INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS

CLUSTER AUTOMOTRIZ RECONFIGURABLE BASADO EN MICROPROCESADOR CON ARQUITECTURA INTEL x86_64

Trabajo recepcional que para obtener el diploma de

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Adriana Lucía Moreno Vergel

Asesor: Raúl Campos Rodríguez
Asesor: Héctor Antonio Rivas Silva

Tlaquepaque, Jalisco. 15 de noviembre de 2016.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS

CLUSTER AUTOMOTRIZ RECONFIGURABLE BASADO EN MICROPROCESADOR CON ARQUITECTURA INTEL x86_64

Trabajo recepcional que para obtener el diploma de

ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: Adriana Lucía Moreno Vergel
Becario CONACYT No. 424478

Asesor: Raúl Campos Rodríguez
Asesor: Héctor Antonio Rivas Silva

Tlaquepaque, Jalisco. 15 de noviembre de 2016.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS

RECONFIGURABLE AUTOMOTIVE CLUSTER BASED ON INTEL x86_64 MICROPROCESSOR ARCHITECTURE

Final work that to obtain the diploma of

EMBEDDED SYSTEM SPECIALIST

Presents: Adriana Lucía Moreno Vergel
CONACYT Scholarship No. 424478

Advisor: Raúl Campos Rodríguez
Advisor: Héctor Antonio Rivas Silva

Tlaquepaque, Jalisco. 15 de noviembre de 2016.

AKNOWLEDGMENTS

First and above all, I praise God, the almighty for providing me this opportunity and granting me his blessings to finish this thesis successfully.

The author wishes to express her appreciation to her advisors Dr. Raúl Campos and M.Sc. Hector Rivas, who guided her during this thesis work. Also I would like to acknowledge the support of Raul Camacho who guided me throughout this project.

I wish to extend my gratitude to Dr. Raúl Campos who helped me to review this work and for providing me the necessary elements to successfully finish it. I'm also grateful to ITESO who gave access to the laboratory and research facilities. Without their precious support it would not be possible to conduct this research.

I place on record, my sincere thank you to CONACYT, for the received scholarship number 424478.

I am also grateful to my parents, brother and sister for always supporting me unconditionally during this thesis development and my life in general. Also, Thanks to Elias Reyes, for all his love and encouragement.

Last but not the least, I would like to thank Continental, the company where I current work, for providing me the opportunity to grow both professionally and personally, also for the valuable support they have offered me.

DEDICATION

To my parents, William and Marlene, for their love, effort and encouragement.

Thanks to you I have achieved one more of my aspirations.

RESUMEN

Los clusters automotrices reconfigurables ofrecen amplias posibilidades de funcionalidad y personalización de contenido. Actualmente, una aplicación llamada CLUSTER_PI está disponible y corre bajo la arquitectura ARM en una tarjeta Raspberry Pi. Las aplicaciones configuradas en este microcontrolador requieren una gran demanda de procesamiento de datos y no se le puede dar seguimiento debido a la limitada capacidad de memoria. Para proyectos con mayor demanda de gráficos es necesario tener una arquitectura que permita mayor procesamiento de información.

Debido a que los proyectos futuros necesitan más demanda de información, capacidad de procesamiento y gráficos, se ha optado por cambiar a la plataforma Minnowboard Max la cual ofrece más prestaciones que la Raspberry Pi. El objetivo principal de este trabajo es migrar la aplicación CLUSTER_PI de la arquitectura ARM a x86_64. Para lograr este objetivo, es necesario configurar un nuevo kernel de acuerdo a los requerimientos de la aplicación y posteriormente compilarlo. Para la configuración y compilación del kernel con los módulos, librerías y configuraciones necesarias se utiliza Buildroot. Esta herramienta utiliza compilación cruzada y además permite la creación de un sistema embebido Linux, el cual incluye: gestor de arranque, sistema de archivos raíz, imagen del kernel y herramienta de compilación cruzada. En primer lugar se revisan los paquetes necesarios para ejecutar la aplicación CLUSTER_PI tales como QT5 y SSH, con los cuales se procede a configurar el Kernel.

Los componentes más importantes así como sus dependencias son explicados a detalle en la Sección 4. Una vez guardada la configuración se crea el archivo Makefile, el cual se encarga de: descargar los paquetes; configurar, construir e instalar las herramientas de compilación; generar la imagen del kernel, el gestor de arranque y el sistema de archivos raíz. Después de generar el sistema Linux, se procede a ejecutar la aplicación CLUSTER_PI en la tarjeta Minnowboard Max obteniendo las mismas imágenes y funcionalidades que en la Raspberry Pi.

Con el desarrollo de este trabajo se da oportunidad a que nuevos trabajos de investigación surjan basados en la arquitectura x86_64, permitiendo así mayor soporte a otras aplicaciones automotrices, mejor desempeño de gráficos y escalabilidad a sistemas computacionales.

ABSTRACT

Reconfigurable automotive clusters offer extensive functionality and content customization capabilities. An application called CLUSTER_PI is currently available and runs under the ARM architecture on a Raspberry Pi board. Applications configured on this microcontroller require a high demand for data processing and cannot be tracked due to limited memory capacity. For projects with a greater demand for graphics, it is necessary to have an architecture that allows more information processing.

Because future projects need more information demand, processing capacity and graphics, Minnowboard Max is chosen over Raspberry Pi since it offers more features and performance. The main objective of this work is to migrate CLUSTER_PI application from ARM to x86_64 architecture, for this, a new kernel must be configured according to the application's requirements and then compiled. Buildroot is used to configure and compile the Kernel with the necessary modules, libraries and configurations. This tool uses cross-compilation and also allows the creation of a complete Linux embedded system, which includes: bootloader, root filesystem, kernel image and cross-compilation toolchain. At first, mandatory packages for CLUSTER_PI App like QT5 and SSH are selected in the kernel configuration.

The most important components and their dependencies are explained in detail in Section 4. Once the configuration is saved, the Makefile is created, which is responsible for: downloading the packages; configure, build and install the compilation tools; generate the kernel image, the bootloader, and the root filesystem. After generating the Linux system, CLUSTER_PI application is run on the Minnowboard Max obtaining the same graphic environment and functionalities as in the Raspberry Pi.

With the development of this work, it is possible that new research work will emerge based on the x86_64 architecture, thus allowing greater support to other automotive applications, better graphics performance and scalability to computer systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

CAN	Control Area Network.
CLUSTER_PI	A previous project developed a reconfigurable automotive cluster application, called “CLUSTER_PI App” in this document for future references, based on ARM architecture and deployed on Raspberry PI board.
Cross-compilation	Process whereby compilation is done on a PC (host) with the purpose to be executed on other system, called target (target). When cross-compilation is done, executables for another system are produced. This happens when the target does not have the toolchain native compilation or when the host is faster and has better resources (CPU, memory, etc.).
Distro	Linux distribution.
DRI	Direct Rendering Infrastructure.
EGL	OpenGL ES Native Platform Graphics Interface
Host	Laptop in charge of executing kernel and application cross-compilation. It also is connected to the embedded platform through Ethernet for sharing information.
GUI	Graphical User Interface.
IPC	Inter-process Communication.
HMI	Human Machine Interface.
HW	Hardware.
mSD	Micro SD.
OpenSSH	Open Secure Shell.
OpenGL	Open Graphics Library.
OS	Operative System.
RAM	Random Access Memory.
Rootfs	Root filesystem.
RTOS	Real-Time Operating System
Toolchain	A set of tools for software development, often used in sequence so that the output of one tool comprises the input of the next.
SDK	Software Development Kit.
SW	Software.
X11	X Window System.

1. INTRODUCTION

Abstract: *This chapter briefly presents the background of the object of study, problem definition and justification.*

1.1. Background

Digital instrumentation is trend in different industrial fields such as aeronautics and automotive. A digital cluster offers great flexibility and a reach set of visualization schemes. Visual effects such as 2D and 3D graphics are features to be included in the new models of automobile industry. The digital instruments allow flexibility in the instrument visualization, and a kind of driver personalization.

An embedded system for a reconfigurable automotive cluster is a demanding application, which has to offer a maximum of flexibility of visualization, functionality and reach information content. The digital instrumentation allow different configuration for the instrument panel depending on the situations or driver needs. There are some intermediate commercial solutions in the market, for example [1] and [2]. However it requires fast initialization and response, thus a high performance embedded system must be considered for this application.

The embedded system proposed in this work is based on the Intel Minnowboard Max, which is intended to be used as the hardware platform in reconfigurable cluster application. Since CLUSTER_PI application is going to be migrated from a different architecture, a new Kernel needs to be compiled. It shall be configured according to the requirements of the target the application.

In this project, Kernel will be built with just the required modules, libraries and configurations. The Kernel can be configured through several tools, such as Buildroot, Yocto, or OpenWRT, among others. This tools simplify and automate some the process of building a complete Linux system for an embedded system, using cross-compilation [3], [4], [5]. In this project, the Buildroot toolchain is chosen since it offers an easy to use structure and a lot of documentation. Additionally, the target application CLUSTER_PI was developed based on it.

1.2. Justification

The digital instrumentation is a trend in several fields of the industry. A digital cluster offer a great flexibility in the presentation of the information in production lines, control panels, airplanes and cars. Since most of the visual effects and flexibility is based on embedded software, it is easy to develop one time and to program many times as needed. It simplifies the development and deploy processes. Additionally, basic functionality could be delivered at a first stage and further developments, updates and bug fixings are possible.

1.3. Problem

Several instrument clusters types have been developed in different industry fields. In this work, an automotive digital cluster is implemented. An x86_64 development board is used as hardware platform and a Linux distro as software supporting the development. The QT framework is implemented as visualization framework.

1.4. Objectives

1.4.1. General Objective:

To develop a digital automotive cluster based on an x86_64 hardware platform and an embedded Linux software platform.

1.4.2. Specific Objectives:

1. To develop a digital automotive cluster that represents basic instruments such as odometer, velocimetry, temperature, and battery level, among others.
2. To use the Intel Minnowboard Max development board as the hardware platform
3. To use an embedded Linux distro as software platform
4. To use the QT cross development platform for the graphical representation of the cluster instruments.

2. STATE OF THE ART

Abstract: *This chapter briefly reviews some works that are relevant in the development of this project.*

2.1. Mentor Graphics Digital Instrument Cluster

Mentor Graphics has developed a flexible intermediate solution for the construction of digital instrumentation cluster, allowing a reach and flexible set of tools for the presentation of graphical instruments. The solution allows mixing critical instruments with non-critical information in a smooth fashion.

The solution of Mentor Graphics provides a rich set of HMI 3D and 2D graphics, based on embedded Linux and proprietary RTOS. It is compatible with the QT framework, Socionext CGI Studio, and Altia, with a high performance graphics up to 120 fps. The Figure 1 shows a layout of the Mentor Graphics' instrument cluster [15].

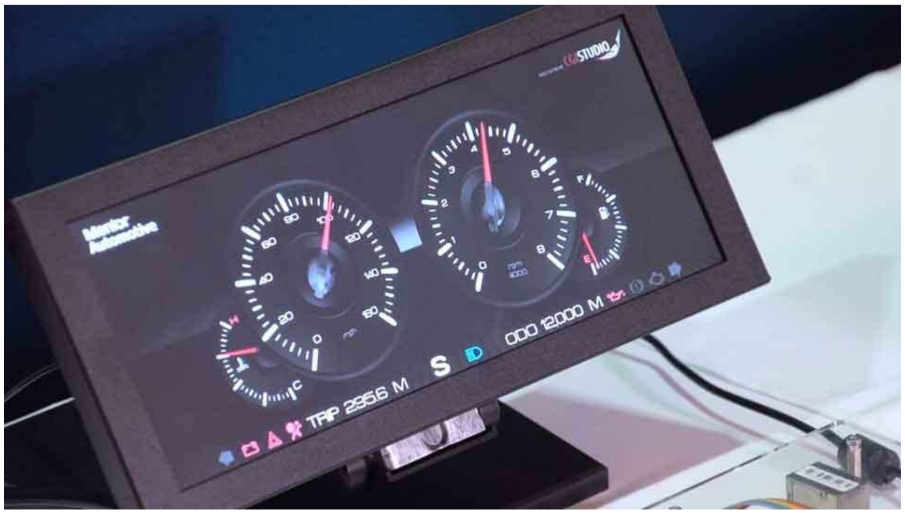


Figure 1. The Mentor Graphics' Automotive Instrument Cluster [15].

2.2. Librow Automotive Digital Instrument Cluster

Librow offers development and production of digital instruments cluster for automotive applications. The company also provides a reengineering, virtualization, development and manufacturing of existing mechanical and electromechanical clusters, including all the graphical design, hardware customization and embedded software development.

The solution provided by Librow includes 2D and 3D graphics, user interface, RTOS and low level drivers. The Figure 2 depicts a layout of the Librow's digital instrument cluster [16].

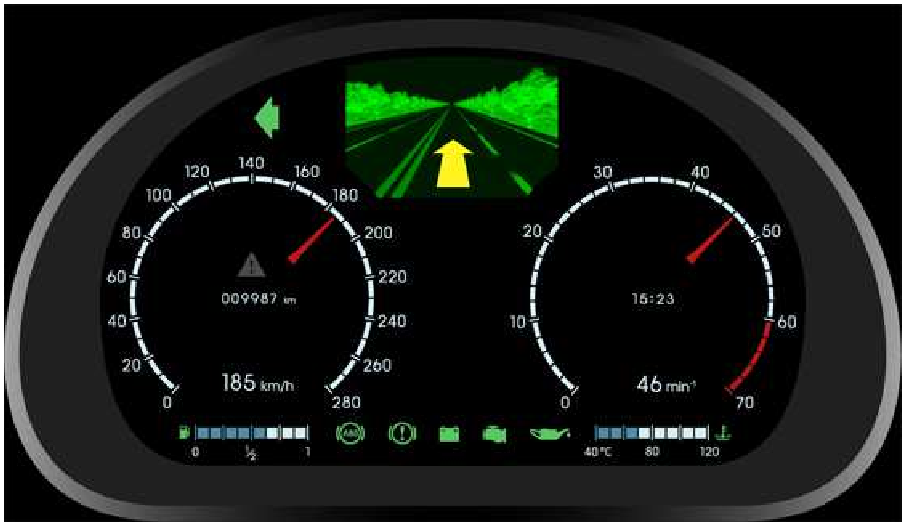


Figure 2. The Librow’s Digital Instrument Cluster [16].

2.3. NVidia Digital Instrument Cluster

NVidia addresses the high demanding cluster design of new car instruments by its NVIDIA DRIV CX solution portfolio. It is an embedded solution empowering high demanding 3D features for instrument visualization and advanced infotainment cluster, speech recognition and image processing capabilities.

The NVIDIA Drive PX 2 is a single processor solution ideal for image processing, real time applications and speech recognition, among others. The architecture of the NVIDIA solutions includes a pipe line for neural network processing. The Figure 3 shows a layout of the Drive PX 2 solution [17].

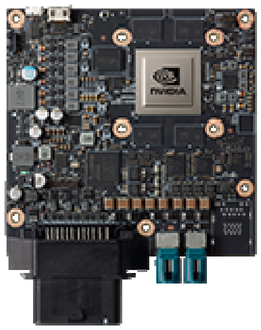


Figure 3. NVIDIA Drive PX 2 and Digital Navigation System [17]

2.4. Dakota Digital Gauge Cluster

Dakota Digital offers a series of digital gauge cluster for almost every requirement in the industry. The company provides complete clusters and individual instruments in different shapes, such as round traditional instruments and rectangular shaped instruments. The Figure 4 shows the layout of the Universal 5 Gage Cluster and the Round Digital Instrument VFD3 [18].

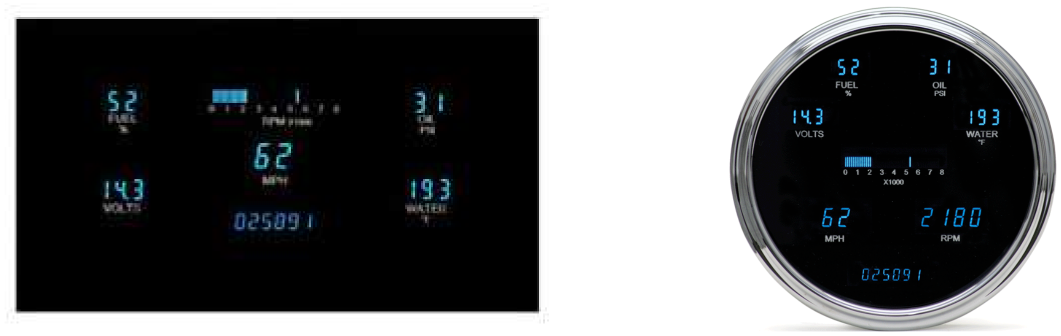


Figure 4. Dakota Digital Dash Universal Design [18]

3. CONCEPTUAL FRAMEWORK

Abstract: *Theoretical and conceptual framework are described in this chapter. Buildroot, the tool used to build the embedded system is described here as well as most important packages required to deploy CLUSTER_PI App.*

3.1. Introduction

In this chapter theoretical and conceptual framework is provided, Buildroot and most important packages are described in Section 3.2. Buildroot is a tool able to provide the bootloader (Section 3.3**Error! Reference source not found.**), the kernel image (Section 3.4), the root filesystem (Section 3.5) as well as the cross-compilation toolchain (Section 3.6) for the embedded system. Also, CLUSTER_PI App makes use of shared memory and it needs to be supported in the kernel (Section 3.3).

3.2. Buildroot

Buildroot is tool that automates the process to generate embedded Linux systems through cross-compilation. It is able to generate bootloader, kernel image, root filesystem and cross-compilation toolchain. Also, it provides a configuration environment similar to kernel menuconfig, xconfig or gconfig and support several packages like QT5, X.org, OpenGL, among others [3].

The basic commands that allow to generate kernel image fitting the user needs with all the features and packages selected are: make menuconfig|nconfig|gconfig|xconfig and make [3].

3.2.1. QT

QT is a cross-platform application framework used mainly for developing application SW with GUI, it works on several platforms like X11 and EGLFS (Linux) [6].
QT5.4.1 was used in CLUSTER_PI App, so this or any QT version onwards will work. In this project QT5.6.0 is used with EGLFS (EGL Full Screen) support.

3.2.1. X Window System (X11)

X11 is a graphical windowing system based on a client/server model. It provides the basic framework for building GUI environments [7].

3.2.2. Mesa3D

Mesa3D is a graphics library that provides an open source implementation of OpenGL to render graphics in several platforms. It provides the Direct Rendering Infrastructure (DRI), which is a framework that allows direct access to graphics hardware under the X Window System in a safe and efficient manner. [8][9].

3.2.3. OpenSSH

OpenSSH is a connectivity tool for remote login with the SSH protocol. It is a suite of programs that can be used for remote operations and key management [10].

3.3. Bootloader

Bootloader or boot manager is a simple program designed to prepare what is needed to let the operative system work, i.e. it is in charge of place OS into memory. It resides on the master boot record (MBR) and it is called after basic input-output system (BIOS) performs some test after a computer is powered-up or restarted [11].

3.4. Kernel Image

It is a compressed kernel file stored in /boot partition. In this project, bzimage (zimage compressed to the max) is the kernel binary format selected in buildroot.

3.5. Root Filesystem

It is the filesystem contained in the same partition on which the root directory (/) is located; all other file systems are mounted below it as the system boots up. Main subdirectories under root filesystem are [12]:

/bin: Commands needed during bootup that might be used by normal users (probably after bootup).

/sbin: Like /bin, but the commands are not intended for normal users, although they may use them if necessary and allowed. /sbin is not usually in the default path of normal users, but will be in root's default path.

/etc: Configuration files specific to the machine.

/root: The home directory for user root. This is usually not accessible to other users on the system.

/lib: Shared libraries needed by the programs on the root filesystem.

/lib/modules: Loadable kernel modules, especially those that are needed to boot the system when recovering from disasters (e.g., network and filesystem drivers).

/dev: Device files. These are special files that help the user interface with the various devices on the system.

/tmp: Temporary files. Programs running often store temporary files in here.

/boot: Files used by the bootstrap loader, e.g., LILO or GRUB. Kernel images are often kept here instead of in the root directory.

/mnt: Mount point for temporary mounts by the system administrator. Programs aren't supposed to mount on /mnt automatically. /mnt might be divided into subdirectories.

/proc, /usr, /var, /home: Mount points for the other filesystems.

3.6. Toolchain

A toolchain is a set of development tools linked together used to build embedded software. It may consist of a compiler, linker, libraries and debugger. In embedded development, using a cross toolchain (cross compiler) is very common; host machine (e.g. x86_64) produces binary code for the target (e.g. ARM) [13].

3.7. Inter-process Communication (IPC)

IPC allows the processes to communicate with each other, this is done by means of kernel. It is requested to allocate the space which can be used to communicate between processes, which can open, and read/write the file or variables. There are several IPC mechanisms, below are described the ones used in CLUSTER_PI App [14]:

- Shared memory: Memory segment where values can be exchanged, a segment can be created by one process, and subsequently written to and read from by any number of processes. The information addressed by the calling process is mapped directly from a memory segment (Figure 1).

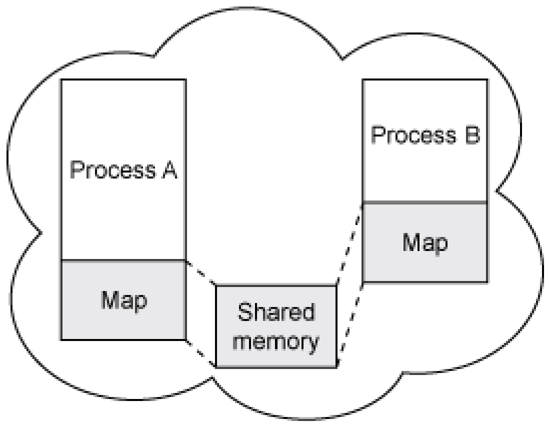


Figure 1. The shared memory segment is mapped by Process A and Process B [15]

- Semaphores: When write access is allowed for more than one process, an outside mechanism such as a semaphore can be used to prevent inconsistencies and collisions (critical regions), so areas of critical code can only be executed by one process at a time.

