps, as well as a mcu directory (having micro ros related repos and ros2 packages).
- Build the development workspace: If there is an existing build environment, clean it and construct the workspace using the colcon command, then install the setup.
- Install dependencies for specific platform: Install platform-specific dependencies among all platform dependencies present in a given ros distribution, such as "foxy," while ignoring a few keys. Creating a firmware folder with all of the necessary information.
- Installing toolchain: GNU Arm Embedded Toolchain is a ready-to-use, open-source collection of tools for C, C++, and assembly development. Download the arm compiler.
- Import repos related to the board: Import the freertos-related apps and configure them to run on target boards.(Discussed in brief below)
- The ros dependencies for the board included in the mcu workspace were installed.
- Analysis of freertos board repos: Because different microcontroller families have distinct architectures, the approach taken by freertos varies.

  For freertos to run on stm32 family board the following are needed:

  Requirement:

  - Drivers:Most embedded devices requires software initialization and administration. The software that interacts with and controls this hardware directly is known as a device driver.
  - HAL driver:The hardware abstraction layer (HAL) is a programming layer that allows a computer operating system to communicate with a hardware device at a higher level of abstraction than the detailed hardware level. The HAL drivers include a comprehensive collection of ready-to-use APIs that simplify user application development.
  - CMSIS: The Common Microcontroller Software Interface Standard (CMSIS) is a vendor-independent abstraction layer for microcontrollers using Arm Cortex processors. CMSIS establishes generic tool interfaces and ensures that devices

are continuously supported. The CMSIS software interfaces make reusing software much easier. CMSIS communicates with real-time operating systems and middleware, as well as CPUs and peripherals. CMSIS includes a device, board, and software delivery system, as well as the ability to mix and match software components from a variety of vendors.

- Include:This is where you'll find all of the header files for source files.
- Some header files worth mentioning are:
  - FreeRTOSConfig.h : FreeRTOS is customized using the FreeRTOSConfig.h configuration file. Every FreeRTOS software must include a FreeRTOSConfig.h header file in its pre-processor inclusion path. Using FreeRTOSConfig.h, the RTOS kernel is customized for the application. Because it is specific to the application rather than the RTOS, it should be stored in an application directory rather than one of the RTOS kernel source code directories.
  - Allocator.h: Returns a pointer to the memory block after allocating a memory block with the size of the memory block in bytes.
- Middleware:

  Third party middleware's are used here
  - Lwip:The open source LwIP (Lightweight IP) middleware TCP/IP stack is targeted for embedded devices.
  - Contains data related to portable and cmis rtos v2.

  Source: The logic for all the files in the include folder is present here, including main.c (which contains the main body program), freertos.c (which contains freertos apps), transport, memory allocation, and so on.

  flash.ld files:Some bytes FLASH and some Kbytes RAM linker script for STM32 series.

  microros.ioc: Includes code generating parameters that may be customized using the inbuilt STM32CubeMX editor, and aids in the creation of Cmake project structures.

  For freertos to run on esp32 family board the following are needed:
  - Main , Cmake , Toolchain , Sdk configuration, Libraries etc...

Observations: Based on the foregoing research, freertos takes a unique approach to each board family's hardware, i.e. there is no general strategy.

Configure the firmware

- Checking if firmware exists and configure script exist: Part of the validation process is to see if the firmware and configuration scripts are present before moving on to the next step.
- Parsing micro-ROS arguments: Checks for the availability of the provided app and handles the configuration relevant to how data is transported:

  The options available for this configuration is:

- transport or -t: udp, serial or any hardware-specific transport label
- dev or -d: agent string descriptor in a serial-like transport
- ip or -i: agent IP in a network-like transport
- port or -p: agent port in a network-like transport

Configure specific firmware folder if needed: In this phase, based on the transport type, the specific firmware folder related to a combination of RTOS and board is configured, and the generic implementation of DDS middleware configuration is also set.

Set the DDS and Transport: DDS-related parameters are taken care of, and transport is determined based on the information provided by the user.

Building the firmware

- Parse cli arguments: The arguments are related to fast build, verbose mode etc .Command line arguments are simple arguments that are stated on the system's command line following the program's name, and the values of these arguments are passed on to the program during program execution.
- Checking if firmware exists: A stage in the validation process that checks if the firmware is present.
- Clean paths : Setup a clean atmosphere Before creating a specific firmware folder, it is preferable to source it rather than run it.
- Building the specific firmware folder: By configuring the utilities, cleaning the microros build, generating a microros stack, and lastly building the specific firmware, you can create the firmware folder that is supported by RTOS.
- Initialize the environment
- Retrieve the app to be built
- Choose configuration based on transport and host.
- Build the app using west command line tool.

Flashing the firmware

- Checking if firmware exists: Checking if there is any firmware setup already established is part of the validation stage to ensure that the firmware folder is not
- Flash specific firmware folder if needed: Depending on the RTOS and its approach to implementation for a certain firmware, the firmware folder may be customized or generic.
- Open On-Chip Debugger, such as openocd, supports flashing by providing an interface and a specific configuration file for the board.

Chapter 4

Implementation

4.1 Introduction

We can now experiment with integrating microros on an RTOS for a certain hardware with all of the information we've obtained. The application viewpoint technique is used, which entails making modifications from the application towards the hardware.

The approach focuses on adapting the working setup for a reference STM32 board named olimex-e407 by understanding how microros headers are being incorporated onto the the reference board as the method followed here is by doing systematic changes from application(top) to hardware(down) and can be called a topdown approach.

4.2 Top down approach for integrating microros+freertos on any stm32 family board:

In order to reproduce the output the following initial setup are needed:

- olimex-e407 related ioc file [1] in local pc.
- Folders of microros+freertos+olimex-e407 related setup [2]
- Add the following extensions (diff folders, c/c++ related extensions as per VS code suggestions) to the VS code editor for quick comparison of directories and files.
- Stm32Cubemx for generating code related to B-L475E-IOT01A Discovery kit for IoT node [3]
- ROS2 foxy installation.
- Clone the microros setup[4] and freertos[5] apps GitHub repositories for making changes.

Changes that needs to be made in the GitHub repository cloned:

- Freertos folder in Config folder [6] :Create a custom folder with boards name in my case it is discovery l475 iot1.

  The following files are required to be arranged in the folder created and changes made to them:

First clone the content from repo[7] or any stm32 related board folder contents.

- board.repos : Change url to your cloned freertos apps[8] repo. The structure that needs to be followed in the freertos _apps repo will be discussed soon as of now make a folder named microros_your target board for example microros_discovery_l4751_iot_extensions.

  build.sh: Change the EXTENSIONS _DIR=FW_TARGETDIR \freertos_apps\microros_

  olimex_e407_extensions To your intended directory for example

  EXTENSIONS_DIR=FW_TARGETDIR \freertos_apps\microros_discovery_l475_iot1_

  extensions

  client-colcon.meta: This file contains the middleware configuration and rcl related repos it can be customized as per user needs.

  client _uros .repos: No changes required as it is common for all boards supported by freertos.

  configure.sh : Change the EXTENSIONS =FW _TARGETDIR/freertos _apps/microros _olimexe407_extensions To your intended directory for example EXTENSIONS_DIR=FW _TARGETDIR/freertos_ndapps/microros _discovery 475_iot1 _extensions

  create.sh: As the GNU Arm Embedded Toolchain is supported by stm32 there is no need to change this file.

  flash.sh:Open ocd on chip debugger is used and a combination of interface and target board config file needs to be selected. Changes needs to be made here[9].