4. METHODOLOGICAL DEVELOPMENT

Abstract: *This chapter presents in detail the methodological development that includes steps and process to follow in order to configure the Kernel to run the CLUSTER_PI App.*

4.1. Introduction

In this chapter a methodological development is provided with the purpose of guide the user with knowledge and experience in embedded systems, to follow the steps that allow to configure the kernel in order to deploy CLUSTER_PI App in Minnowboard Max board. Most of this chapter is provided as recipe instructions.

Buildroot is used to achieve this objective; it provides a simple structure that allows the user to configure which packages are going to be built in the kernel. Also, the most important packages are mentioned with their respective dependencies.

This chapter is organized in subsections, starting with system attributes description, then requirements needed to run CLUSTER_PI App in x86_64 architecture and finally detailed steps to configure and cross-compile the kernel are given.

4.2. System Attributes

The following requirements need to be fulfilled in other to successfully deploy CLUSTER_PI App in x86_64 architecture.

4.2.1. Hardware Attributes

MinnowBoard Max is powered by Intel Atom Bay-Trail-I E3825 dual core processor with 2GB RAM, SATA II, USB 3.0/2.0, Gigabit Ethernet ports, etc. It supports operating systems like Debian GNU, Ubuntu, Fedora, Linux Mint, Android 4.4 (Kitkat) and 5.0 (Lollipop), Microsoft Windows 8.1 and 10, see [16] and [17].

Table 1. MinnowBoard Max technical features

Category	Feature
Core Logic	64-bit Intel®Atom™ Bay-Trail-I E3825 (dual-core, 1.33 GHz) processor
	Integrated Intel HD Graphics with Open Source hardware-accelerated drivers for Linux OS
Memory	DDR3 RAM System Memory
	2 GB
	8 MB SPI Flash System Firmware Memory
Video	Intel HD Graphics (1920x1080 max resolution)
	HDMI 1.4a (micro HDMI connector)
Audio	Digital via HDMI
	Analog available separately via MinnowBoard MAX Lure
I/O	Micro SD SDIO
	SATA2 3Gb/sec
	USB 3.0/2.0
	Serial debug via FTDI cable
	10/100/1000 Ethernet, RJ45 connector
Experimenter Features	8 x Buffered GPIO pins (2 pins support PWM)
	I2C & SPI bus
	2 x 16550 HS UARTs, one with CTS/RTS
	System Firmware Flash Programming Header
Board Dimensions	99 x 74mm (2.9 x 3.9in)
Temperature Range	0 – 40 deg C
Power	5V DC (min 2.5 A)
OS (supported)	Debian, Ubuntu, Fedora, Linux Mint
	Yocto Project Compatible, Android 4.4 and 5.0 System
	Microsoft Windows 8.1 and 10
System Boot Firmware	UEFI Firmware

The Figure 2 shows a conceptual diagram of the hardware of the main processor in the MinnowBoard MAX development board.

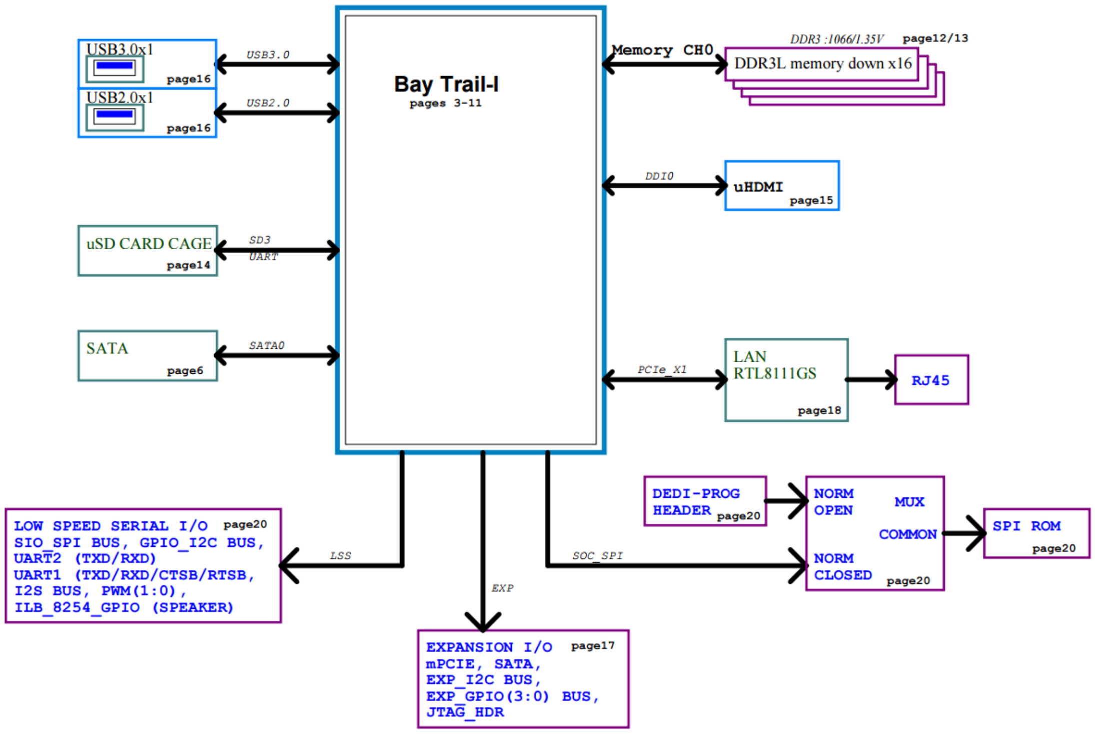


Figure 2. MinnowBoard Max block diagram

CLUSTER_PI App is shown through a LCD TFT 10.1" display connected through HDMI to MinnowBoard Max. The Figure 3 shows a conceptual diagram.

MinnowBoard Max
uHDMI to HDMI connector
Display

Figure 3. Display Panel connected to MinnowBoard Max

The Figure 4 shows the Minnowboard MAX development board and its principal pins and connectors.

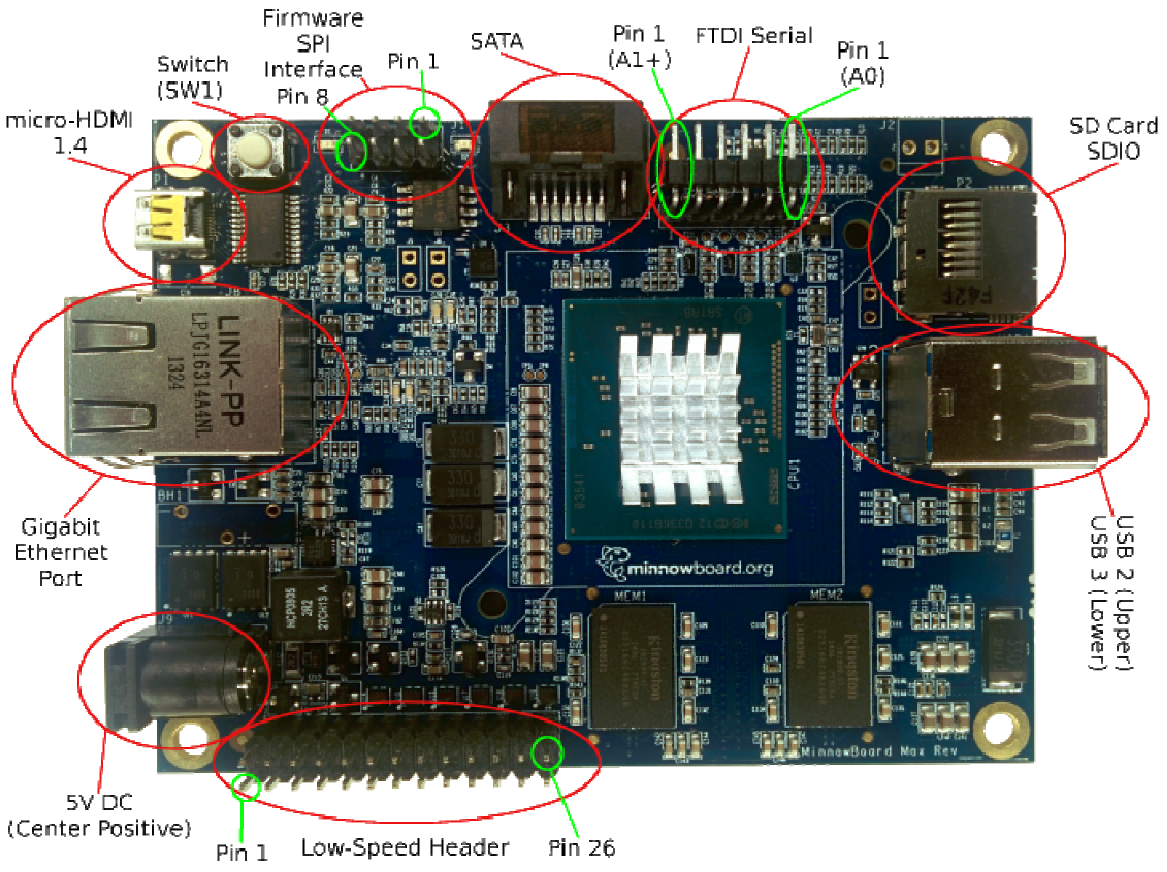


Figure 4. MinnowBoard Max board layout [16]

By the other hand, the host has the following technical features [18]:

Table 2. Host main technical features

Category	Feature
Processor	64-bit Intel® Core™ i7-3630QM CPU @ 2.40GHz
RAM	DDR3 1600 MHz SDRAM, 8.00 GB
Video	NVIDIA® GeForce® GT 635M with 2GB DDR3 VRAM
I/O	802.11 b/g/n, 10/100/1000 Ethernet, RJ45 connector
	USB 3.0
	SD card reader
OS (supported)	Kubuntu, Debian, Ubuntu, Fedora, Linux Mint, etc.
	Windows 10

4.2.2. Linux Distro and Kernel Version

The following Linux has been used in the development of this project:

- Distro installed in host is Kubuntu 14.04 LTS 64bits with kernel version 4.6.0.
- Distro installed in target is Buildroot x86_64 GNU/Linux with kernel version 4.4.0.

4.3. CLUSTER_PI App Requirements

Previously, CLUSTER_PI App, a reconfigurable automotive cluster application based on ARM architecture was deployed on Raspberry PI board. This project migrates CLUSTER_PI App to x86_64 architecture allowing scalability to computer systems, major graphic performance and future support for other automotive applications.

CLUSTER_PI App consists of two applications communicated by means of shared memory. The main application, “Dashboard”, is a GUI which displays the instrument cluster (Figure 5), several warnings caused by open door, seat belt, unlocked car, etc. can be shown to the user. Packages like X window system, QT and mesa3d and are used for graphic rendering. Also OpenSSH is needed to transfer information between target and host.

The second application, “tsw” (Figure 6), consist of HMI test client which handles an option menu which allow the user to activate/deactivate warnings available on Dashboard, it also allows to increase/decreases de speedometer and other elements in the instrument cluster. In order to allow shared memory between the two applications, kernel shall support Inter-process Communication (IPC).

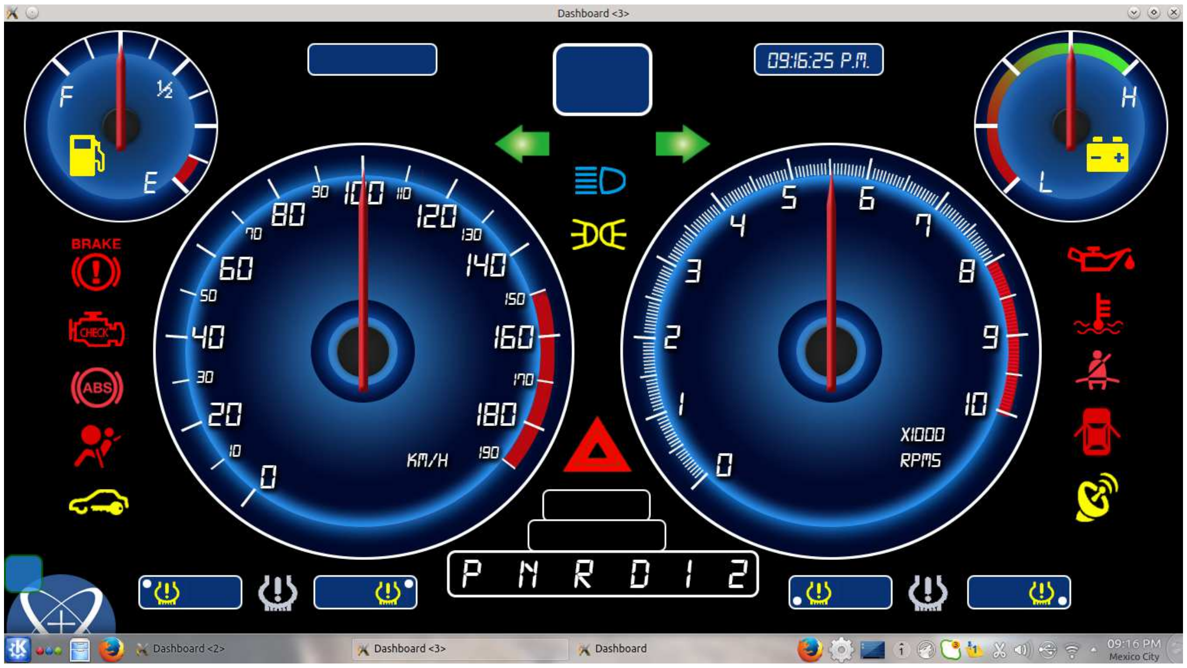
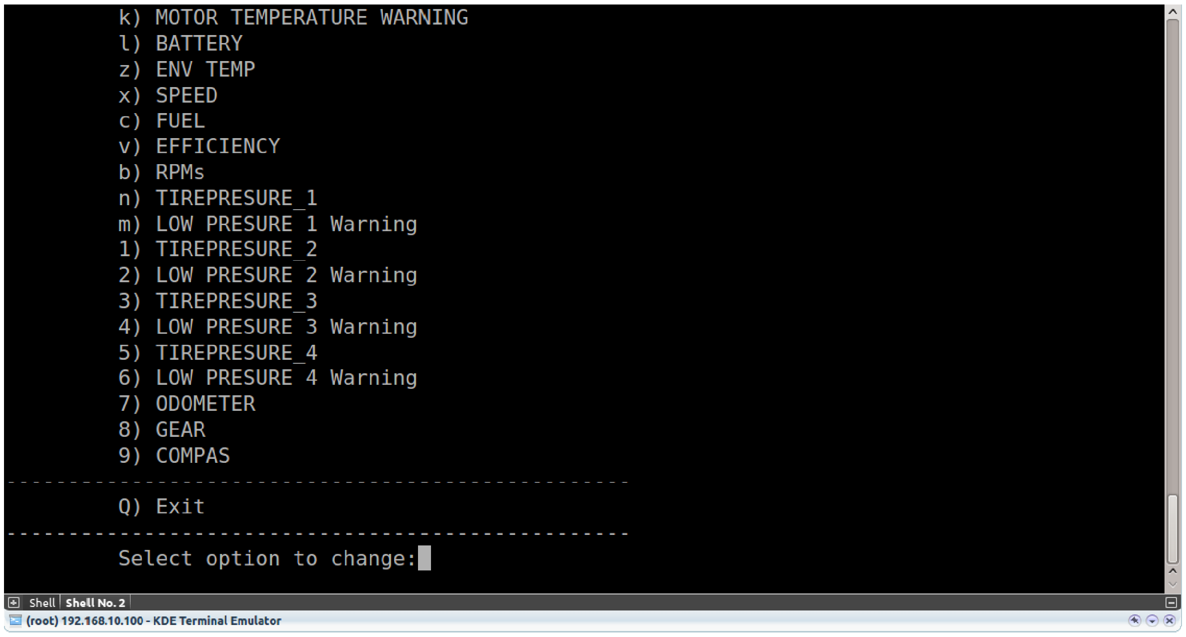


Figure 5. Dashboard with initial values and warnings activated



```
        k) MOTOR TEMPERATURE WARNING
        l) BATTERY
        z) ENV TEMP
        x) SPEED
        c) FUEL
        v) EFFICIENCY
        b) RPMs
        n) TIREPRESURE_1
        m) LOW PRESURE 1 Warning
        1) TIREPRESURE_2
        2) LOW PRESURE 2 Warning
        3) TIREPRESURE_3
        4) LOW PRESURE 3 Warning
        5) TIREPRESURE_4
        6) LOW PRESURE 4 Warning
        7) ODOMETER
        8) GEAR
        9) COMPAS
--------------------------------------------------
        Q) Exit
--------------------------------------------------
        Select option to change:
```

Figure 6. TSW application (HMI test client)

4.4. Buildroot Configuration (Host)

The objective of this project is to provide an SDK and root filesystem for the MinnowBoard Max that is lightweight and takes full advantage of the hardware available. The resulting image produced is small Linux distribution known as Cluster_Max.

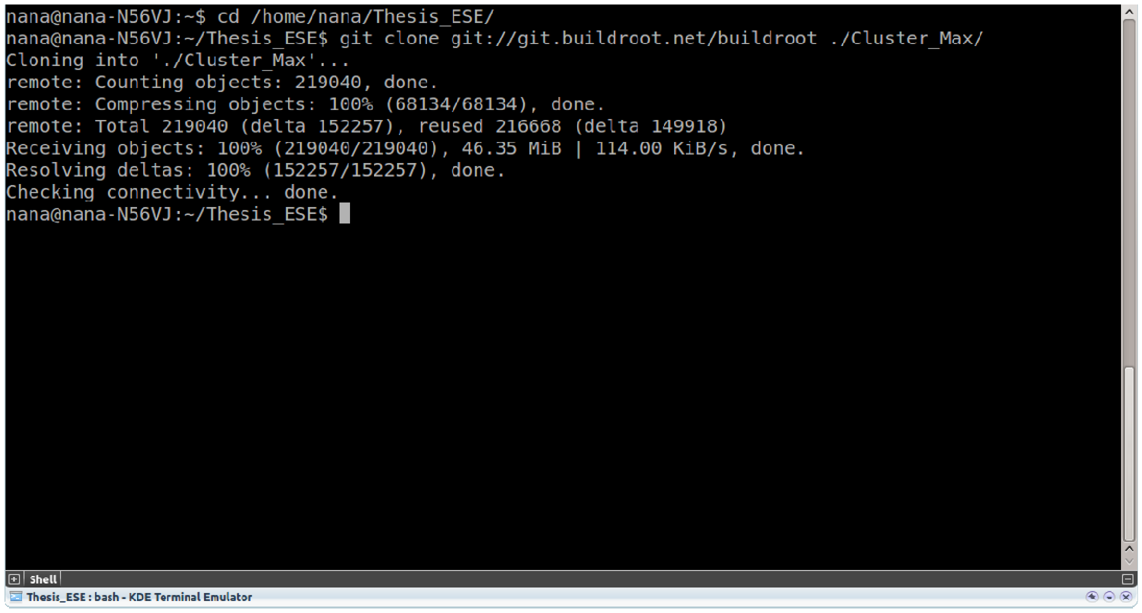
The Cluster_Max SDK provides a GCC 4.9.x toolchain for building x86_64 binaries in ELF format, as well as bootloader, kernel image, rootfs, and development sysroot for the MinnowBoard Max.

4.4.1. Getting and building the Cluster_Max SDK

First of all enter into your directory of interest:
$ cd /home/nana/Thesis_ESE/

Create Cluster_Max directory:
$ mkdir Cluster_Max

In this project Buildroot-2016.08.1 release was used, its repository can be cloned inside Cluster_Max directory by executing the following command into the Linux shell:
$ git clone git://git.buildroot.net/buildroot ./Cluster_Max/



```
nana@nana-N56VJ:~$ cd /home/nana/Thesis_ESE/
nana@nana-N56VJ:~/Thesis_ESE$ git clone git://git.buildroot.net/buildroot ./Cluster_Max/
Cloning into './Cluster_Max'...
remote: Counting objects: 219040, done.
remote: Compressing objects: 100% (68134/68134), done.
remote: Total 219040 (delta 152257), reused 216668 (delta 149918)
Receiving objects: 100% (219040/219040), 46.35 MiB | 114.00 KiB/s, done.
Resolving deltas: 100% (152257/152257), done.
Checking connectivity... done.
nana@nana-N56VJ:~/Thesis_ESE$
```

Figure 7. Cloning Buildroot

4.4.2. Adding dlt-daemon package to Buildroot

Dlt-Daemon needs to be added because it is not included by default in the Buildroot packages and is requires to compile Cluster_Pi App, first enter into the following directory:

```
$ cd /home/nana/Thesis_ESE/Cluster_Max/package/
```

Then add a new folder and call it “dlt-daemon”:

```
$ mkdir dlt-daemon
$ cd dlt-daemon
```

Create two files: “Config.in” and “dlt-daemon.mk”. First one contains help about the package that is going to show through Buildroot configuration options; second one contains dlt-daemon version and URL to be downloaded:

```
$ nano Config.in
```

A copy the following information inside the nano editor and saving it is required:

```
config BR2_PACKAGE_DLT_DAEMON
	bool "dlt-daemon"
	help
	  DLT_DAEMON receives log messages from DLT user applications and temporary storage of log messages if DLT Client isn't available. Transmit log messages to DLT Client and response to control messages.

	  DLT is a reusable open source software component for standardized logging and tracing in infotainment ECUs based on the AUTOSAR 4.0 standard.

	  http://projects.genivi.org/diagnostic-log-trace/documentation
```

```
$ nano dlt-daemon.mk
```

Copy the following information inside it:

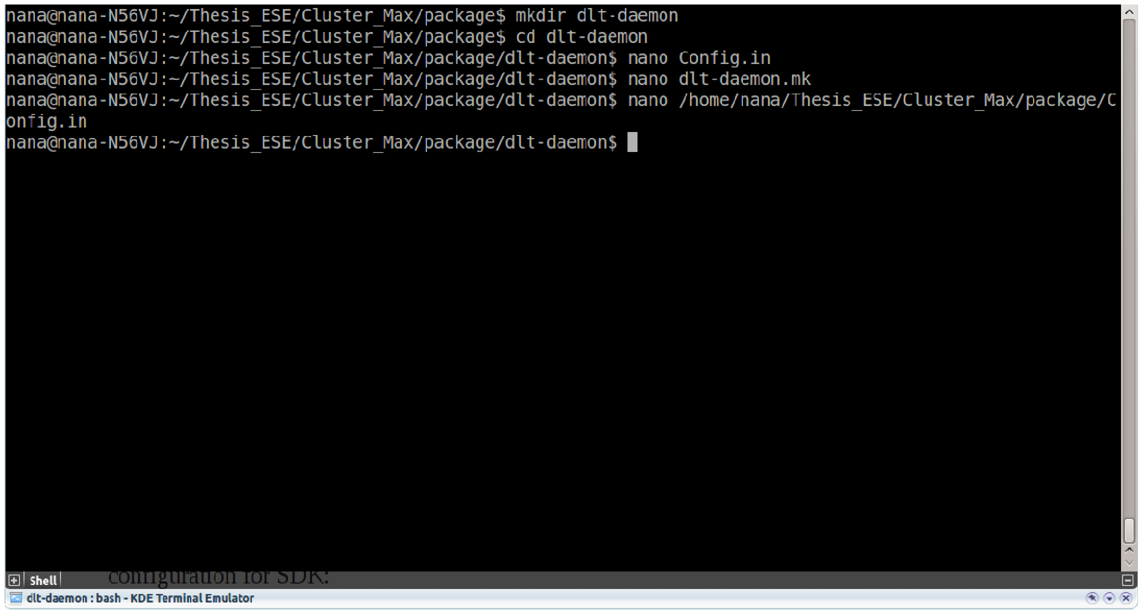
```
################################################################
#
# dlt-daemon
#
################################################################

DLT_DAEMON_VERSION = v2.13.0
DLT_DAEMON_SITE = http://git.projects.genivi.org/dlt-daemon.git
```

DLT_DAEMON_SITE_METHOD = git
DLT_DAEMON_INSTALL_STAGING = YES
DLT_DAEMON_INSTALL_TARGET = YES

$(eval $(cmake-package))

Add dlt-daemon package inside “Config.in” file located in:
$ nano /home/nana/Thesis_ESE/Cluster_Max/package/Config.in

Copy the following information inside menu "Debugging, profiling and benchmark":
source "package/dlt-daemon/Config.in"



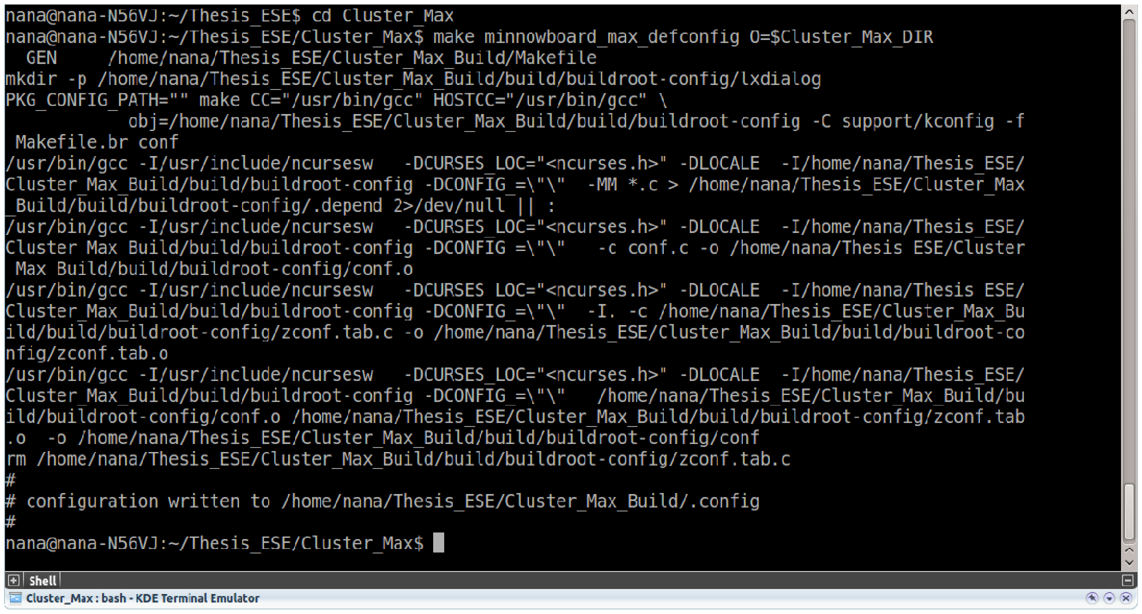
```
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max/package$ mkdir dlt-daemon
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max/package$ cd dlt-daemon
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max/package/dlt-daemon$ nano Config.in
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max/package/dlt-daemon$ nano dlt-daemon.mk
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max/package/dlt-daemon$ nano /home/nana/Thesis_ESE/Cluster_Max/package/C
onfig.in
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max/package/dlt-daemon$
```

Figure 8. Adding dlt-daemon to Buildroot

4.4.3. Generating Makefile

Create the directory where you want your SDK to be built and export its path:
$ cd /home/nana/Thesis_ESE/
$ mkdir Cluster_Max_Build
$ export Cluster_Max_DIR=/home/nana/Thesis_ESE/Cluster_Max_Build

Enter into Cluster_Max directory and generate a Makefile with Minnowboard Max default configuration for SDK:
$ cd /home/nana/Thesis_ESE/Cluster_Max/
$ make minnowboard_max_defconfig O=$Cluster_Max_DIR



```
nana@nana-N56VJ:~/Thesis_ESE$ cd Cluster_Max
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max$ make minnowboard_max_defconfig O=$Cluster_Max_DIR
  GEN     /home/nana/Thesis_ESE/Cluster_Max_Build/Makefile
mkdir -p /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config/lxdialog
PKG_CONFIG_PATH="" make CC="/usr/bin/gcc" HOSTCC="/usr/bin/gcc" \
            obj=/home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config -C support/kconfig -f
 Makefile.br conf
/usr/bin/gcc -I/usr/include/ncursesw   -DCURSES_LOC="<ncurses.h>" -DLOCALE  -I/home/nana/Thesis_ESE/
Cluster_Max_Build/build/buildroot-config -DCONFIG_=\"\"  -MM *.c > /home/nana/Thesis_ESE/Cluster_Max
_Build/build/buildroot-config/.depend 2>/dev/null || :
/usr/bin/gcc -I/usr/include/ncursesw   -DCURSES_LOC="<ncurses.h>" -DLOCALE  -I/home/nana/Thesis_ESE/
Cluster_Max_Build/build/buildroot-config -DCONFIG_=\"\"   -c conf.c -o /home/nana/Thesis_ESE/Cluster
_Max_Build/build/buildroot-config/conf.o
/usr/bin/gcc -I/usr/include/ncursesw   -DCURSES_LOC="<ncurses.h>" -DLOCALE  -I/home/nana/Thesis_ESE/
Cluster_Max_Build/build/buildroot-config -DCONFIG_=\"\"  -I. -c /home/nana/Thesis_ESE/Cluster_Max_Bu
ild/build/buildroot-config/zconf.tab.c -o /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-co
nfig/zconf.tab.o
/usr/bin/gcc -I/usr/include/ncursesw   -DCURSES_LOC="<ncurses.h>" -DLOCALE  -I/home/nana/Thesis_ESE/
Cluster_Max_Build/build/buildroot-config -DCONFIG_=\"\"   /home/nana/Thesis_ESE/Cluster_Max_Build/bu
ild/buildroot-config/conf.o /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config/zconf.tab
.o  -o /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config/conf
rm /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config/zconf.tab.c
#
# configuration written to /home/nana/Thesis_ESE/Cluster_Max_Build/.config
#
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max$
```

Figure 9. MinnowBoard Max default configuration

4.4.4. Running Makefile

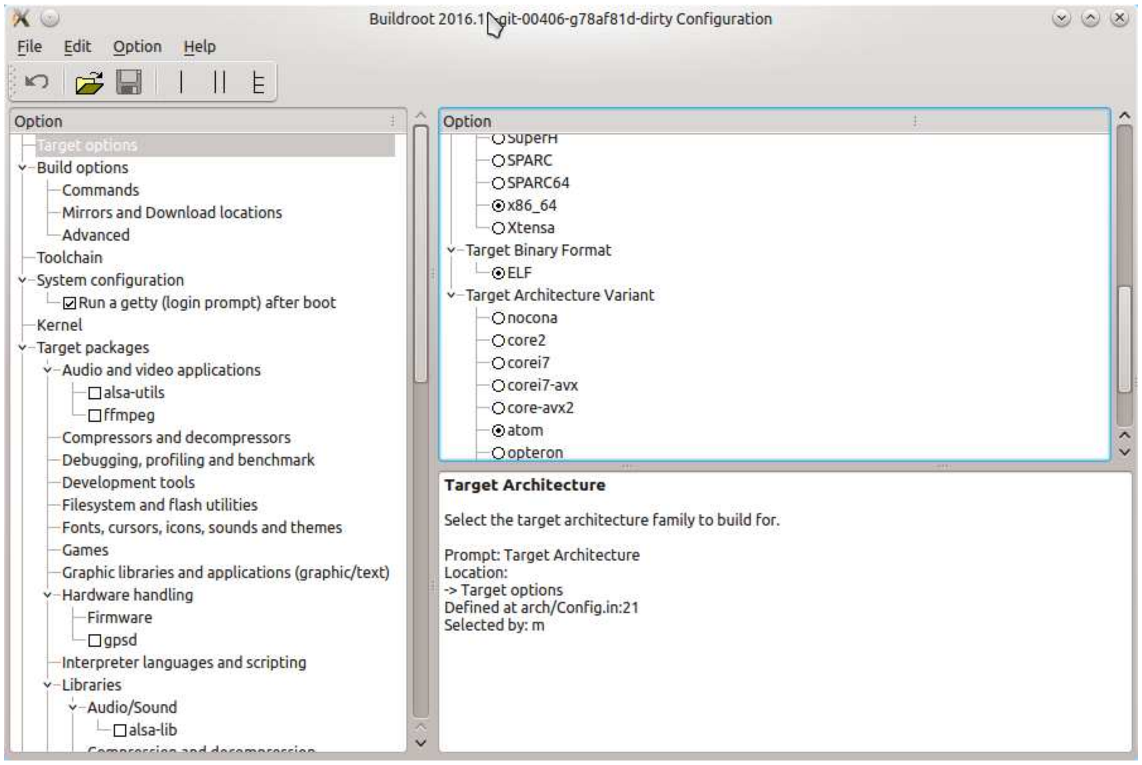
Change to your SDK directory and start Buildroot configuration:

```
$ cd $Cluster_Max_DIR
$ make xconfig
```

Note that some configurations are already selected by default, since minnowboard_max_defconfig was chosen to create the initial ".config" file. It contains all the necessary information that is going to be compiled when "make" command is executed. In the following subsections most relevant packages and configurations are described.

4.4.4.1. Target options

- BR2_x86_64
x86-64 is an extension of the x86 instruction set (Intel i386 architecture compatible microprocessor).
- BR2_BINFMT_ELF
ELF (Executable and Linkable Format) is a format for libraries and executables used across different architectures and operating systems.
- BR2_x86_atom: Atom is selected as the architecture variant.



Buildroot 2016.11-git-00406-g78af81d-dirty Configuration

```
Option
Target options
Build options
    Commands
    Mirrors and Download locations
    Advanced
Toolchain
System configuration
    ☑ Run a getty (login prompt) after boot
Kernel
Target packages
    Audio and video applications
        ☐ alsa-utils
        ☐ ffmpeg
    Compressors and decompressors
    Debugging, profiling and benchmark
    Development tools
    Filesystem and flash utilities
    Fonts, cursors, icons, sounds and themes
    Games
    Graphic libraries and applications (graphic/text)
    Hardware handling
        Firmware
        ☐ gpsd
    Interpreter languages and scripting
    Libraries
        Audio/Sound
            ☐ alsa-lib
```

```
Option
    ○ SuperH
    ○ SPARC
    ○ SPARC64
    ● x86_64
    ○ Xtensa
Target Binary Format
    ● ELF
Target Architecture Variant
    ○ nocona
    ○ core2
    ○ corei7
    ○ corei7-avx
    ○ core-avx2
    ● atom
    ○ opteron
```

Target Architecture

Select the target architecture family to build for.

Prompt: Target Architecture
Location:
-> Target options
Defined at arch/Config.in:21
Selected by: m

Figure 10. Target configuration

4.4.4.2. Build Options

- BR2_STRIP_strip

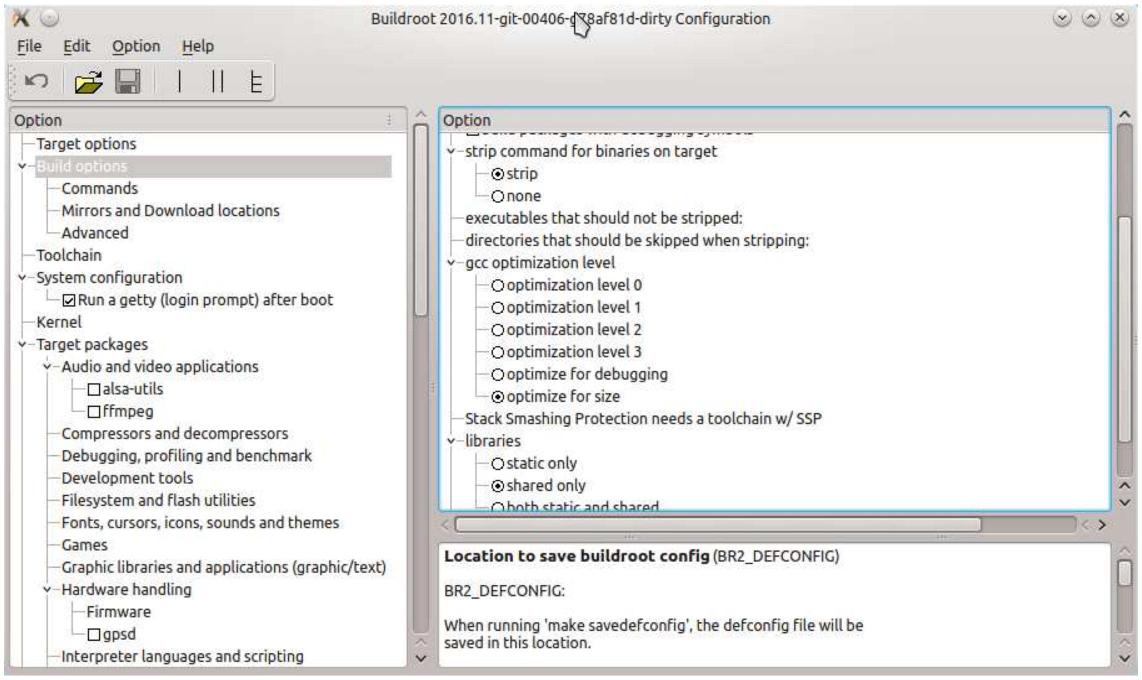
Binaries and libraries in the target filesystem will be stripped using the normal 'strip' command. This allows to save space, mainly by removing debugging symbols.

- BR2_OPTIMIZE_S

Optimize for size.

- BR2_SHARED_LIBS

Build and use only shared libraries. This is the recommended solution as it saves space and builds time.


Buildroot 2016.11-git-00406-g78af81d-dirty Configuration
File
Edit
Option
Help
Option
Target options
Build options
Commands
Mirrors and Download locations
Advanced
Toolchain
System configuration
Run a getty (login prompt) after boot
Kernel
Target packages
Audio and video applications
alsa-utils
ffmpeg
Compressors and decompressors
Debugging, profiling and benchmark
Development tools
Filesystem and flash utilities
Fonts, cursors, icons, sounds and themes
Games
Graphic libraries and applications (graphic/text)
Hardware handling
Firmware
gpsd
Interpreter languages and scripting
Option
strip command for binaries on target
strip
none
executables that should not be stripped:
directories that should be skipped when stripping:
gcc optimization level
optimization level 0
optimization level 1
optimization level 2
optimization level 3
optimize for debugging
optimize for size
Stack Smashing Protection needs a toolchain w/ SSP
libraries
static only
shared only
both static and shared

Location to save buildroot config (BR2_DEFCONFIG)

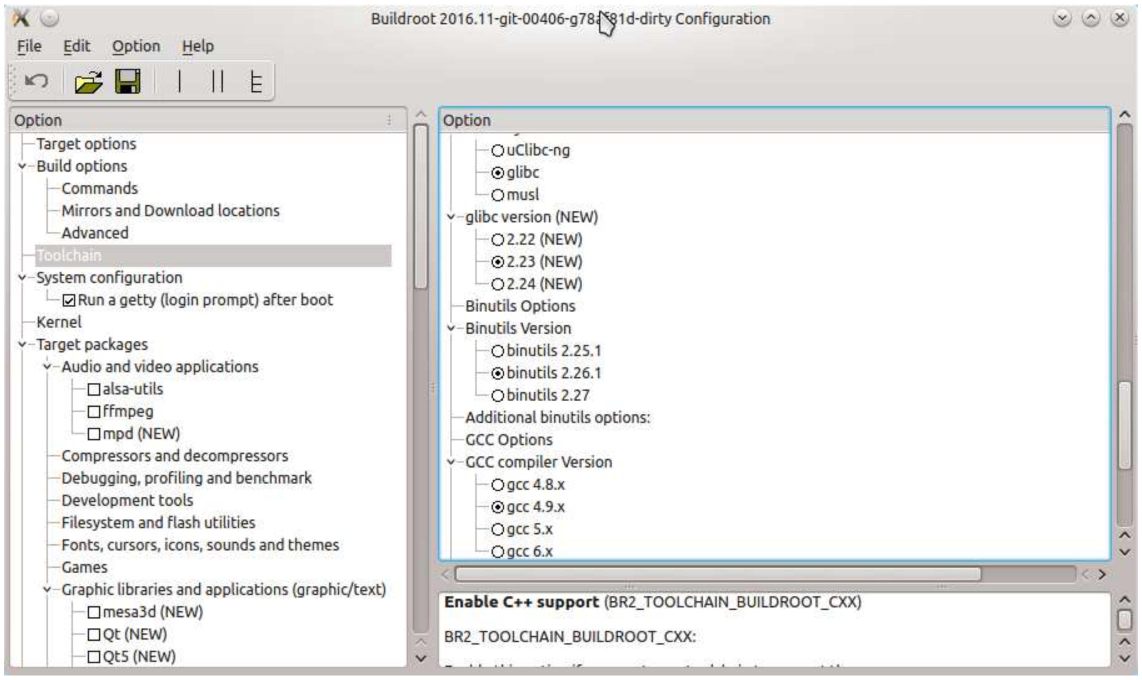
BR2_DEFCONFIG:

When running 'make savedefconfig', the defconfig file will be saved in this location.