Changes that needs to be made for freertos_apps github repo:

Clone the repo onto your personal laptop and start the following procedure :

- Start the stm32cubemx select the required board in my case it is B-L475E-IOT01A create a new project it is better to use use the default configuration

  Choose connectivity as per you board example for stm32 iot board we need to set usb connectivity make sure that the connectivity is not partially disabled.(Refer figure 4.2)

  Under middleware choose freertos as middleware.

  Check the clock configuration [10][11] videos for setting up clock configuration as project is created using the project with default configuration it is fine.(Figure 4.4)

  Under project management tab make sure that the toolchain/IDE is cmake and name your project.

  Note : once the project directory where it is being created is set and we generate we can't change it so makesre that the project ioc file is present the folder related to our target board.

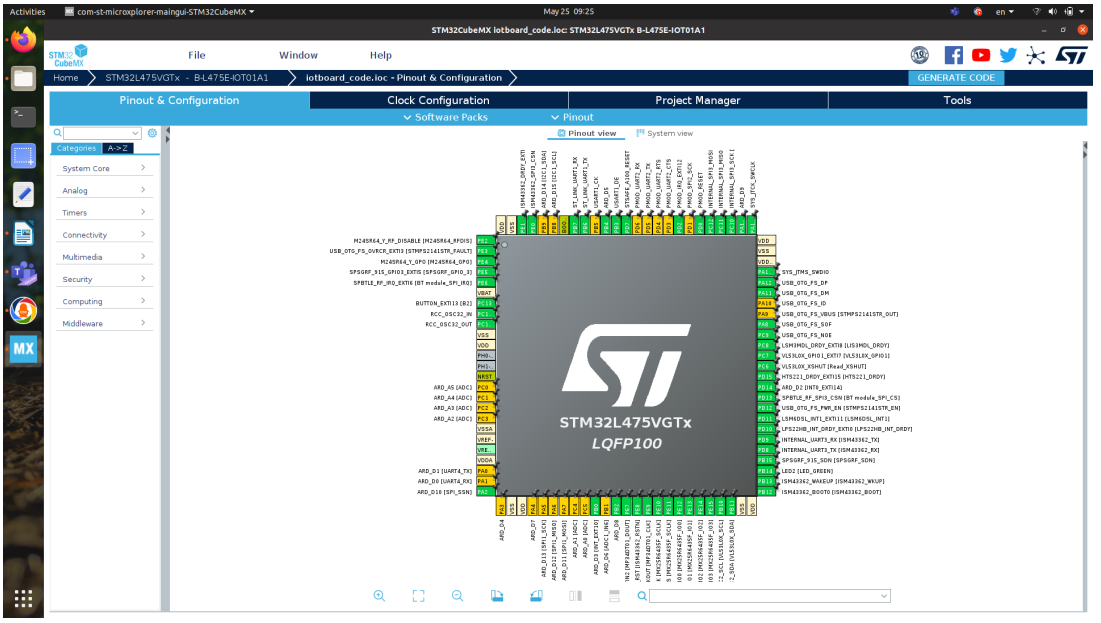

Figure 4.1: Pin configuration

generate the code from ioc file the folder which would look similar like this.

The structure depends on the board selected, toolchains ,rtoses etc But will have Drivers , inc , src , Makefile and starupstm32.s and stm32 flash.ld files in common.

.Generating code for a particular stm32 family using freertos as middlware is completed till this point.

Before going to the next steps it is very helpful to try out external build system for [12]microros on stm32 microcontroller.

How is it helpful :

Doing this exercise makes us familiar with the following :

- Making the changes in cmakefile while integrating microros
- Integrating the cubemx generated code with microros.
- Knowing how to flash the microros application with open ocd.
- Experience with errors faced while trying to build app ranging from variables to adding system calls.
- Some steps that are specific to target board that helps us to understand board specific requirements and its behavior.