Figure 11. Build configuration

4.4.4.3. Toolchain

- BR2_TOOLCHAIN_BUILDROOT
Buildroot toolchain is selected.
- BR2_KERNEL_HEADERS_AS_KERNEL
Kernel headers are the same version as the kernel being built.
- BR2_PACKAGE_HOST_LINUX_HEADERS_CUSTOM_4_6
Custom kernel header series.
- BR2_TOOLCHAIN_BUILDROOT_GLIBC
This option selects glibc as the C library for the cross-compilation toolchain.
- BR2_GLIBC_VERSION_2_23
Glibc version 2.23
- BR2_BINUTILS_VERSION_2_26_X
The GNU Binutils (v2.26.1) are a collection of binary tools. The main ones are: **ld** - the GNU linker, and **as** - the GNU assembler.
- BR2_GCC_VERSION_4_9_X
GCC compiler version 4.9.X
- BR2_TOOLCHAIN_BUILDROOT_CXX
This option is enabled in order to have C++ language support in the toolchain, also C++ libraries are going to be installed on your target system.



```
Target options
Build options
    Commands
    Mirrors and Download locations
    Advanced
Toolchain
System configuration
    [✓] Run a getty (login prompt) after boot
Kernel
Target packages
    Audio and video applications
        ☐ alsa-utils
        ☐ ffmpeg
        ☐ mpd (NEW)
    Compressors and decompressors
    Debugging, profiling and benchmark
    Development tools
    Filesystem and flash utilities
    Fonts, cursors, icons, sounds and themes
    Games
    Graphic libraries and applications (graphic/text)
        ☐ mesa3d (NEW)
        ☐ Qt (NEW)
        ☐ Qt5 (NEW)
```

```
    ○ uClibc-ng
    ● glibc
    ○ musl
glibc version (NEW)
    ○ 2.22 (NEW)
    ● 2.23 (NEW)
    ○ 2.24 (NEW)
Binutils Options
Binutils Version
    ○ binutils 2.25.1
    ● binutils 2.26.1
    ○ binutils 2.27
Additional binutils options:
GCC Options
GCC compiler Version
    ○ gcc 4.8.x
    ● gcc 4.9.x
    ○ gcc 5.x
    ○ gcc 6.x
```

Enable C++ support (BR2_TOOLCHAIN_BUILDROOT_CXX)

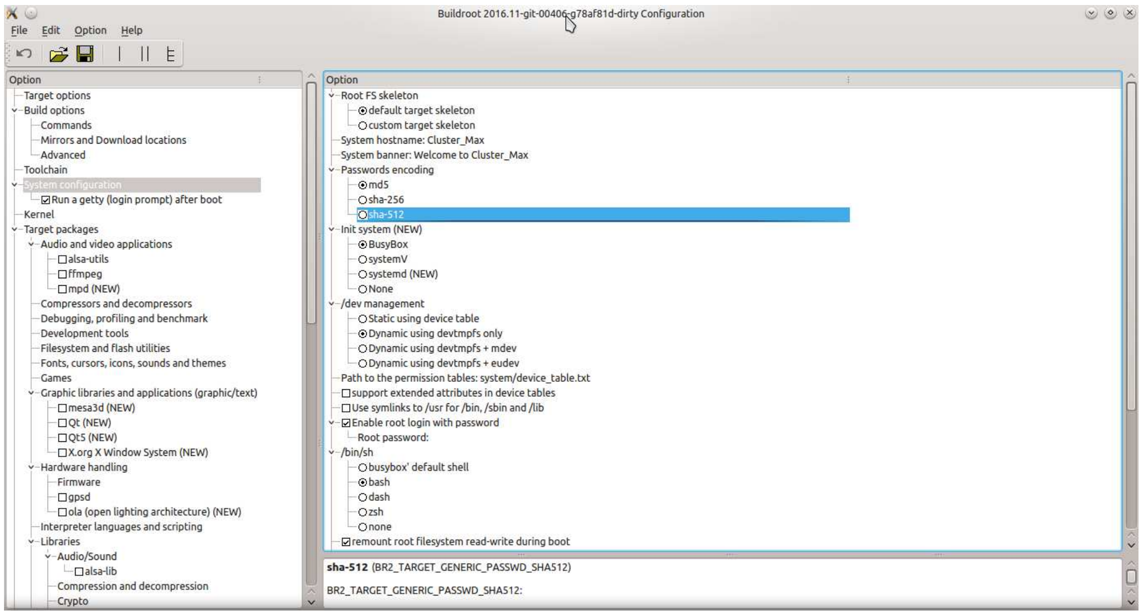
BR2_TOOLCHAIN_BUILDROOT_CXX:

Figure 12. Toolchain configuration

4.4.4.4. System Configuration

- BR2_ROOTFS_SKELETON_DEFAULT
Use default target skeleton.
- BR2_TARGET_GENERIC_HOSTNAME [=Cluster_Max]
System hostname to be stored in /etc/hostname.
- BR2_TARGET_GENERIC_ISSUE [=Welcome to Cluster_Max]
System banner (/etc/issue) to be displayed at login.
- BR2_TARGET_GENERIC_PASSWD_MD5
Use MD5 to encode passwords.
- BR2_INIT_BUSYBOX
Busybox is used to init system.
- BR2_ROOTFS_DEVICE_CREATION_DYNAMIC_DEVTMPFS
/dev management dynamic using devtmpfs only.
- BR2_TARGET_ENABLE_ROOT_LOGIN
Allow root to log in with a password.
- BR2_SYSTEM_BIN_SH_BASH
/bin/sh bash (Bourne again shell) is selected as command processor.
Depends on: BR2_PACKAGE_BUSYBOX_SHOW_OTHERS (Section 4.4.4.6).
- BR2_TARGET_GENERIC_REMOUNT_ROOTFS_RW

The root filesystem is typically mounted read-only at boot. By default, Buildroot remounts it in read-write mode early during the boot process.
- BR2_ENABLE_LOCALE_PURGE
Explicitly specify what locales to install on target.

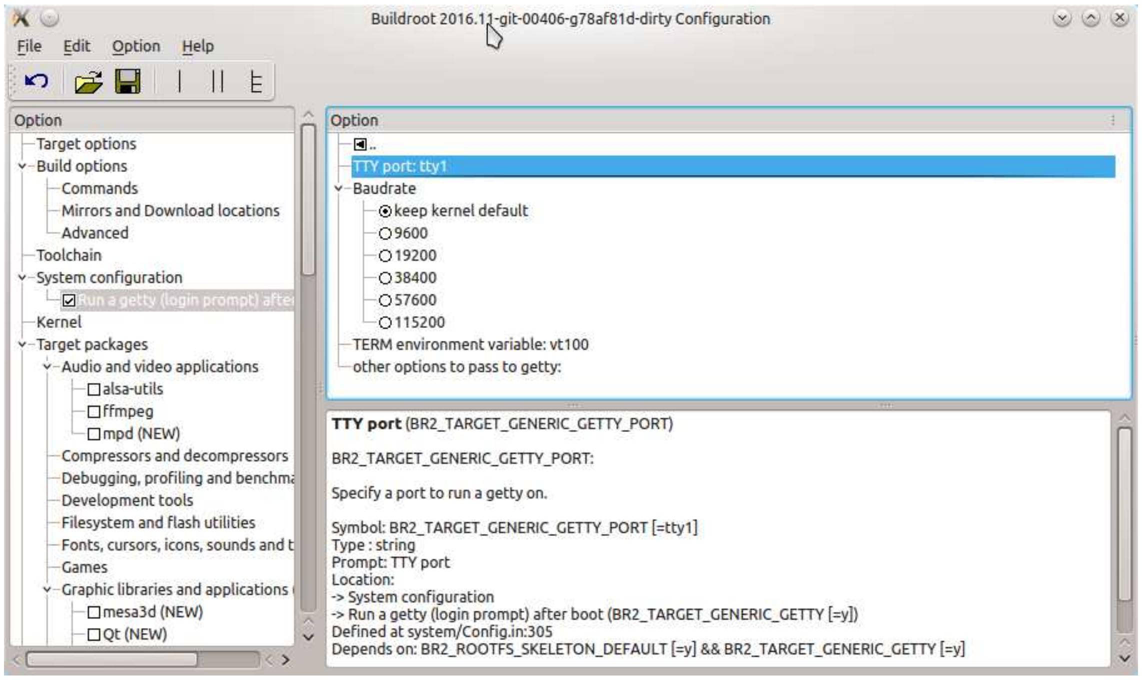


```
Option
 Target options
 Build options
   Commands
   Mirrors and Download locations
   Advanced
 Toolchain
 System configuration
   ☑ Run a getty (login prompt) after boot
 Kernel
 Target packages
   Audio and video applications
     ☐ alsa-utils
     ☐ ffmpeg
     ☐ mpd (NEW)
   Compressors and decompressors
   Debugging, profiling and benchmark
   Development tools
   Filesystem and flash utilities
   Fonts, cursors, icons, sounds and themes
   Games
   Graphic libraries and applications (graphic/text)
     ☐ mesa3d (NEW)
     ☐ Qt (NEW)
     ☐ Qt5 (NEW)
     ☐ X.org X Window System (NEW)
   Hardware handling
     Firmware
     ☐ gpsd
     ☐ ola (open lighting architecture) (NEW)
   Interpreter languages and scripting
   Libraries
     Audio/Sound
       ☐ alsa-lib
     Compression and decompression
     Crypto
```

```
Option
 Root FS skeleton
   ● default target skeleton
   ○ custom target skeleton
 System hostname: Cluster_Max
 System banner: Welcome to Cluster_Max
 Passwords encoding
   ● md5
   ○ sha-256
   ○ sha-512
 Init system (NEW)
   ● BusyBox
   ○ systemV
   ○ systemd (NEW)
   ○ None
 /dev management
   ○ Static using device table
   ● Dynamic using devtmpfs only
   ○ Dynamic using devtmpfs + mdev
   ○ Dynamic using devtmpfs + eudev
 Path to the permission tables: system/device_table.txt
 ☐ support extended attributes in device tables
 ☐ Use symlinks to /usr for /bin, /sbin and /lib
 ☑ Enable root login with password
   Root password:
 /bin/sh
   ○ busybox' default shell
   ● bash
   ○ dash
   ○ zsh
   ○ none
 ☑ remount root filesystem read-write during boot

sha-512 (BR2_TARGET_GENERIC_PASSWD_SHA512)
BR2_TARGET_GENERIC_PASSWD_SHA512:
```

Figure 13. System configuration

- BR2_TARGET_GENERIC_GETTY_PORT [=tty1]
tty1 is specified as port to run a getty on.
- BR2_TARGET_GENERIC_GETTY_BAUDRATE_KEEP
Keep kernel default baud rate.


Buildroot 2016.11-git-00406-g78af81d-dirty Configuration
File
Edit
Option
Help
Option
Target options
Build options
Commands
Mirrors and Download locations
Advanced
Toolchain
System configuration
Run a getty (login prompt) after
Kernel
Target packages
Audio and video applications
alsa-utils
ffmpeg
mpd (NEW)
Compressors and decompressors
Debugging, profiling and benchma
Development tools
Filesystem and flash utilities
Fonts, cursors, icons, sounds and t
Games
Graphic libraries and applications
mesa3d (NEW)
Qt (NEW)
Option

TTY port: tty1
Baudrate
- [✓ keep kernel default]
- 9600
- 19200
- 38400
- 57600
- 115200

TERM environment variable: vt100
other options to pass to getty:

TTY port (BR2_TARGET_GENERIC_GETTY_PORT)

BR2_TARGET_GENERIC_GETTY_PORT:

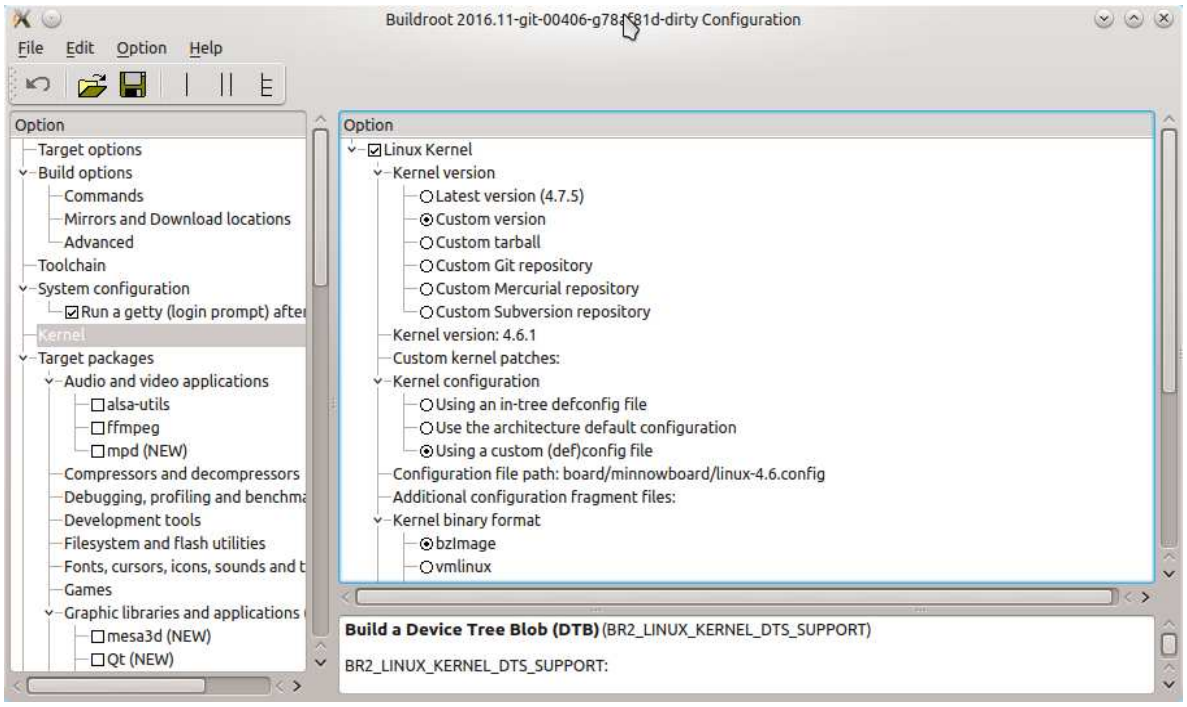
Specify a port to run a getty on.

Symbol: BR2_TARGET_GENERIC_GETTY_PORT [=tty1]
Type : string
Prompt: TTY port
Location:
-> System configuration
-> Run a getty (login prompt) after boot (BR2_TARGET_GENERIC_GETTY [=y])
Defined at system/Config.in:305
Depends on: BR2_ROOTFS_SKELETON_DEFAULT [=y] && BR2_TARGET_GENERIC_GETTY [=y]

Figure 14. TTY port configuration

4.4.4.5. Kernel

- BR2_LINUX_KERNEL_CUSTOM_VERSION
This option allows to use a specific official version from kernel.org.
- BR2_LINUX_KERNEL_CUSTOM_VERSION_VALUE [=4.6.1]
Kernel version used is 4.6.1.
- BR2_LINUX_KERNEL_USE_CUSTOM_CONFIG
Using a custom (def) config file.
- BR2_LINUX_KERNEL_CUSTOM_CONFIG_FILE [=board/minnowboard/linux-4.6.config]
Path to the kernel configuration file.
- BR2_LINUX_KERNEL_BZIMAGE
Kernel binary format is bzimage.
- BR2_LINUX_KERNEL_GZIP
gzip kernel compression format.


File
Edit
Option
Help
Option
Target options
Build options
Commands
Mirrors and Download locations
Advanced
Toolchain
System configuration
Run a getty (login prompt) after
Kernel
Target packages
Audio and video applications
alsa-utils
ffmpeg
mpd (NEW)
Compressors and decompressors
Debugging, profiling and benchma
Development tools
Filesystem and flash utilities
Fonts, cursors, icons, sounds and t
Games
Graphic libraries and applications
mesa3d (NEW)
Qt (NEW)
Option

```
Buildroot 2016.11-git-00406-g78??81d-dirty Configuration

☑ Linux Kernel
  Kernel version
    ○ Latest version (4.7.5)
    ◉ Custom version
    ○ Custom tarball
    ○ Custom Git repository
    ○ Custom Mercurial repository
    ○ Custom Subversion repository
  Kernel version: 4.6.1
  Custom kernel patches:
  Kernel configuration
    ○ Using an in-tree defconfig file
    ○ Use the architecture default configuration
    ◉ Using a custom (def)config file
  Configuration file path: board/minnowboard/linux-4.6.config
  Additional configuration fragment files:
  Kernel binary format
    ◉ bzImage
    ○ vmlinux

Build a Device Tree Blob (DTB) (BR2_LINUX_KERNEL_DTS_SUPPORT)

BR2_LINUX_KERNEL_DTS_SUPPORT:
```

Figure 15. Kernel configuration

4.4.4.6. Target Packages

- BR2_PACKAGE_BUSYBOX_SHOW_OTHERS
Show packages in menuconfig that are also provided by busybox.

Development tools

- BR2_PACKAGE_BINUTILS
Install binutils on the target (this includes zlib).

Hardware handling

- BR2_PACKAGE_DBUS
The D-Bus message bus system.

Text and terminal handling

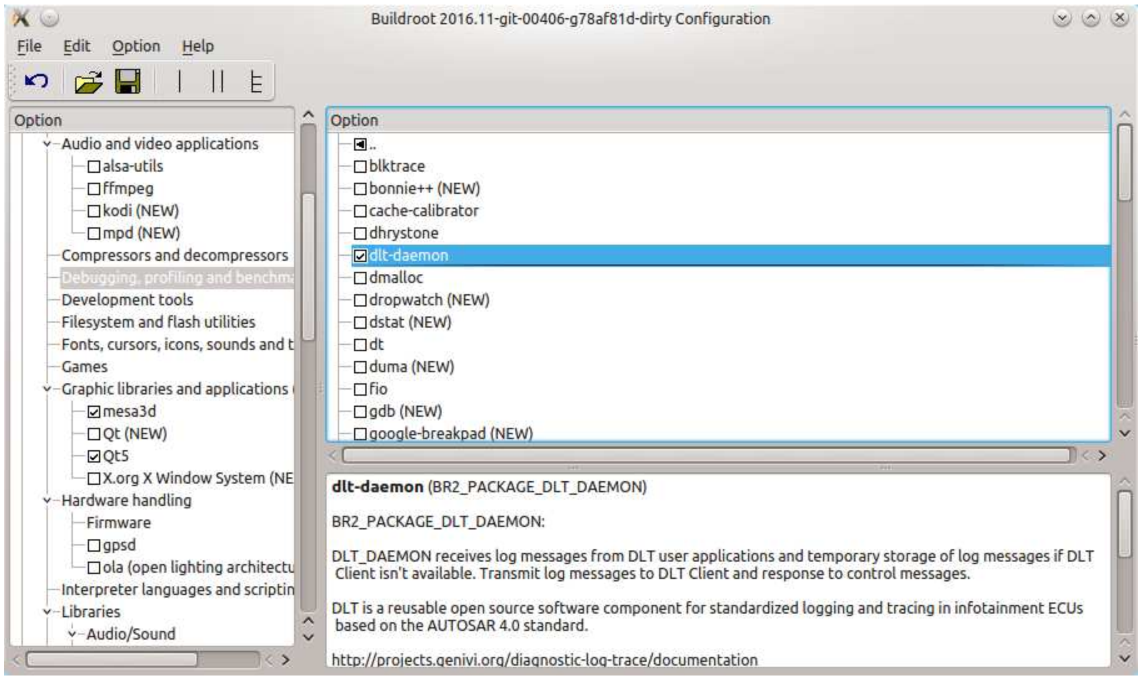
- BR2_PACKAGE_NANO
A nice ncurses-based editor. Started out as a clone of pico. Great editor for new users.
-BR2_PACKAGE_NANO_TINY

Disable all features for the sake of size.

Debugging, profiling and benchmark

- BR2_PACKAGE_DLT_DAEMON

DLT_DAEMON receives log messages from DLT user applications and temporary storage of log messages if DLT Client isn't available. Transmit log messages to DLT Client and response to control messages. This package was added to Buildroot as explained in Section 4.4.2.



```
Audio and video applications
  ☐ alsa-utils
  ☐ ffmpeg
  ☐ kodi (NEW)
  ☐ mpd (NEW)
Compressors and decompressors
Debugging, profiling and benchma
Development tools
Filesystem and flash utilities
Fonts, cursors, icons, sounds and t
Games
Graphic libraries and applications (
  ☑ mesa3d
  ☐ Qt (NEW)
  ☑ Qt5
  ☐ X.org X Window System (NE
Hardware handling
  Firmware
  ☐ gpsd
  ☐ ola (open lighting architectu
Interpreter languages and scriptin
Libraries
  Audio/Sound
```

```
..
☐ blktrace
☐ bonnie++ (NEW)
☐ cache-calibrator
☐ dhrystone
☑ dlt-daemon
☐ dmalloc
☐ dropwatch (NEW)
☐ dstat (NEW)
☐ dt
☐ duma (NEW)
☐ fio
☐ gdb (NEW)
☐ google-breakpad (NEW)
```

dlt-daemon (BR2_PACKAGE_DLT_DAEMON)

BR2_PACKAGE_DLT_DAEMON:

DLT_DAEMON receives log messages from DLT user applications and temporary storage of log messages if DLT Client isn't available. Transmit log messages to DLT Client and response to control messages.

DLT is a reusable open source software component for standardized logging and tracing in infotainment ECUs based on the AUTOSAR 4.0 standard.

http://projects.genivi.org/diagnostic-log-trace/documentation

Figure 16. dlt-daemon package selection

Graphic libraries and applications (graphic/text)

- BR2_PACKAGE_MESA3D

Mesa 3D, an open-source implementation of the OpenGL specification.

- BR2_PACKAGE_MESA3D_GALLIUM_DRIVER_SWRAST

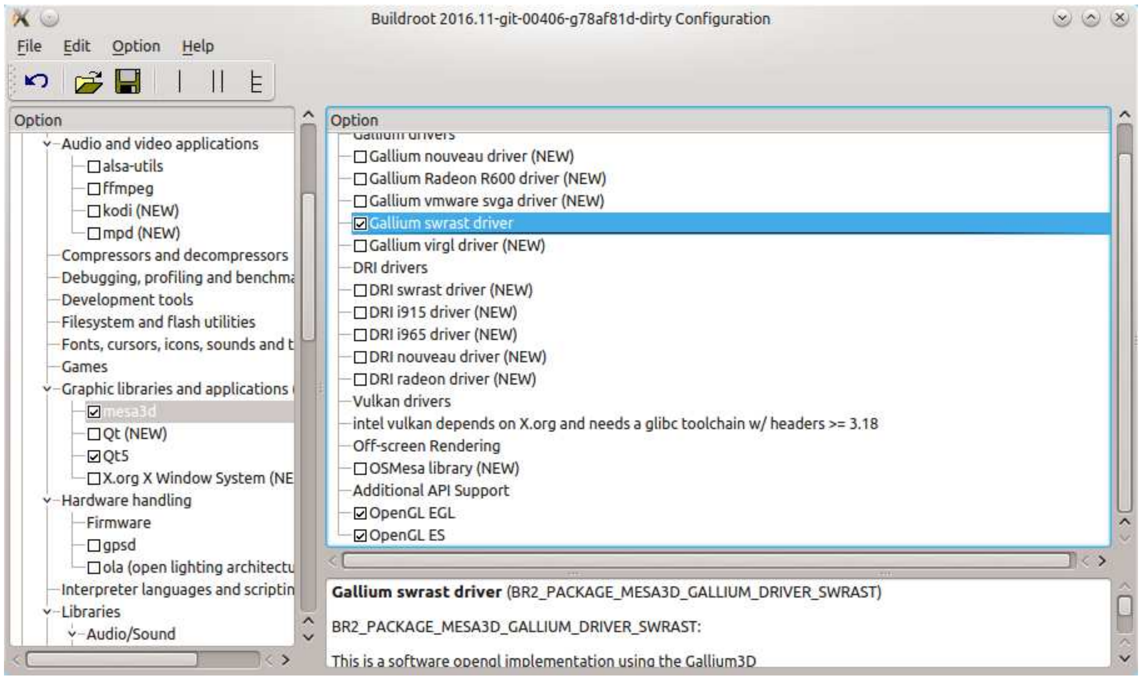
This is a software opengl implementation using the Gallium3D infrastructure.

- BR2_PACKAGE_MESA3D_OPENGL_EGL

Use the Khronos EGL APIs. EGL is a window manager for OpenGL applications similar to GLX, for X, and WGL, for Windows.

- BR2_PACKAGE_MESA3D_OPENGL_ES

Use the Khronos OpenGL ES APIs. This is commonly used on embedded systems and represents a subset of the OpenGL API.



```
Audio and video applications
    ☐ alsa-utils
    ☐ ffmpeg
    ☐ kodi (NEW)
    ☐ mpd (NEW)
Compressors and decompressors
Debugging, profiling and benchma
Development tools
Filesystem and flash utilities
Fonts, cursors, icons, sounds and t
Games
Graphic libraries and applications (
    ☑ mesa3d
    ☐ Qt (NEW)
    ☑ Qt5
    ☐ X.org X Window System (NE
Hardware handling
    Firmware
    ☐ gpsd
    ☐ ola (open lighting architectu
Interpreter languages and scriptin
Libraries
    Audio/Sound
```

```
☐ Gallium nouveau driver (NEW)
☐ Gallium Radeon R600 driver (NEW)
☐ Gallium vmware svga driver (NEW)
☑ Gallium swrast driver
☐ Gallium virgl driver (NEW)
DRI drivers
☐ DRI swrast driver (NEW)
☐ DRI i915 driver (NEW)
☐ DRI i965 driver (NEW)
☐ DRI nouveau driver (NEW)
☐ DRI radeon driver (NEW)
Vulkan drivers
intel vulkan depends on X.org and needs a glibc toolchain w/ headers >= 3.18
Off-screen Rendering
☐ OSMesa library (NEW)
Additional API Support
☑ OpenGL EGL
☑ OpenGL ES
```

Gallium swrast driver (BR2_PACKAGE_MESA3D_GALLIUM_DRIVER_SWRAST)

BR2_PACKAGE_MESA3D_GALLIUM_DRIVER_SWRAST:

This is a software opengl implementation using the Gallium3D

Figure 17. Mesa3D configuration

- BR2_PACKAGE_QT5

This option enables the Qt5 framework. Sub-options allow to select which modules should be built.

- BR2_PACKAGE_QT5BASE

Qt is a cross-platform application and UI framework for developers using C++. This package corresponds to the qt5base module, which contains the base Qt libraries: QtCore, QtNetwork, QtGui, QtWidgets, etc.

- BR2_PACKAGE_QT5BASE_LICENSE_APPROVED

Select this if you approve one of the available free licenses for the Qt5 library. By doing this you will not be asked while the library is compiled.

- BR2_PACKAGE_QT5BASE_SQLITE_NONE

Do not compile any kind of SQLite support.

- BR2_PACKAGE_QT5BASE_OPENGL_ES2

Use OpenGL ES 2.0 and later versions.

BR2_PACKAGE_QT5BASE_OPENGL_LIB:

This option enables the Qt5OpenGL library. This library includes OpenGL support classes provided to ease porting from Qt 4.x.

- BR2_PACKAGE_QT5BASE_EGLFS

Enable eglfs support.

- BR2_PACKAGE_QT5BASE_FONTCONFIG:

This option enables Fontconfig and Freetype support using the system fontconfig and freetype2 libraries.

- BR2_PACKAGE_QT5BASE_GIF:

This compiles and installs the plug-in for GIF reading support.

-BR2_PACKAGE_QT5BASE_JPEG:

This option enables JPEG support using the system libjpeg library.

BR2_PACKAGE_QT5BASE_PNG:

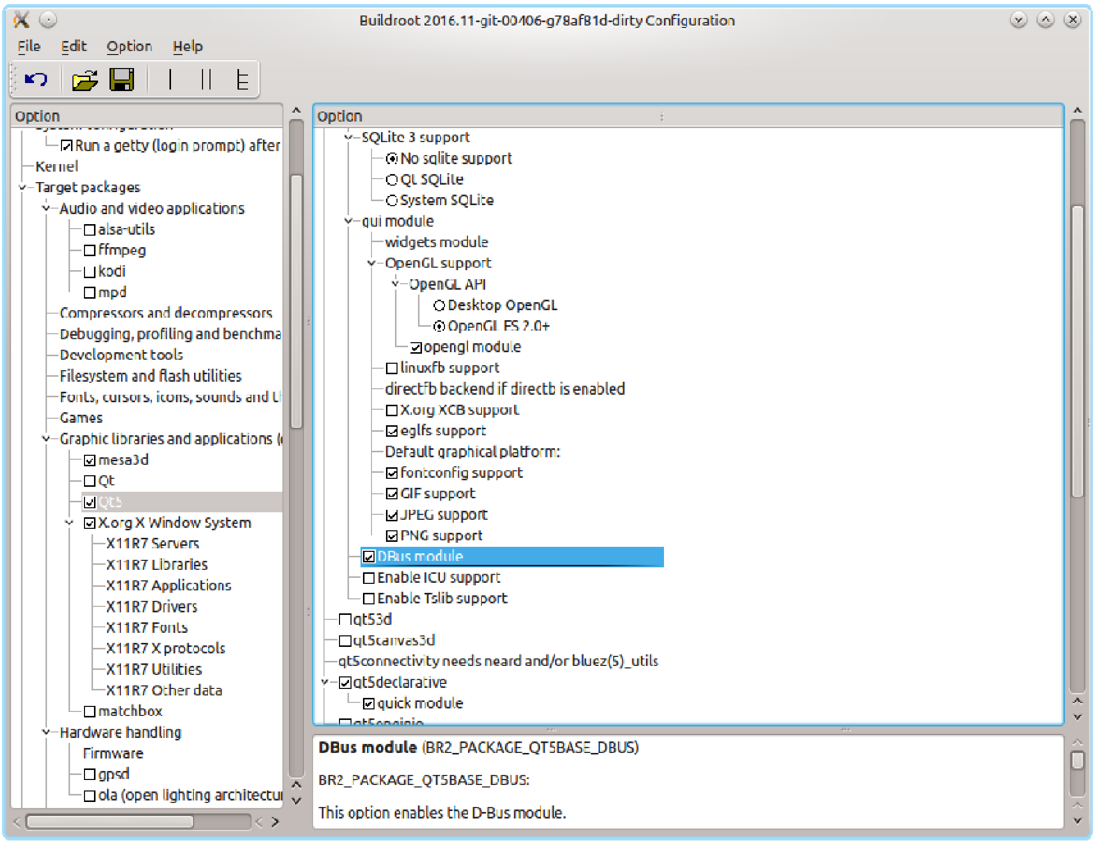
This option enables PNG support using the system libpng library.

BR2_PACKAGE_QT5BASE_DBUS:

This option enables the D-Bus module.

- BR2_PACKAGE_QT5DECLARATIVE

- BR2_PACKAGE_QT5DECLARATIVE_QUICK



```
☑ Run a getty (login prompt) after
Kernel
Target packages
  Audio and video applications
    ☐ alsa-utils
    ☐ ffmpeg
    ☐ kodi
    ☐ mpd
  Compressors and decompressors
  Debugging, profiling and benchma
  Development tools
  Filesystem and flash utilities
  Fonts, cursors, icons, sounds and U
  Games
  Graphic libraries and applications (
    ☑ mesa3d
    ☐ Qt
    ☑ Qt5
    ☑ X.org X Window System
      X11R7 Servers
      X11R7 Libraries
      X11R7 Applications
      X11R7 Drivers
      X11R7 Fonts
      X11R7 X protocols
      X11R7 Utilities
      X11R7 Other data
    ☐ matchbox
  Hardware handling
    Firmware
    ☐ gpsd
    ☐ ola (open lighting architectu
```

```
  SQLite 3 support
    ◉ No sqlite support
    ○ QL SQLite
    ○ System SQLite
  gui module
    widgets module
    OpenGL support
      OpenGL API
        ○ Desktop OpenGL
        ◉ OpenGL ES 2.0+
      ☑ opengl module
    ☐ linuxfb support
    directfb backend if directb is enabled
    ☐ X.org XCB support
    ☑ eglfs support
    Default graphical platform:
    ☑ fontconfig support
    ☑ GIF support
    ☑ JPEG support
    ☑ PNG support
  ☑ DBus module
  ☐ Enable ICU support
  ☐ Enable Tslib support
☐ qt53d
☐ qt5canvas3d
qt5connectivity needs neard and/or bluez(5)_utils
☑ qt5declarative
  ☑ quick module
```

DBus module (BR2_PACKAGE_QT5BASE_DBUS)

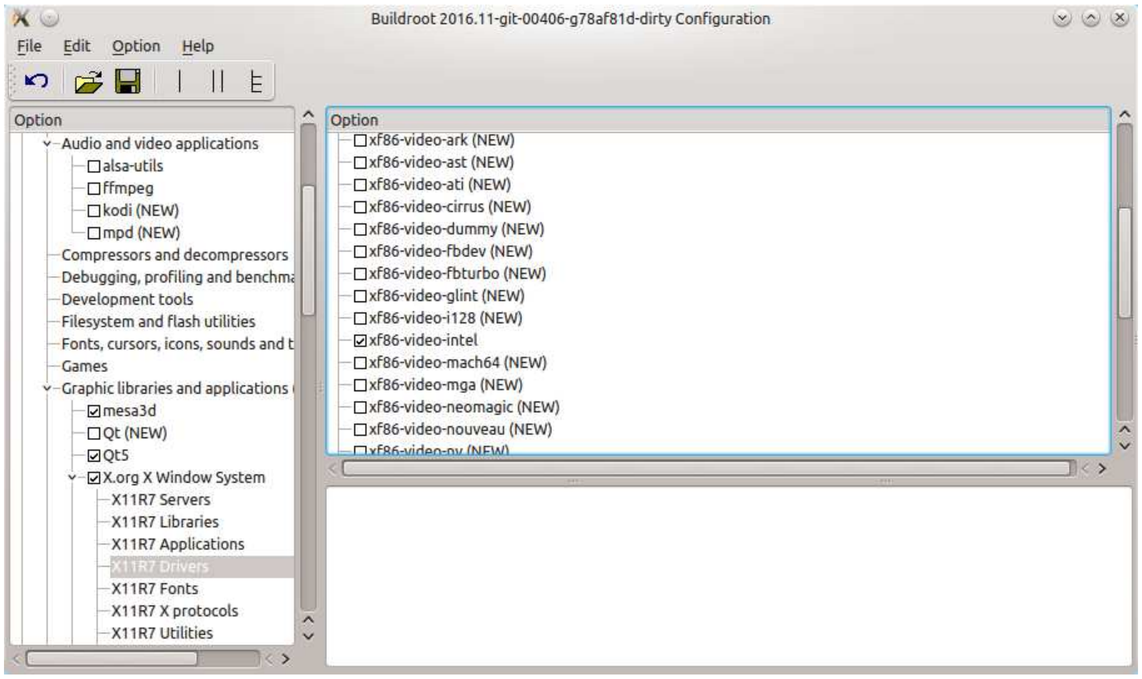
BR2_PACKAGE_QT5BASE_DBUS:

This option enables the D-Bus module.

Figure 18. QT5 configuration

- BR2_PACKAGE_XTERM

Xterm terminal emulator.
- BR2_PACKAGE_XORG7:
Support for X11R7 libraries, servers, drivers, and/or applications in the target.
- BR2_PACKAGE_XSERVER_XORG_SERVER
X.Org X server.
- BR2_PACKAGE_XAPP_TWM
-BR2_PACKAGE_XAPP_XAUTH
X authority file utility.
- BR2_PACKAGE_XAPP_XCLOCK
Analog / digital clock for X.
- BR2_PACKAGE_XAPP_XHOST
Controls host and/or user access to a running X server.
- BR2_PACKAGE_XAPP_XINIT
X Window System initializer.
- BR2_PACKAGE_XDRIVER_XF86_VIDEO_INTEL
Intel video driver.



Option
- Audio and video applications
  - ☐ alsa-utils
  - ☐ ffmpeg
  - ☐ kodi (NEW)
  - ☐ mpd (NEW)
- Compressors and decompressors
- Debugging, profiling and benchma
- Development tools
- Filesystem and flash utilities
- Fonts, cursors, icons, sounds and t
- Games
- Graphic libraries and applications
  - ☑ mesa3d
  - ☐ Qt (NEW)
  - ☑ Qt5
  - ☑ X.org X Window System
    - X11R7 Servers
    - X11R7 Libraries
    - X11R7 Applications
    - X11R7 Drivers
    - X11R7 Fonts
    - X11R7 X protocols
    - X11R7 Utilities

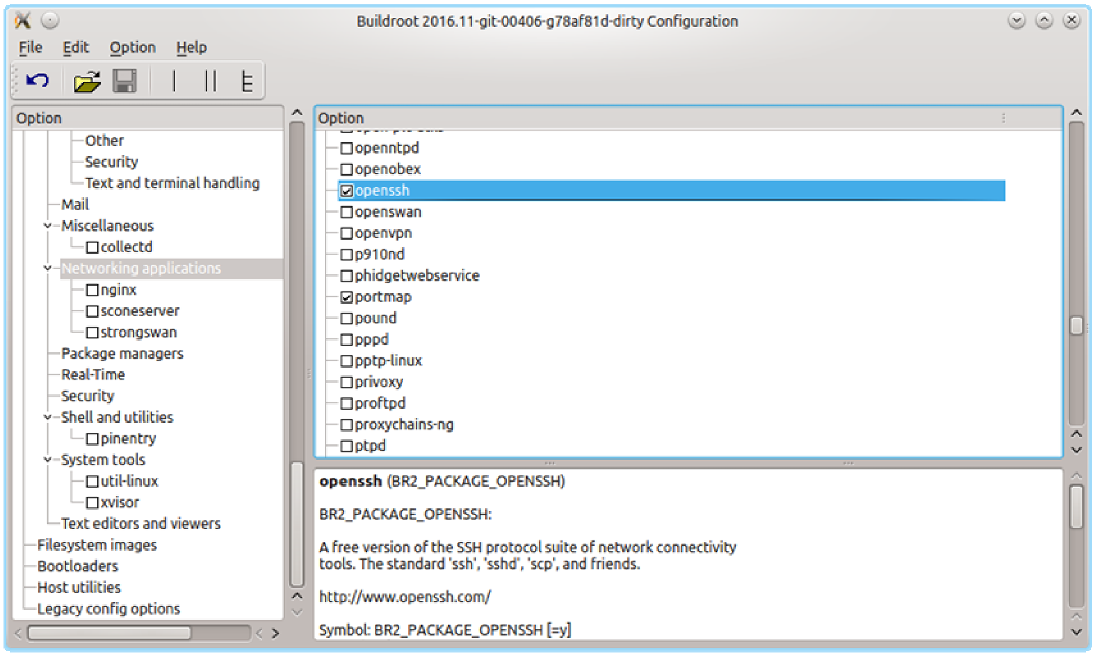
Option
- ☐ xf86-video-ark (NEW)
- ☐ xf86-video-ast (NEW)
- ☐ xf86-video-ati (NEW)
- ☐ xf86-video-cirrus (NEW)
- ☐ xf86-video-dummy (NEW)
- ☐ xf86-video-fbdev (NEW)
- ☐ xf86-video-fbturbo (NEW)
- ☐ xf86-video-glint (NEW)
- ☐ xf86-video-i128 (NEW)
- ☑ xf86-video-intel
- ☐ xf86-video-mach64 (NEW)
- ☐ xf86-video-mga (NEW)
- ☐ xf86-video-neomagic (NEW)
- ☐ xf86-video-nouveau (NEW)
- ☐ xf86-video-nv (NEW)

Figure 19. Mesa3D configuration

4.4.4.7. Networking Applications

- BR2_PACKAGE_DHCP
DHCP relay agent from the ISC DHCP distribution.

- BR2_PACKAGE_DHCP_CLIENT
DHCP client from the ISC DHCP distribution.
- BR2_PACKAGE_OPENSSH
A free version of the SSH protocol suite of network connectivity tools. The standard 'ssh', 'sshd', 'scp', and friends.
- BR2_PACKAGE_PORTMAP
The standard portmapper for Remote Procedure Call (RPC) services.



openssh (BR2_PACKAGE_OPENSSH)

BR2_PACKAGE_OPENSSH:

A free version of the SSH protocol suite of network connectivity tools. The standard 'ssh', 'sshd', 'scp', and friends.

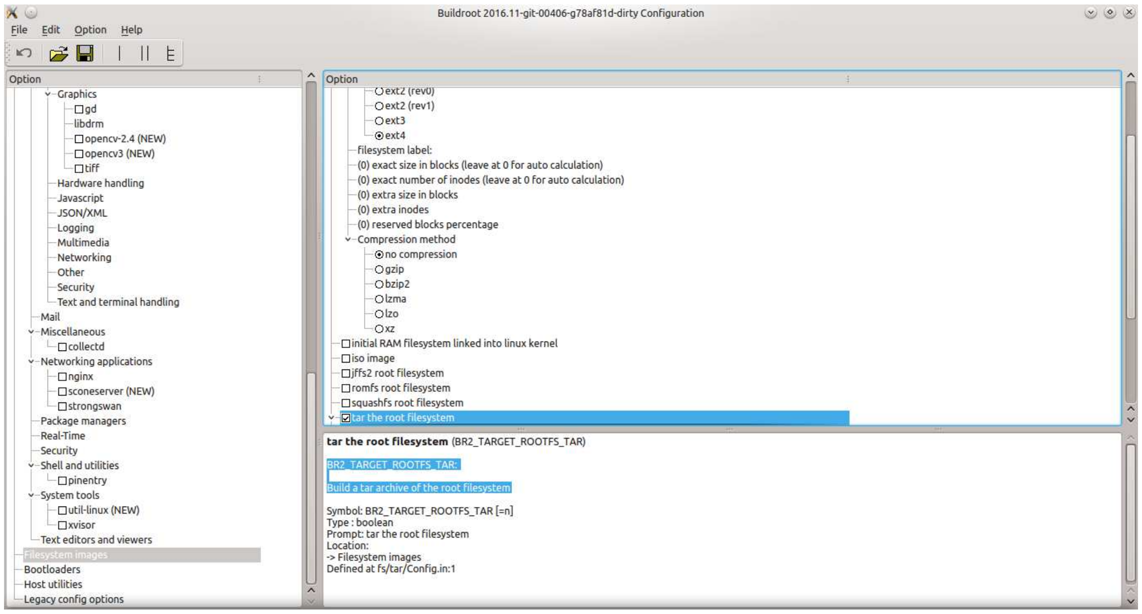
http://www.openssh.com/

Symbol: BR2_PACKAGE_OPENSSH [=y]

Figure 20. Networking applications configuration

4.4.4.8. Filesystem Images

- BR2_TARGET_ROOTFS_EXT2_4
Build an ext4 rootfs.
- BR2_TARGET_ROOTFS_TAR
Build a tar archive of the root filesystem



```
Buildroot 2016.11-git-00406-g78af81d-dirty Configuration

Graphics
    ☐ gd
    libdrm
    ☐ opencv-2.4 (NEW)
    ☐ opencv3 (NEW)
    ☐ tiff
  Hardware handling
  Javascript
  JSON/XML
  Logging
  Multimedia
  Networking
  Other
  Security
  Text and terminal handling
Mail
Miscellaneous
  ☐ collectd
Networking applications
  ☐ nginx
  ☐ sconeserver (NEW)
  ☐ strongswan
Package managers
Real-Time
Security
Shell and utilities
  ☐ pinentry
System tools
  ☐ util-linux (NEW)
  ☐ xvisor
Text editors and viewers
Filesystem images
Bootloaders
Host utilities
Legacy config options
```

```
    ○ ext2 (rev0)
    ○ ext2 (rev1)
    ○ ext3
    ● ext4
  filesystem label:
  (0) exact size in blocks (leave at 0 for auto calculation)
  (0) exact number of inodes (leave at 0 for auto calculation)
  (0) extra size in blocks
  (0) extra inodes
  (0) reserved blocks percentage
  Compression method
    ● no compression
    ○ gzip
    ○ bzip2
    ○ lzma
    ○ lzo
    ○ xz
☐ initial RAM filesystem linked into linux kernel
☐ iso image
☐ jffs2 root filesystem
☐ romfs root filesystem
☐ squashfs root filesystem
☑ tar the root filesystem
```

```
tar the root filesystem (BR2_TARGET_ROOTFS_TAR)

BR2_TARGET_ROOTFS_TAR:

Build a tar archive of the root filesystem

Symbol: BR2_TARGET_ROOTFS_TAR [=n]
Type : boolean
Prompt: tar the root filesystem
Location:
-> Filesystem images
Defined at fs/tar/Config.in:1
```

Figure 21. Filesystem images configuration

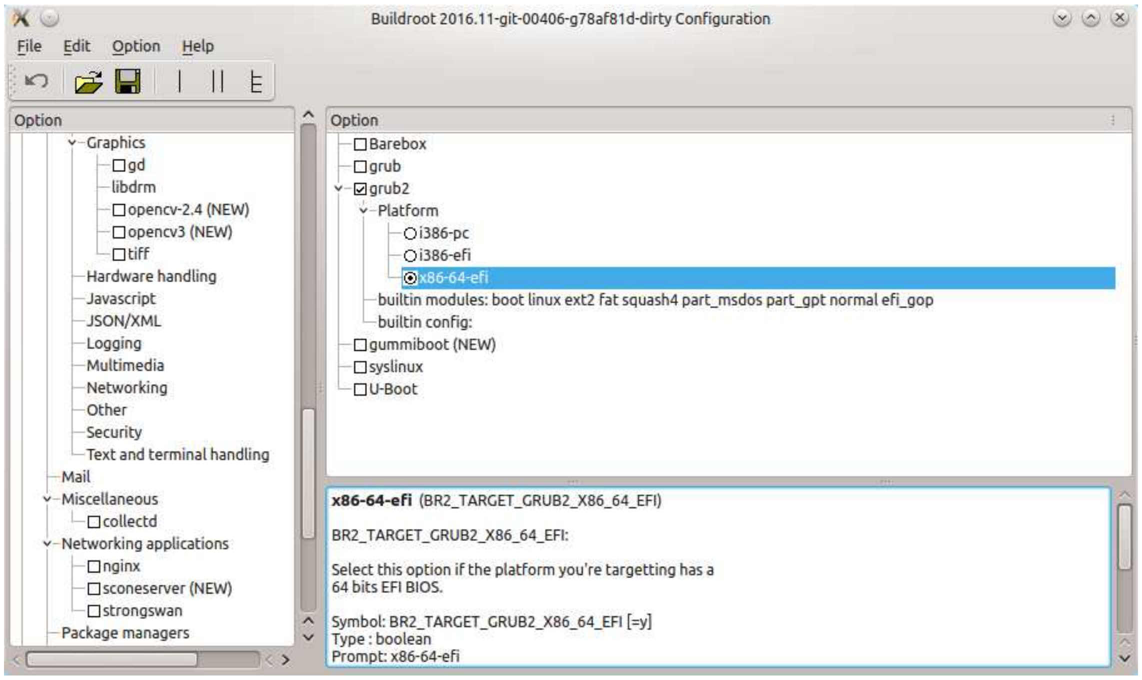
4.4.4.9. Bootloader

- BR2_TARGET_GRUB2

GNU GRUB is a Multiboot boot loader. It was derived from GRUB, the GRand Unified Bootloader, which was originally designed and implemented by Erich Stefan Boleyn. GRUB 2 has replaced what was formerly known as GRUB (i.e. version 0.9x), which has, in turn, become GRUB Legacy.