In case there is no time refer to this video here[14]

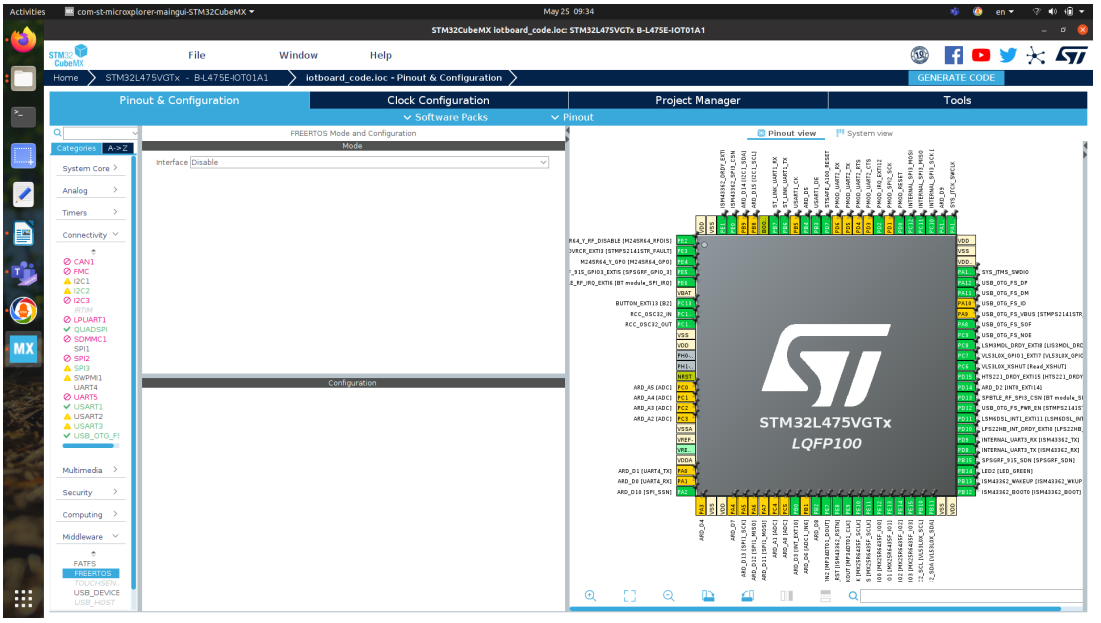

Figure 4.2: Board Connectivity

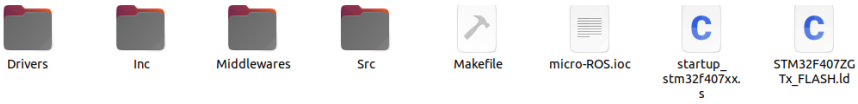

Figure 4.3: Generated code structure

Before making changes keep the following setup ready:(VS code setup)

- Generate the olimex (olimex-stm32-e407):ioc file code you can find the folders etc... generated in the particular file location mentioned project management part while opened by stm32 cubemx.(assume it folder1)
- folders of microros+freertos+olimex-e407 related setup click here clone it to your local computer.(assume it folder 2)
- Create a folder named compare having 2 categories of folders one original(having the folders that are of folder1) and generated (having folders that are generated from olimex ioc file) This is only for reference.

  Arranging the folder in this format eases the process to a large extent and thanks to VS code for it UI and extensions.

  This setup allows us to compare folders and files present and categorizes the 2 compared folders as unchanged files , files needed to be added , files that are deleted.

  For example lets compare Originalsrc(contains the code from this) folder with generated src folder (contains the code of src folder that is generated by olimex ioc file)

While making any changes do the following:

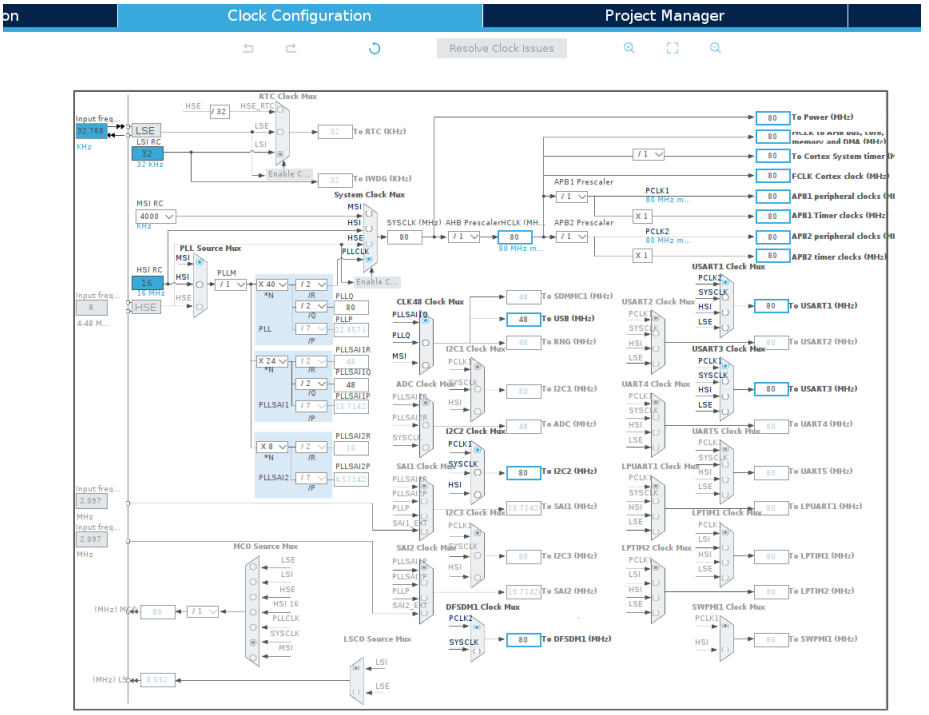

Figure 4.4: Clock configuration

- Compare the file/folder of microros+freertos+board code by microros community with freertos+board code generated by stm32cubemx.
- See how similar the target board for example stm32 iot board is with olimex ioc file generated code.
- Make the changes as per the pin configuration , board , connectivity options chosen and microros related code present in (folder1).

There are changes that needs to be made for this generated code is as follows :

- Add FreeRTOS-Plus-POSIX folder , toolchain file , include folder
- Make changes in cmakefile using (VS code setup)
- Change the build directories

The changes made here starts from main.c from and changing and including the files related for making microros+freertos+board application to run.

After making all the changes :

- Creating the workspace : Follows the commands from here[13] with the following changes in while creating workspace and microros tool download setp
- Make sure to make changes as per the target hardware repos*.
- Normally follow the microros build instructions and it works.

Ros build steps and comments on it :