- BR2_TARGET_GRUB2_X86_64_EFI

Select this option if the platform you're targeting has a 64 bits EFI BIOS.



☐ Barebox
☐ grub
☑ grub2
  Platform
    ○ i386-pc
    ○ i386-efi
    ● x86-64-efi
  builtin modules: boot linux ext2 fat squash4 part_msdos part_gpt normal efi_gop
  builtin config:
☐ gummiboot (NEW)
☐ syslinux
☐ U-Boot

x86-64-efi (BR2_TARGET_GRUB2_X86_64_EFI)

BR2_TARGET_GRUB2_X86_64_EFI:

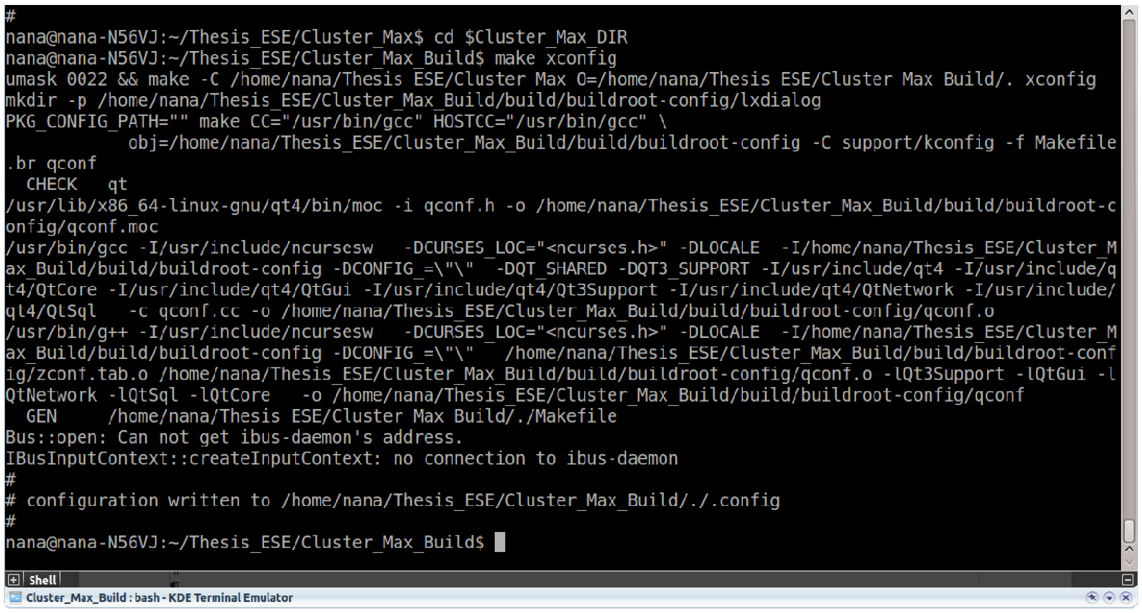
Select this option if the platform you're targetting has a
64 bits EFI BIOS.

Symbol: BR2_TARGET_GRUB2_X86_64_EFI [=y]
Type : boolean
Prompt: x86-64-efi

Figure 22. Bootloader configuration

4.5. Building the Kernel

Proceed to save the configuration (Ctrl + s)

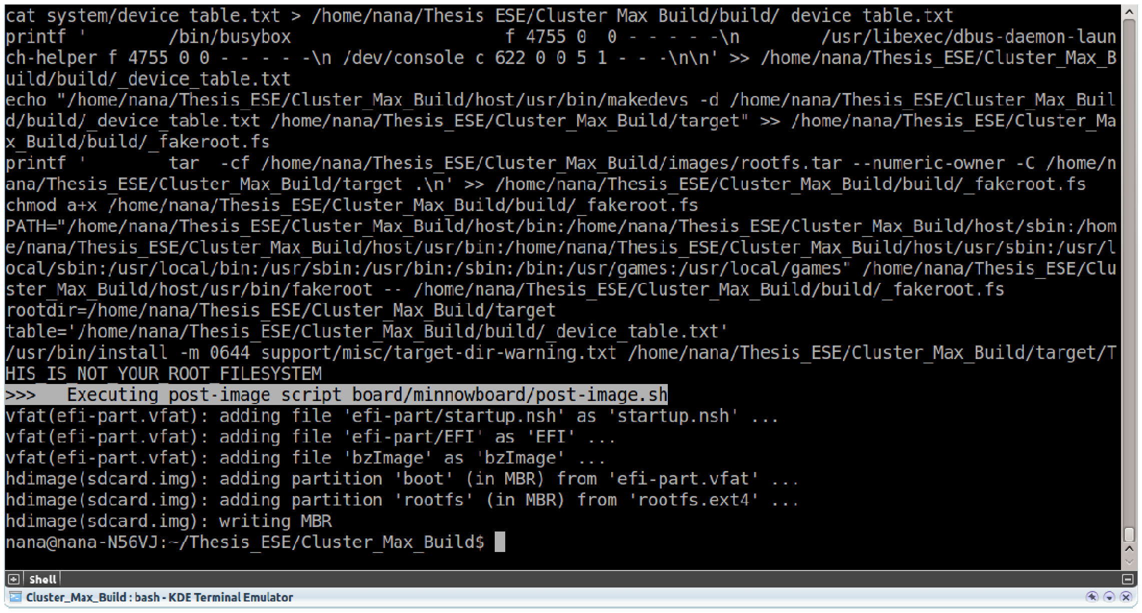


```
#
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max$ cd $Cluster_Max_DIR
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max_Build$ make xconfig
umask 0022 && make -C /home/nana/Thesis_ESE/Cluster_Max O=/home/nana/Thesis_ESE/Cluster_Max_Build/. xconfig
mkdir -p /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config/lxdialog
PKG_CONFIG_PATH="" make CC="/usr/bin/gcc" HOSTCC="/usr/bin/gcc" \
            obj=/home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config -C support/kconfig -f Makefile
.br qconf
  CHECK   qt
/usr/lib/x86_64-linux-gnu/qt4/bin/moc -i qconf.h -o /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-c
onfig/qconf.moc
/usr/bin/gcc -I/usr/include/ncursesw   -DCURSES_LOC="<ncurses.h>" -DLOCALE  -I/home/nana/Thesis_ESE/Cluster_M
ax_Build/build/buildroot-config -DCONFIG_=\"\"  -DQT_SHARED -DQT3_SUPPORT -I/usr/include/qt4 -I/usr/include/q
t4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4/Qt3Support -I/usr/include/qt4/QtNetwork -I/usr/include/
qt4/QtSql   -c qconf.cc -o /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config/qconf.o
/usr/bin/g++ -I/usr/include/ncursesw   -DCURSES_LOC="<ncurses.h>" -DLOCALE  -I/home/nana/Thesis_ESE/Cluster_M
ax_Build/build/buildroot-config -DCONFIG_=\"\"   /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-conf
ig/zconf.tab.o /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config/qconf.o -lQt3Support -lQtGui -l
QtNetwork -lQtSql -lQtCore   -o /home/nana/Thesis_ESE/Cluster_Max_Build/build/buildroot-config/qconf
  GEN     /home/nana/Thesis_ESE/Cluster_Max_Build/./Makefile
Bus::open: Can not get ibus-daemon's address.
IBusInputContext::createInputContext: no connection to ibus-daemon
#
# configuration written to /home/nana/Thesis_ESE/Cluster_Max_Build/./.config
#
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max_Build$
```

Figure 23. Saving configuration (“.config”)

Start the build (this can take a few hours the first time):

```
$ cd $Cluster_Max_DIR
$ make
```



```
cat system/device table.txt > /home/nana/Thesis ESE/Cluster Max Build/build/ device table.txt
printf '        /bin/busybox                        f 4755 0  0 - - - - - -\n        /usr/libexec/dbus-daemon-laun
ch-helper f 4755 0 0 - - - - - -\n /dev/console c 622 0 0 5 1 - - -\n\n' >> /home/nana/Thesis_ESE/Cluster_Max_B
uild/build/_device_table.txt
echo "/home/nana/Thesis_ESE/Cluster_Max_Build/host/usr/bin/makedevs -d /home/nana/Thesis_ESE/Cluster_Max_Buil
d/build/_device_table.txt /home/nana/Thesis_ESE/Cluster_Max_Build/target" >> /home/nana/Thesis_ESE/Cluster_Ma
x_Build/build/_fakeroot.fs
printf '        tar  -cf /home/nana/Thesis_ESE/Cluster_Max_Build/images/rootfs.tar --numeric-owner -C /home/n
ana/Thesis_ESE/Cluster_Max_Build/target .\n' >> /home/nana/Thesis_ESE/Cluster_Max_Build/build/_fakeroot.fs
chmod a+x /home/nana/Thesis_ESE/Cluster_Max_Build/build/_fakeroot.fs
PATH="/home/nana/Thesis_ESE/Cluster_Max_Build/host/bin:/home/nana/Thesis_ESE/Cluster_Max_Build/host/sbin:/hom
e/nana/Thesis_ESE/Cluster_Max_Build/host/usr/bin:/home/nana/Thesis_ESE/Cluster_Max_Build/host/usr/sbin:/usr/l
ocal/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games" /home/nana/Thesis_ESE/Clu
ster_Max_Build/host/usr/bin/fakeroot -- /home/nana/Thesis_ESE/Cluster_Max_Build/build/_fakeroot.fs
rootdir=/home/nana/Thesis_ESE/Cluster_Max_Build/target
table='/home/nana/Thesis_ESE/Cluster_Max_Build/build/_device_table.txt'
/usr/bin/install -m 0644 support/misc/target-dir-warning.txt /home/nana/Thesis_ESE/Cluster_Max_Build/target/T
HIS_IS_NOT_YOUR_ROOT_FILESYSTEM
>>>   Executing post-image script board/minnowboard/post-image.sh
vfat(efi-part.vfat): adding file 'efi-part/startup.nsh' as 'startup.nsh' ...
vfat(efi-part.vfat): adding file 'efi-part/EFI' as 'EFI' ...
vfat(efi-part.vfat): adding file 'bzImage' as 'bzImage' ...
hdimage(sdcard.img): adding partition 'boot' (in MBR) from 'efi-part.vfat' ...
hdimage(sdcard.img): adding partition 'rootfs' (in MBR) from 'rootfs.ext4' ...
hdimage(sdcard.img): writing MBR
nana@nana-N56VJ:~/Thesis_ESE/Cluster_Max_Build$ 
```

Figure 24. End of Cluster_Max SDK compilation

The "make" command executes the following sequence [3]:

1. Download source files.
2. Configure, build and install the cross-compilation toolchain, or simply import an external toolchain.
3. Configure, build and install selected target packages.
4. Build a kernel image, if selected.
5. Build a bootloader image, if selected.
6. Create a root filesystem in selected formats.

4.6. Output Files and Binaries Generated after the Build Process

Buildroot output is stored in a single directory, $Cluster_Max_DIR. This directory contains several subdirectories [3]:

images/ Kernel image, bootloader and root filesystem are stored here. These are intended to be put on target system (Minnowboard Max).

build/ The tools needed by Buildroot on the host and packages compiled for the target are built. This directory contains one subdirectory for each of these components.
staging/ Contains a hierarchy similar to a root filesystem. This directory contains the headers and libraries of the cross-compilation toolchain and all the userspace packages selected for the target. It contains several files, unstripped libraries and binaries used to compile programs that depends on other libraries (this is not the root filesystem).
target/ Contains almost the complete root filesystem for the target except the device files in /dev/ (this is not the root filesystem).
host/ Contains the installation of tools compiled for the host that are needed for the proper execution of Buildroot, including the cross-compilation toolchain.

4.7. Writing Image to mSD Card (Host)

After the build process is done, "sdcard.img" is generated inside /home/nana/Thesis_ESE/Cluster_Max_Build/images/.

Insert an empty mSD card (at least 2Gb is recommended) inside host slot and proceed to burn "sdcard.img" into it.

Find the /dev/ name corresponding to the mSD card by typing the following command:

```
$ ls /dev/sd*
```

Or

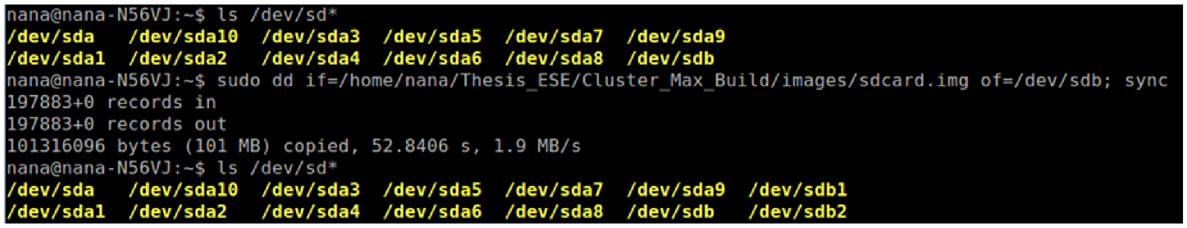
```
$ df -l
```

In this case mSD card name corresponds to "sdb", proceed to execute the following command:

```
$ sudo dd if=/home/nana/Thesis_ESE/Cluster_Max_Build/images/sdcard.img of=/dev/sdb; sync
```

"dd" command assign size partition by default, and it is not enough to run application and saving the images. With the help of GParted or any similar program, partition capacity can be increased.

Check again /dev/ names (execute command: $ ls /dev/sd*) and the two new partitions, boot (/dev/sdb1) and rootfs (/dev/sdb2), shall be shown.



```
nana@nana-N56VJ:~$ ls /dev/sd*
/dev/sda   /dev/sda10  /dev/sda3  /dev/sda5  /dev/sda7  /dev/sda9
/dev/sda1  /dev/sda2   /dev/sda4  /dev/sda6  /dev/sda8  /dev/sdb
nana@nana-N56VJ:~$ sudo dd if=/home/nana/Thesis_ESE/Cluster_Max_Build/images/sdcard.img of=/dev/sdb; sync
197883+0 records in
197883+0 records out
101316096 bytes (101 MB) copied, 52.8406 s, 1.9 MB/s
nana@nana-N56VJ:~$ ls /dev/sd*
/dev/sda   /dev/sda10  /dev/sda3  /dev/sda5  /dev/sda7  /dev/sda9  /dev/sdb1
/dev/sda1  /dev/sda2   /dev/sda4  /dev/sda6  /dev/sda8  /dev/sdb   /dev/sdb2
```

Figure 25. Checking mSD card name

Remove mSD card from host, then insert into the MinnowBoard Max and power the board on.

4.8. Network Configuration and Target Setup

In order to connect target and host through SSH the following steps need to be done:

Target

Set static IP address on MinnowBoard Max

# nano /etc/network/interfaces

Edit the file with the following content:

```
auto lo eth0
iface lo inet loopback

iface eth0 inet static
  address 192.168.10.100
  netmask 255.255.255.0
```

Save the file and restart MinnowBoard Max:

# halt

Then, power off and power on board again.

Verify ethernet configuration:

# ifconfig -a

Command response must show eth0 inet address as 192.168.10.100.

Further configuration is needed to establish connection through SSH with host.

To get target username:

```
# whoami
```
Command response is: root.

To change target password:
```
# passwd
```
Set it to root.

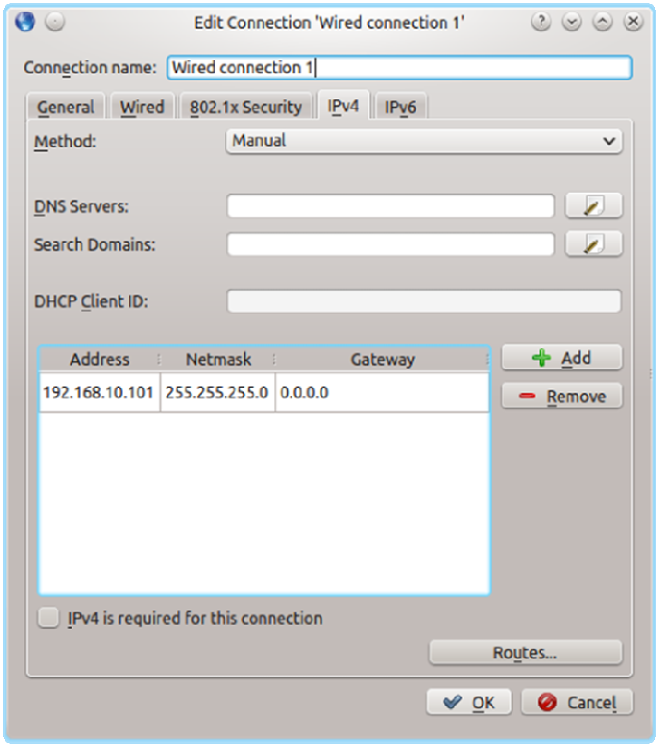
Generate and add SSH key:
```
# ssh-keygen
```
Let the default options by typing intro.
```
# cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys
```

Edit file “sshd_config” in target:
```
# nano /etc/ssh/sshd_config
```
The following options are mandatory in SSH target configuration:
```
Port 22 (uncommented)
PermitRootLogin yes (uncommented)
RSAAuthentication no (uncommented)
PubkeyAuthentication no (uncommented)
AuthorizedKeysFile .ssh/authorized_keys (uncommented)
```

Save the file and restart MinnowBoard Max:
```
# halt
```
Then, power off and power on board again.

Host

Connect target and host through ethernet cable and set static IP address on host, for that select wired connection, then under “IPV4” tab set the following information:
Method: Manual
Address: 192.168.10.101
Netmask: 255.255.255.0



Connection name: Wired connection 1

Method: Manual

DNS Servers:

Search Domains:

DHCP Client ID:

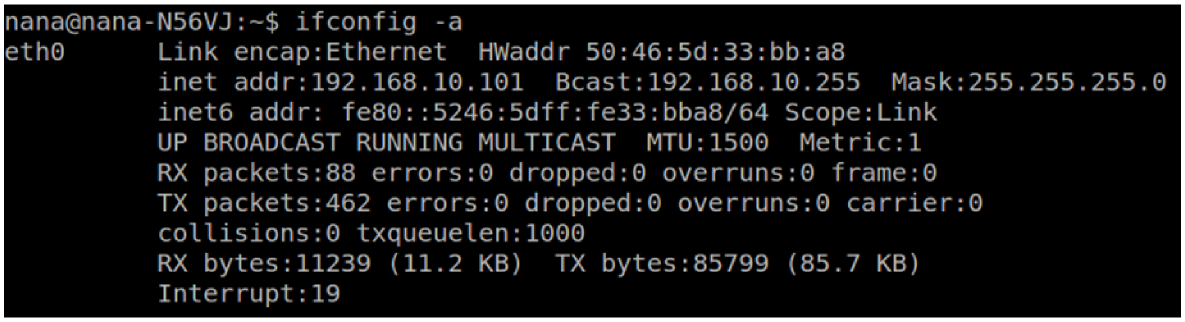
Address	Netmask	Gateway
192.168.10.101	255.255.255.0	0.0.0.0

IPv4 is required for this connection

Figure 26. Wired configuration in host

Press “OK” button and verify ethernet configuration:

$ ifconfig -a

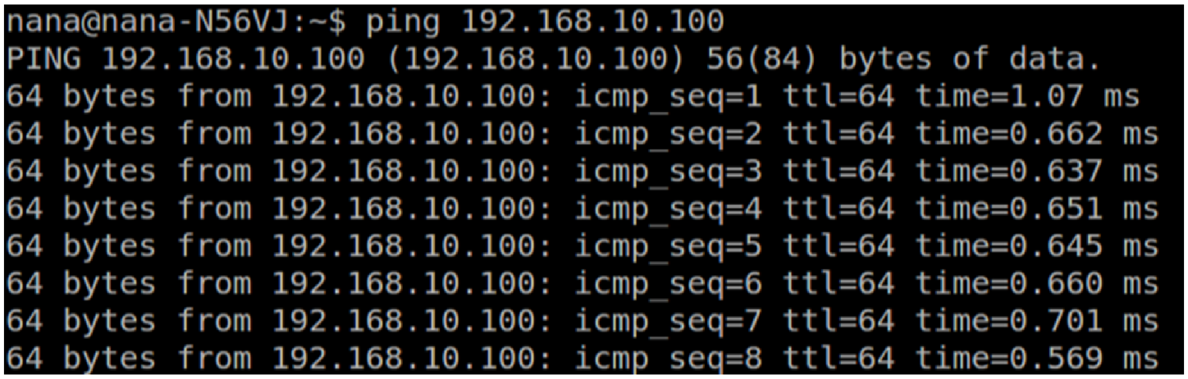


```
nana@nana-N56VJ:~$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 50:46:5d:33:bb:a8
          inet addr:192.168.10.101  Bcast:192.168.10.255  Mask:255.255.255.0
          inet6 addr: fe80::5246:5dff:fe33:bba8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:88 errors:0 dropped:0 overruns:0 frame:0
          TX packets:462 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11239 (11.2 KB)  TX bytes:85799 (85.7 KB)
          Interrupt:19
```

Figure 27. Static IP address on host

Test the connection by pinging target:

$ ping 192.168.10.100

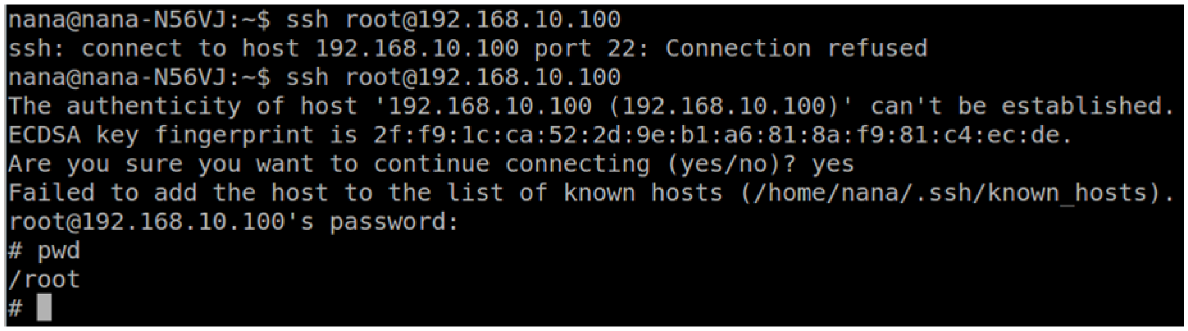


```
nana@nana-N56VJ:~$ ping 192.168.10.100
PING 192.168.10.100 (192.168.10.100) 56(84) bytes of data.
64 bytes from 192.168.10.100: icmp_seq=1 ttl=64 time=1.07 ms
64 bytes from 192.168.10.100: icmp_seq=2 ttl=64 time=0.662 ms
64 bytes from 192.168.10.100: icmp_seq=3 ttl=64 time=0.637 ms
64 bytes from 192.168.10.100: icmp_seq=4 ttl=64 time=0.651 ms
64 bytes from 192.168.10.100: icmp_seq=5 ttl=64 time=0.645 ms
64 bytes from 192.168.10.100: icmp_seq=6 ttl=64 time=0.660 ms
64 bytes from 192.168.10.100: icmp_seq=7 ttl=64 time=0.701 ms
64 bytes from 192.168.10.100: icmp_seq=8 ttl=64 time=0.569 ms
```

Figure 28. Ping target from host

Since target name (root) and its IP address (192.168.10.100) are already known, proceed to connect to target through SSH:

$ ssh root@192.168.10.100



```
nana@nana-N56VJ:~$ ssh root@192.168.10.100
ssh: connect to host 192.168.10.100 port 22: Connection refused
nana@nana-N56VJ:~$ ssh root@192.168.10.100
The authenticity of host '192.168.10.100 (192.168.10.100)' can't be established.
ECDSA key fingerprint is 2f:f9:1c:ca:52:2d:9e:b1:a6:81:8a:f9:81:c4:ec:de.
Are you sure you want to continue connecting (yes/no)? yes
Failed to add the host to the list of known hosts (/home/nana/.ssh/known_hosts).
root@192.168.10.100's password:
# pwd
/root
#
```

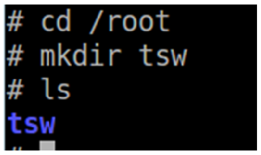
Figure 29. Connecting to target through SSH (Host)

Create “tsw” folder inside /root which is going to contain the Dashboard menu executable file:

Target

cd /root

mkdir tsw



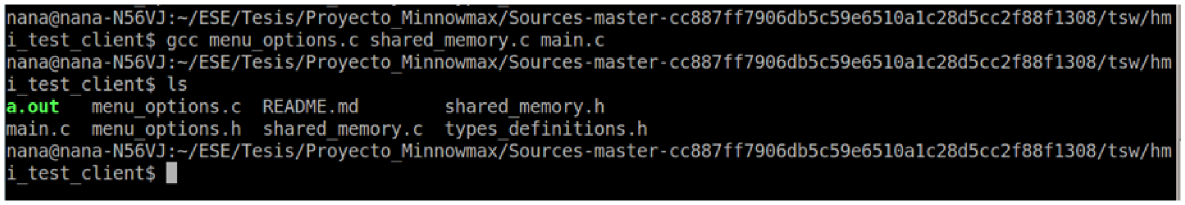
```
# cd /root
# mkdir tsw
# ls
tsw
#
```

Figure 30. “tsw” folder creation on target

Host

Since host and target has the same architecture, just compile “tsw” application with gcc:

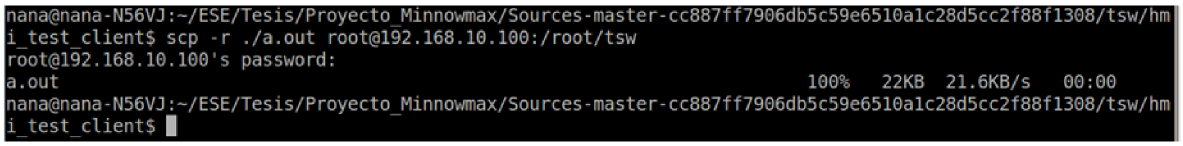
$ cd ../tsw/hmi_test_client
$ gcc menu_options.c shared_memory.c main.c



```
nana@nana-N56VJ:~/ESE/Tesis/Proyecto_Minnowmax/Sources-master-cc887ff7906db5c59e6510a1c28d5cc2f88f1308/tsw/hm
i_test_client$ gcc menu_options.c shared_memory.c main.c
nana@nana-N56VJ:~/ESE/Tesis/Proyecto_Minnowmax/Sources-master-cc887ff7906db5c59e6510a1c28d5cc2f88f1308/tsw/hm
i_test_client$ ls
a.out   menu_options.c  README.md        shared_memory.h
main.c  menu_options.h  shared_memory.c  types_definitions.h
nana@nana-N56VJ:~/ESE/Tesis/Proyecto_Minnowmax/Sources-master-cc887ff7906db5c59e6510a1c28d5cc2f88f1308/tsw/hm
i_test_client$
```

Figure 31. Compiling “tsw” application in host

After compilation is done, copy “a.out” executable file from host to target:
Host
$ scp -r ./a.out root@192.168.10.100:/root/tsw

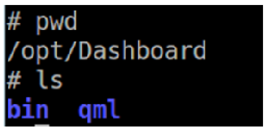


```
nana@nana-N56VJ:~/ESE/Tesis/Proyecto_Minnowmax/Sources-master-cc887ff7906db5c59e6510a1c28d5cc2f88f1308/tsw/hm
i_test_client$ scp -r ./a.out root@192.168.10.100:/root/tsw
root@192.168.10.100's password:
a.out                                                    100%   22KB  21.6KB/s   00:00
nana@nana-N56VJ:~/ESE/Tesis/Proyecto_Minnowmax/Sources-master-cc887ff7906db5c59e6510a1c28d5cc2f88f1308/tsw/hm
i_test_client$
```

Figure 32. Copy executable file generated from compilation to target

Create the following folder in target:
Target
# cd /opt/
# mkdir Dashboard
# cd Dashboard
# mkdir bin

Also, create “qml” folder which is going to contain the qml file, images and fonts needed by the Dashboard interface application
$ mkdir qml



```
# pwd
/opt/Dashboard
# ls
bin  qml
```

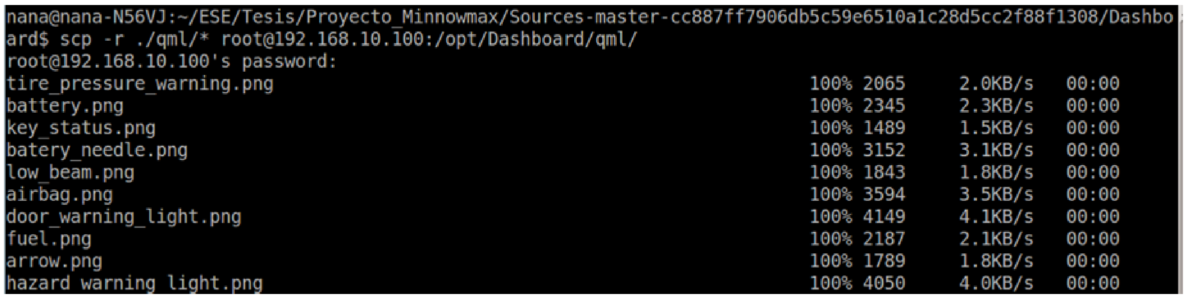
Figure 33. “bin” and “qml” folder creation in target

In host copy “qml” folder that is located inside “Dashboard” folder and paste them into target “qml” folder:

Host

$ cd ../Dashboard

$ scp –r ./qml/Dashboard/* root@192.168.10.100:/opt/Dashboard/qml



```
nana@nana-N56VJ:~/ESE/Tesis/Proyecto_Minnowmax/Sources-master-cc887ff7906db5c59e6510a1c28d5cc2f88f1308/Dashbo
ard$ scp -r ./qml/* root@192.168.10.100:/opt/Dashboard/qml/
root@192.168.10.100's password:
tire_pressure_warning.png                                                    100% 2065     2.0KB/s   00:00
battery.png                                                                  100% 2345     2.3KB/s   00:00
key_status.png                                                               100% 1489     1.5KB/s   00:00
batery_needle.png                                                            100% 3152     3.1KB/s   00:00
low_beam.png                                                                 100% 1843     1.8KB/s   00:00
airbag.png                                                                   100% 3594     3.5KB/s   00:00
door_warning_light.png                                                       100% 4149     4.1KB/s   00:00
fuel.png                                                                     100% 2187     2.1KB/s   00:00
arrow.png                                                                    100% 1789     1.8KB/s   00:00
hazard_warning_light.png                                                     100% 4050     4.0KB/s   00:00
```

Figure 34. Copy images, fonts and “main.qml” from host to target

4.9. QT Creator Configuration (Host)

The QT Creator and QT Libraries have been downloaded from the following web site: https://www.qt.io/download-open-source/

Proceed to install it and open QT Creator. Then configure it:

Select Tools → Options.

Since host and target have the same architecture, x86_64, GCC compiler and QT Libraries used for cross-compiling are the same ones than installed on host machine.

Then select “Build & Run” menu → “Kits” tab and press “Add” button:
Name: MinnowMax
Device type: Generic Linux Device
Press “Manage” button
Press “Add” button

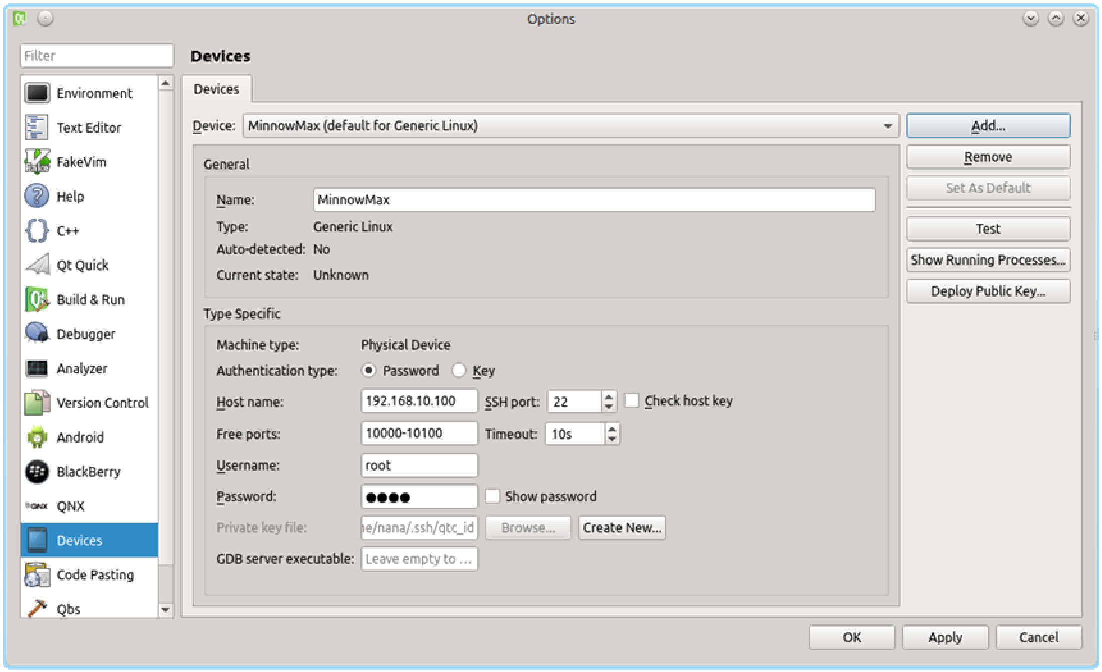
- Select “Generic Linux Device”
- Press “Start Wizard” button and set the following values:
  - The name to identify this configuration: MinnowMax
  - The device's host name or IP address: 192.168.10.100 (Section 4.8)
  - The user name to log into the device: root
  - The authentication type: Password
  - The user's password: root

Press “Next” button.

Press “Finish” button - Verify that the Device was successfully tested.

Press Close button

Press Apply button



Devices

Device: MinnowMax (default for Generic Linux)

General

Name: MinnowMax

Type: Generic Linux

Auto-detected: No

Current state: Unknown

Type Specific

Machine type: Physical Device

Authentication type: [✓ Password] Key

Host name: 192.168.10.100 SSH port: 22 Check host key

Free ports: 10000-10100 Timeout: 10s

Username: root

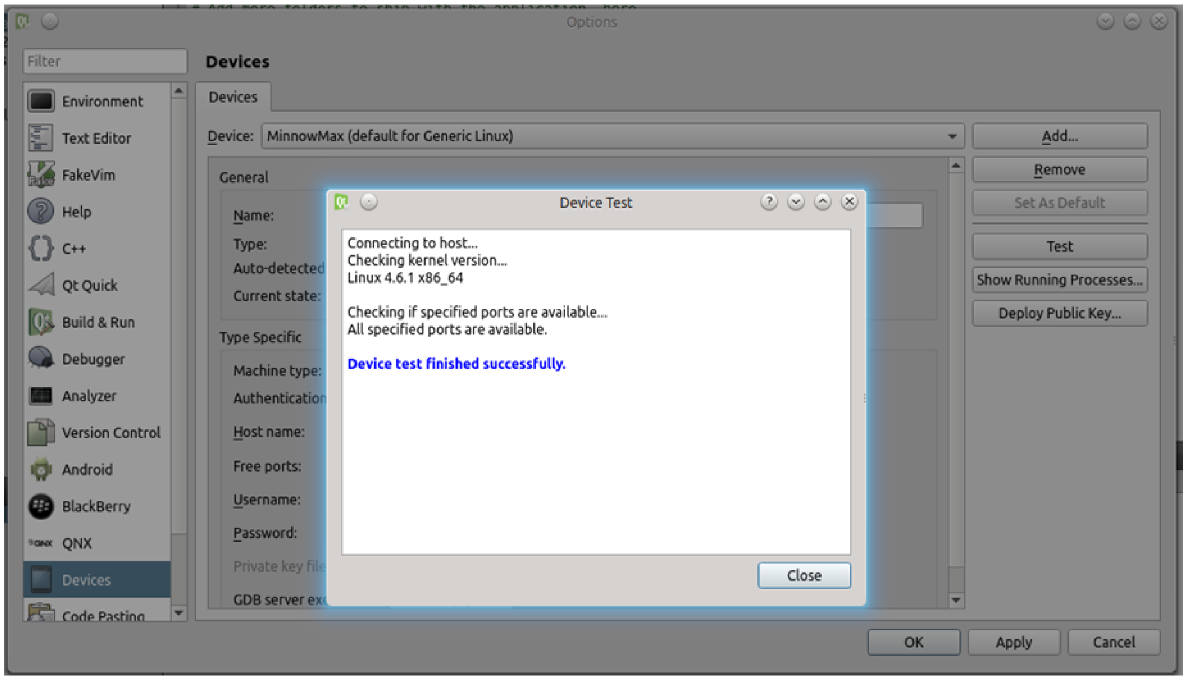
Password: ●●●● Show password

Private key file: e/nana/.ssh/qtc_id

GDB server executable: Leave empty to ...

Figure 35. QT Devices configuration

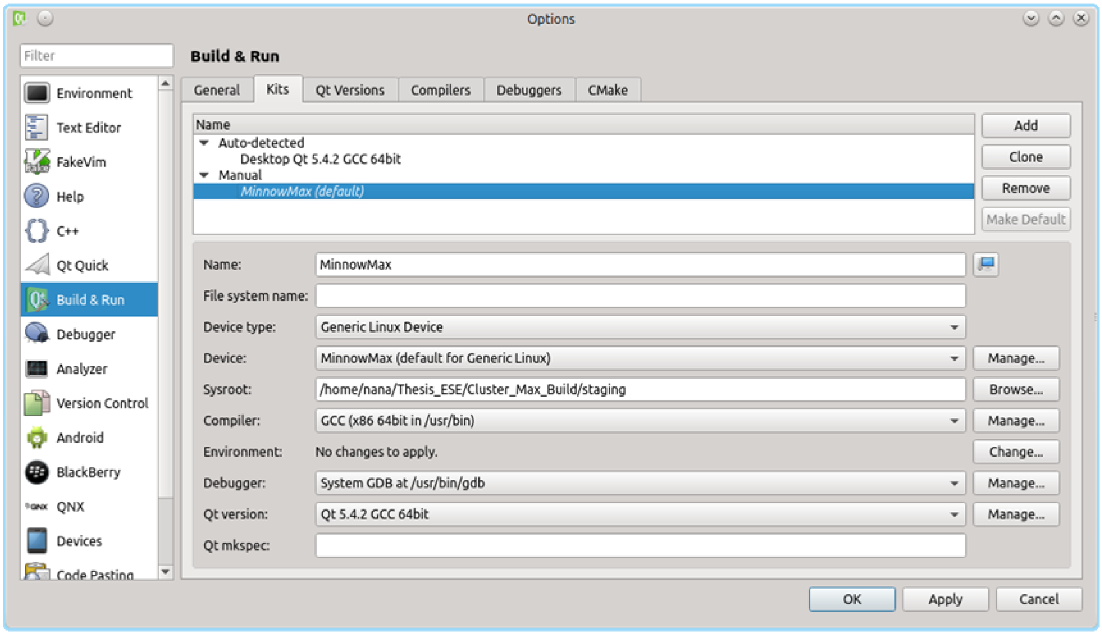
Check that target is properly connected to host by pressing “Test” button:



```
Connecting to host...
Checking kernel version...
Linux 4.6.1 x86_64

Checking if specified ports are available...
All specified ports are available.

Device test finished successfully.
```

Figure 36. Remote connection to target is successful

Return to “Build & Run” menu -> Kits tab in order to finish the kit configuration
Select “MinnowMax (default)” under “Manual” sub-menu and set the following options:
Sysroot: /.../Cluster_Max_Build/staging
Compiler: GCC (x86 64 bit in /usr/bin)
Qt version: Qt 5.4.2 GCC 64 bit



Name
- Auto-detected
  - Desktop Qt 5.4.2 GCC 64bit
- Manual
  - *MinnowMax (default)*

Name:	MinnowMax
File system name:	
Device type:	Generic Linux Device
Device:	MinnowMax (default for Generic Linux)
Sysroot:	/home/nana/Thesis_ESE/Cluster_Max_Build/staging
Compiler:	GCC (x86 64bit in /usr/bin)
Environment:	No changes to apply.
Debugger:	System GDB at /usr/bin/gdb
Qt version:	Qt 5.4.2 GCC 64bit
Qt mkspec:	

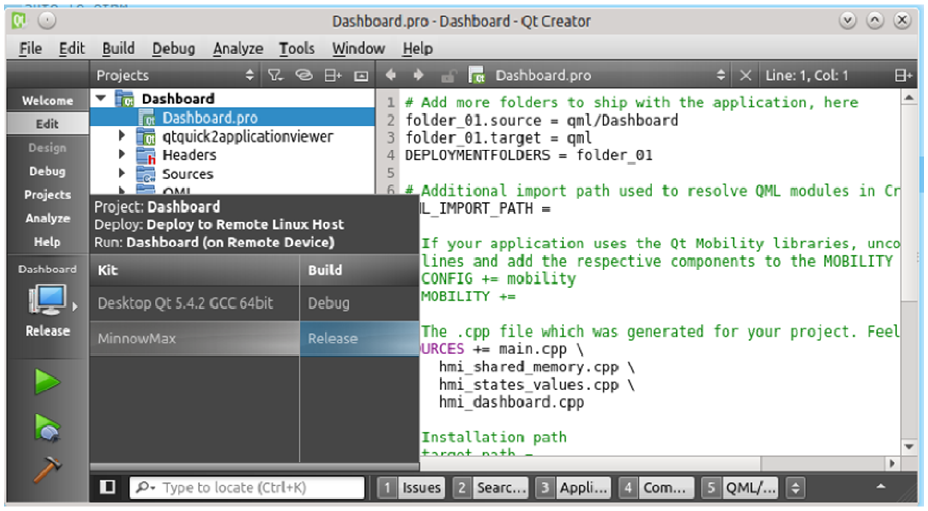
Figure 37. QT Build & Run configuration

Finally, Qt Creator is configured to program on a remote device using the Qt Libraries.