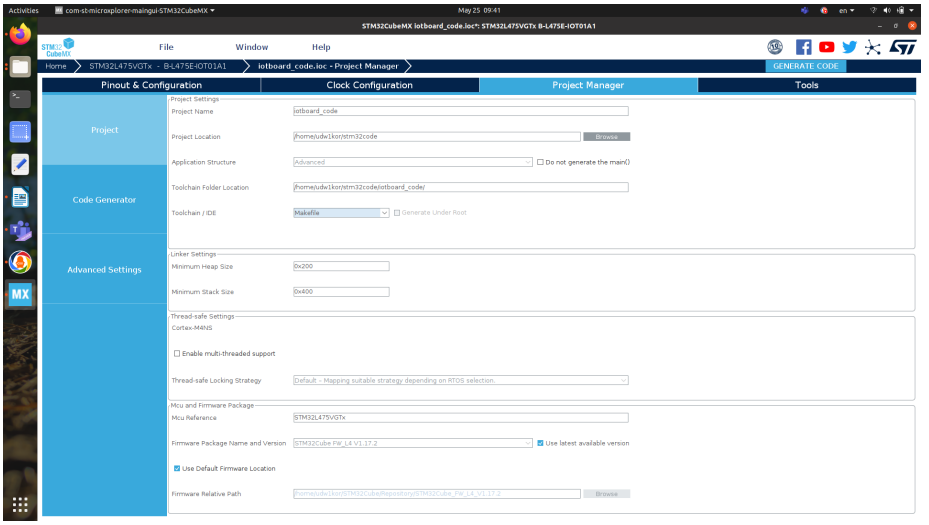

Figure 4.5: Project manager

- Create step:
  - Errors faced :
  - /home/udw1kor/topdown/install/microrossetup/config /freertos/discoveryl475iot1/configure.sh To prevent the error all files permissions needs to be changed

The Configure , Build , Flash steps which are the procedure for microros build can be reproduced with no errors..

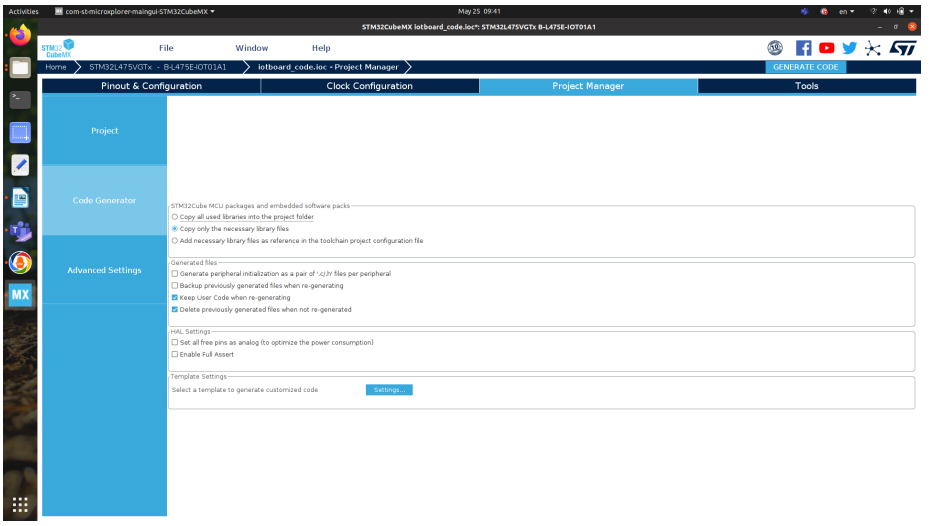

Figure 4.6: Code generator

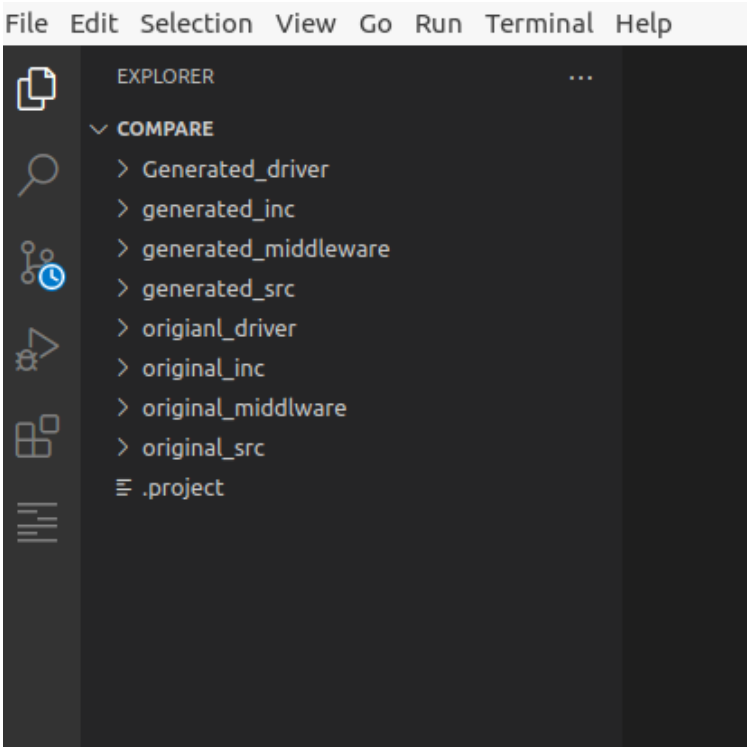

Figure 4.7: folder structure only for reference

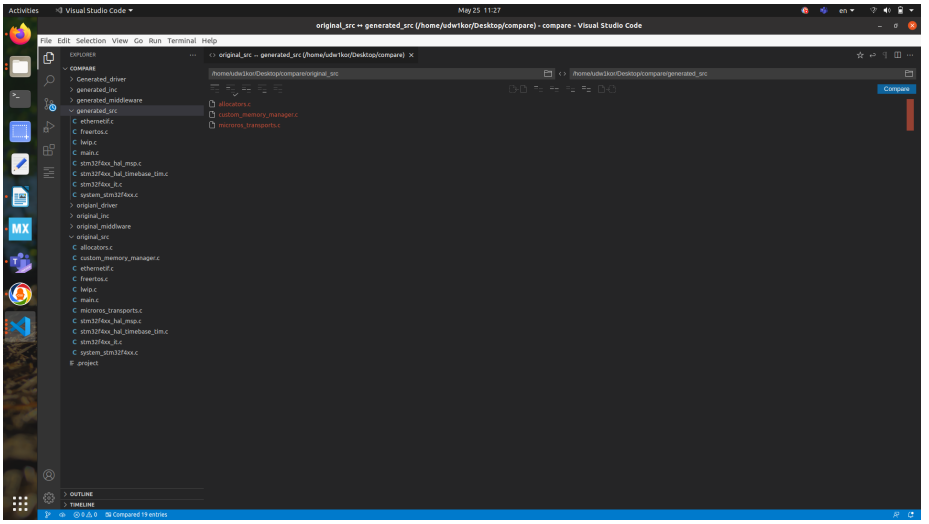

Figure 4.8: folder structure only for reference

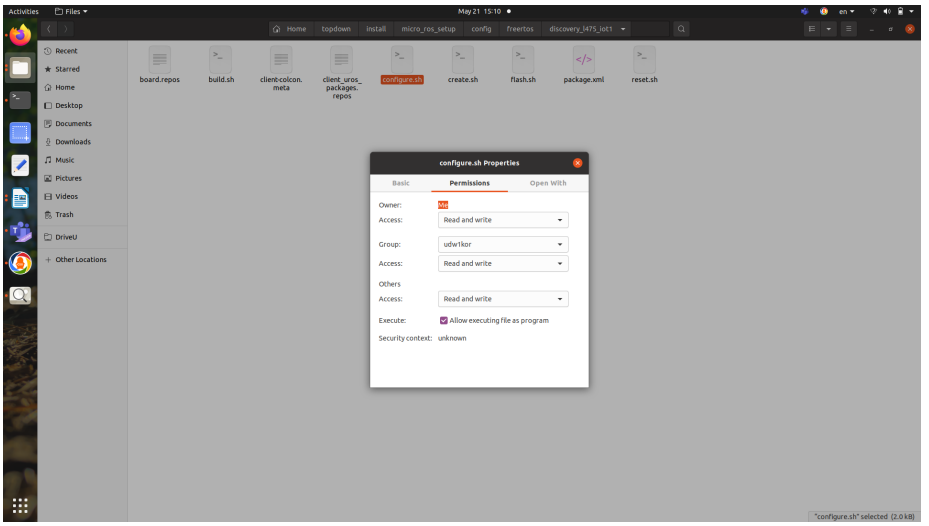

Figure 4.9: Solution for create step error

Chapter 5

Conclusions and future scope

Integration of microros onto automotive helps in bringing the robotic framework onto the vehicles and provides a simple integration of a variety of tools, including as visualisation of robot kinematics and sensor data, path planning and perception algorithms, and lowlevel drivers for commonly used sensors.The level of complexity in software development required in robotics projects can be considerably decreased with the help of ROS. ROS has an integrated framework and toolsets for robotics development, which speeds up software creation and aids in redistribution there by leading to new and faster innovations.By attempting to incorporate microros onto a target board and keeping the technique as general as feasible, this is a first step in understanding the feasibility of using microros in automotive domain.