4.10. Cross-Compile “Dashboard” Application on QT Creator (Host)

After configuring QT Creator as mentioned on Section 4.9, proceed to open “Dashboard” application:

1. Host: Open “Dashboard.pro” file with QT Creator. This file is provided along with this document.
2. Host: After Dashboard project is open, select MinnowMax kit and release in build option (Figure 38).



```
# Add more folders to ship with the application, here
folder_01.source = qml/Dashboard
folder_01.target = qml
DEPLOYMENTFOLDERS = folder_01

# Additional import path used to resolve QML modules in Cr
L_IMPORT_PATH =

If your application uses the Qt Mobility libraries, unco
lines and add the respective components to the MOBILITY
CONFIG += mobility
MOBILITY +=

The .cpp file which was generated for your project. Feel
URCES += main.cpp \
  hmi_shared_memory.cpp \
  hmi_states_values.cpp \
  hmi_dashboard.cpp

Installation path
```

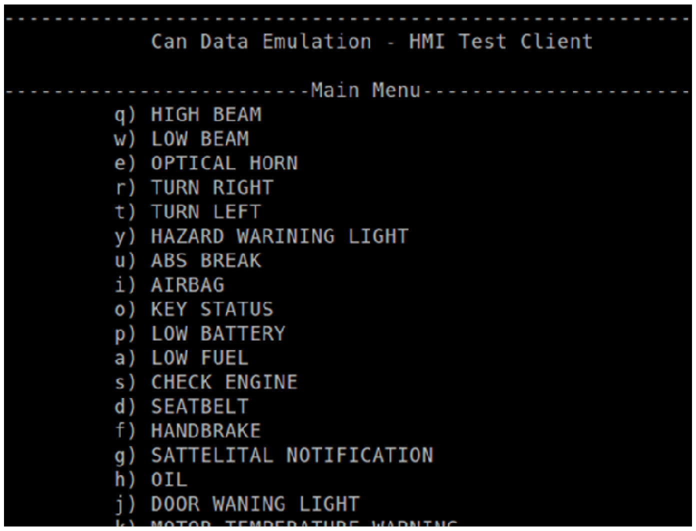
Project: Dashboard
Deploy: Deploy to Remote Linux Host
Run: Dashboard (on Remote Device)

Kit	Build
Desktop Qt 5.4.2 GCC 64bit	Debug
MinnowMax	Release

Figure 38. Deploy to remote Linux host (target)

3. Host: Execute remote console for target (SSH).
4. Target: Execute “xinit” command (press Ctrl + c to exit X11).
5. Host: Build (Ctrl + b) and run (Ctrl + r).
6. Host: Execute remote another console for target (SSH).
7. Target: Run “tsw” application (Figure 39):

   # ./a.out



```
-------------------------------------------------------------
          Can Data Emulation - HMI Test Client

-------------------------Main Menu---------------------
         q) HIGH BEAM
         w) LOW BEAM
         e) OPTICAL HORN
         r) TURN RIGHT
         t) TURN LEFT
         y) HAZARD WARINING LIGHT
         u) ABS BREAK
         i) AIRBAG
         o) KEY STATUS
         p) LOW BATTERY
         a) LOW FUEL
         s) CHECK ENGINE
         d) SEATBELT
         f) HANDBRAKE
         g) SATTELITAL NOTIFICATION
         h) OIL
         j) DOOR WANING LIGHT
```

Figure 39. “tsw” application is running on target

8. Target: After selecting and modifying values in the option menu from remote connection, changes shall be reflected in Dashboard GUI. Results are shown in Section 5.1.

5. RESULTS

Abstract: *In this chapter, the results obtained after executing CLUSTER_PI App in MinnowBoard Max are shown.*

5.1. Results

Results obtained from the procedure detailed in Section 4 are described below:

After executing step 5 in the Section 4.10, Dashboard with initial values is shown in MinnowBoard Max display connected through the Micro-HDMI:

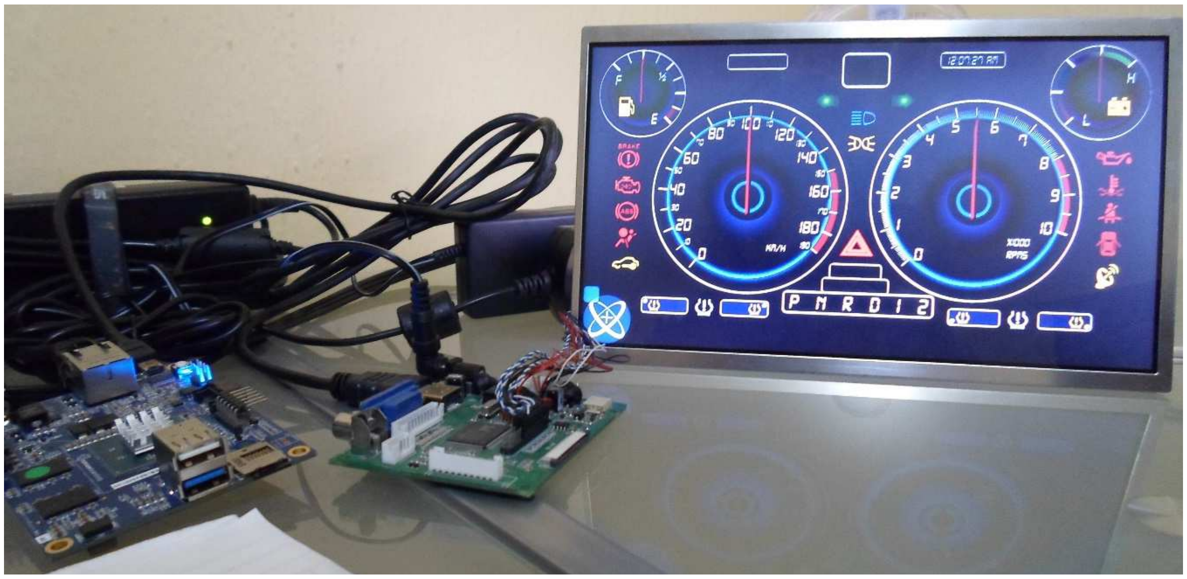
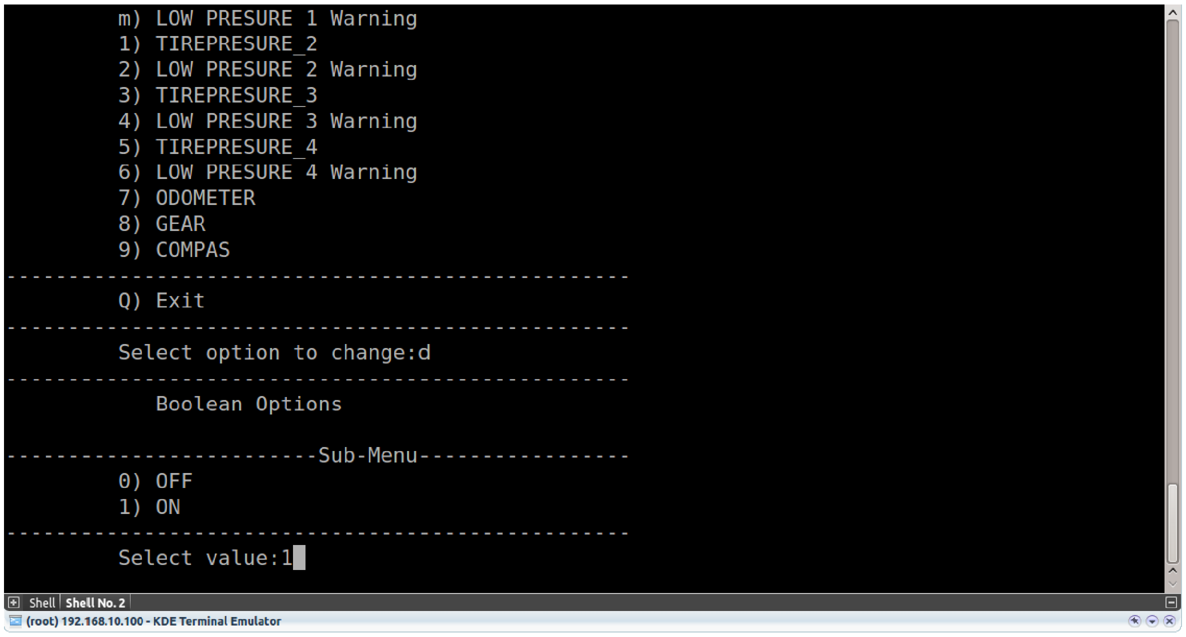


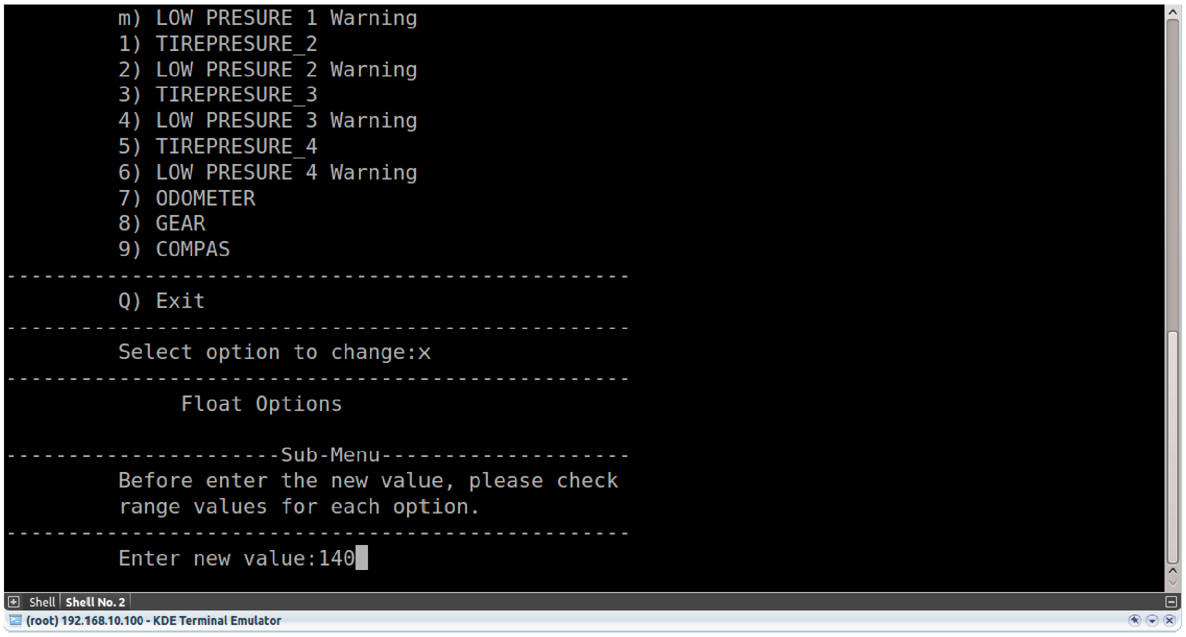
Figure 40. Dashboard with initial values running on target (MinnowMax)

In step 8, seatbelt warning and speedometer are set as follows:



```
        m) LOW PRESURE 1 Warning
        1) TIREPRESURE_2
        2) LOW PRESURE 2 Warning
        3) TIREPRESURE_3
        4) LOW PRESURE 3 Warning
        5) TIREPRESURE_4
        6) LOW PRESURE 4 Warning
        7) ODOMETER
        8) GEAR
        9) COMPAS
--------------------------------------------------
        Q) Exit
--------------------------------------------------
        Select option to change:d
--------------------------------------------------
           Boolean Options

------------------------Sub-Menu-----------------
        0) OFF
        1) ON
--------------------------------------------------
        Select value:1
```

Figure 41. Seatbelt warning was activated in remote “tsw” application



```
        m) LOW PRESURE 1 Warning
        1) TIREPRESURE_2
        2) LOW PRESURE 2 Warning
        3) TIREPRESURE_3
        4) LOW PRESURE 3 Warning
        5) TIREPRESURE_4
        6) LOW PRESURE 4 Warning
        7) ODOMETER
        8) GEAR
        9) COMPAS
--------------------------------------------------
        Q) Exit
--------------------------------------------------
        Select option to change:x
--------------------------------------------------
             Float Options

---------------------Sub-Menu--------------------
        Before enter the new value, please check
        range values for each option.
--------------------------------------------------
        Enter new value:140
```

Figure 42. Speedometer was set to 140 km/h in remote “tsw” application

Commanded options by “tsw” application are applied to Dashboard GUI by pressing intro. Then, changes are refreshed into the image:

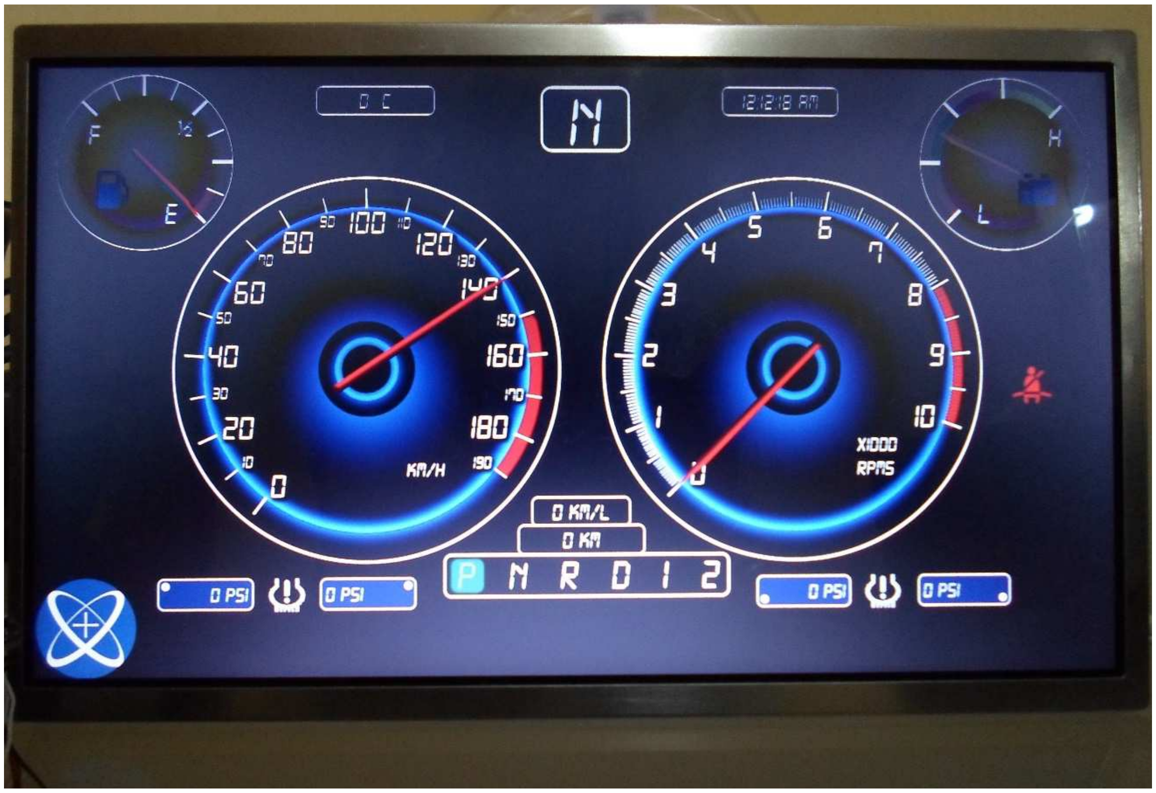


Figure 43. Dashboard with seatbelt warning activated and speedometer set to 140 km/h

Notice that the speed reported for the cluster is now 140 km/h and also the seat-belt is on, as expected.

6. CONCLUSIONS

Abstract: *In this chapter, some conclusions and future work related with this project are presented.*

6.1. Conclusions

This work described the implementation of a digital automotive cluster based on an Intel X86_64 microprocessor architecture. A brief review of the industrial trend on digital clusters has been presented. The MinnowBoard MAX architecture has been reviewed, as well as the software architecture based on a Linux distro. The Buildroot tool has been considered for the customization of the Linux kernel and the construction of a bootable image suitable for the Minnowboard MAX board.

A cross platform development environment based on the QT framework has been configured for the compilation of the CLUSTER_PI application, which is the responsible of the visual representation of the instruments of the cluster.

A shared memory mechanism in the Linux kernel has been used for the inter process communication between the low level drivers, such as CAN and UART, and the CLUSTER_PI application. In this way, the development of the solution can be modularized.

This work is based on the CLUSTER_PI App, which was previously executed on an architecture ARM (Raspberry PI). Buildroot was used to migrate the application to x86_64 architecture. Necessary packages used in CLUSTER _PI, such as: QT5, SSH, etc. were taken as initial requirement. With this information the kernel is configured and then compiled. As a final result of the building a complete Linux Embedded System (bootloader, root filesystem, Kernel image and cross-compilation toolchain) is generated. CLUSTER_PI App was successfully deployed in Minnowboard Max, showing the same graphic environment and functionality than Raspberry PI. Due to this work is focused on kernel configuration, the cluster application was not optimized enough; further analysis needs to be done especially in the graphic related section.

Since both Host machine and target has the x86_64 architecture, cross compilation toolchain is not mandatory to be selected when compiling application since the toolchain available in host can be used (gcc, qmake, etc) allowing more reusability and scalability in more platforms than ARM architecture.

With this work results it is demonstrated than tools like Buildroot allows to user fully personalization of packages, kernel version, binary output format, etc. Also, this tool makes use of several scripts which automates the building process.

Finally, architecture migration was deployed successfully since the Dashboard's graphics were updated after a commanded input was sent from host to target through SSH. However, this work offers many possibilities for cluster reconfiguration and extra functionality, as well as further improvement as mentioned in the following section.

6.2. Future Work

The future work of this development includes an optimization process to speed up the responsiveness of the virtual digital instruments in the cluster. Additionally, further development of low-level driver for its integration within the automotive network is considered. This includes drivers for CAN, Flex-Ray, CAN-FD and Ethernet, among others.

It is highly recommendable to use tools like Buildroot since it allows to user fully personalization of packages, kernel version. However, the use of Yocto for the configuration of the Linux kernel for the MinnowBoard MAX board is considered as future trend in this work.

Besides, the integration of sensors to the cluster, such as Leap motion or Intel RealSense, is considered in future versions of this technological development.

This lead to propose future work focused on:

- Improve Dashboard response time.
- Add CAN driver support to Kernel.
- Include extra functionality with Leap Motion.

In addition, OpenGL can be used along with QT to allow more versatility and customization for future projects.

BIBLIOGRAPHY

[1] Delphi, «Delphi Reconfigurable Displays,» © Delphi Automotive LLP, [En línea]. Available: http://www.delphi.com/manufacturers/auto/controls/displays/reconfigurable_displays. [Último acceso: 27 08 2016].

[2] Magneti Mirelli, «Instrument Clusters,» © Magneti Marelli S.p.A. , [En línea]. Available: http://www.magnetimarelli.com/excellence/technological-excellences/instrument-clusters. [Último acceso: 27 08 2016].

[3] Buildroot, «The Buildroot user manual,» Buildroot, [En línea]. Available: https://buildroot.org/downloads/manual/manual.html. [Último acceso: 27 08 2016].

[4] Yocto Project, «MinnowBoard,» © Yocto Project, A Linux Foundation Collaborative Project., [En línea]. Available: https://www.yoctoproject.org/product/minnowboard. [Último acceso: 27 08 2016].

[5] OpenWrt, «OpenWrt,» OpenWrt, [En línea]. Available: https://openwrt.org. [Último acceso: 27 08 2016].

[6] QT, «QT for Application Development,» © 2016 The Qt Company, [En línea]. Available: https://www.qt.io/qt-for-application-development/. [Último acceso: 05 09 2016].

[7] Indiana University, «What is the X Window System?,» © 2016 The Trustees of Indiana University, 20 08 2015. [En línea]. Available: https://kb.iu.edu/d/adnu. [Último acceso: 05 09 2016].

[8] freedesktop.org, «DRI Wiki,» freedesktop.org, 11 04 2014. [En línea]. Available: https://dri.freedesktop.org/wiki/. [Último acceso: 10 09 2016].

[9] Mesa3D, «The Mesa 3D Graphics Library,» Mesa3D, [En línea]. Available: http://www.mesa3d.org/intro.html. [Último acceso: 09 10 2016].

[10] OpenBSD Project, «OpenSSH,» © 1999-2016 OpenBSD, [En línea]. Available: http://www.openssh.com/. [Último acceso: 11 09 216].

[11] M. Rouse, «Bootloader,» TechTarget, 2016. [En línea]. Available: http://searchenterpriselinux.techtarget.com/definition/boot-loader. [Último acceso: 18 09 2016].

[12] J. O. S. S. A. W. Lars Wirzenius, «Chapter 3. Overview of the Directory Tree,» de *The Linux System Administrator's Guide,* The Linux Documentation Project.

[13] eLinux, «Toolchains,» eLinux, 22 07 2016. [En línea]. Available:

http://elinux.org/Toolchains. [Último acceso: 15 11 2016].

[14] David A Rusling, «Interprocess Communication Mechanisms,» de *The Linux Kernel*, Wokingham, David A Rusling, 1999, pp. 51-58.

[15] M. Streicher, «Speaking UNIX: Interprocess communication with shared memory,» IBM, 28 09 2010. [En línea]. Available: https://www.ibm.com/developerworks/aix/library/au-spunix_sharedmemory/. [Último acceso: 26 10 2016].

[16] MinnowBoard MAX, «MinoowBoard MAX,» MinnowBoard MAX, 29 03 2016. [En línea]. Available: http://wiki.minnowboard.org/MinnowBoard_MAX. [Último acceso: 03 09 2016].

[17] MinnowBoard Max, «Minnow2 Board,» CircuitCo LLC., 25 08 2014. [En línea]. Available: http://www.elinux.org/images/f/fd/MinnowMax_RevA1_sch.pdf. [Último acceso: 04 09 2016].

[18] ASUS, «N56VZ,» ©ASUSTeK Computer Inc., [En línea]. Available: https://www.asus.com/latin/Notebooks/N56VZ/specifications/. [Último acceso: 03 09 2016].

[19] J. O. S. S. A. W. Lars Wirzenius, «Chapter 3. Overview of the Directory Tree,» de *Linux System Administrators Guide*, The Linux Documentation Project.