Chapter 6

Bibliography

1. https://github.com/micro-ROS/freertos_apps/blob/humble/microros_olimex_e407_extensions/micro-ROS.ioc

2. https://github.com/micro-ROS/freertos_apps/tree/humble/microros_olimex_e407_extensions

3. https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html

4. https://github.com/micro-ROS/micro_ros_setup/tree/foxy

5. https://github.com/micro-ROS/freertos_apps/tree/foxy

6. https://github.com/micro-ROS/micro_ros_setup/tree/foxy/config/freertos

7. https://github.com/micro-ROS/micro_ros_setup/tree/humble/config/freertos/olimex-stm32-e407

8. https://github.com/micro-ROS/freertos_apps/tree/foxy.git

9. https://github.com/micro-ROS/micro_ros_setup/blob/humble/config/freertos/olimex-stm32-e407/flash.sh#L23

10. https://www.youtube.com/watch?v=o6ZWD0PAoJk

11. https://www.youtube.com/watch?v=Ntmgk2KnRRA

12. https://micro.ros.org/docs/concepts/build_system/external_build_systems/

13. https://micro.ros.org/docs/tutorials/core/first_application_rtos/freertos/

14. https://www.youtube.com/watch?v=xbWaHARjSmk

a1

ORIGINALITY REPORT

19%	18%	4%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	micro-ros.github.io Internet Source	3%
2	www.bosch.com Internet Source	2%
3	micro.ros.org Internet Source	2%
4	www.coursehero.com Internet Source	2%
5	docs.zephyrproject.org Internet Source	1%
6	www.mytechlogy.com Internet Source	1%
7	www.fiware.org Internet Source	1%
8	jessicasiboro.blogspot.com Internet Source	1%
9	Miskam, Mohd Azfar, Syamimi Shamsuddin, Hanafiah Yussof, Abdul Rahman Omar, and Muhammad Zaiyad Muda. "Programming	1%

platform for NAO robot in cognitive interaction applications", 2014 IEEE International Symposium on Robotics and Manufacturing Automation (ROMA), 2014.
Publication

10	developer.arm.com Internet Source	1 %
11	b-ok.org Internet Source	1 %
12	Than D. Le, An T. Le, Duy T. Nguyen. "Model-based Q-learning for humanoid robots", 2017 18th International Conference on Advanced Robotics (ICAR), 2017 Publication	<1 %
13	github.com Internet Source	<1 %
14	archive.mu.ac.in Internet Source	<1 %
15	webthesis.biblio.polito.it Internet Source	<1 %
16	scholars.lib.ntu.edu.tw Internet Source	<1 %
17	www.arm.com Internet Source	<1 %
18	Advances in Intelligent Systems and Computing, 2014.	<1 %

Publication

19	Supriya Katwe, Nalini Iyer. "Map-Based Particle Filter for Localization: Autonomous Vehicle", 2020 International Conference on Computational Performance Evaluation (ComPE), 2020 Publication	<1 %
20	www.asee.org Internet Source	<1 %
21	www.mail-archive.com Internet Source	<1 %
22	Daneil Steen Losvik, Adrian Rutle. "A Domain-Specific Language for the Development of Heterogeneous Multi-robot Systems", 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2019 Publication	<1 %
23	dokumen.pub Internet Source	<1 %
24	"Recent Findings in Intelligent Computing Techniques", Springer Science and Business Media LLC, 2018 Publication	<1 %