Final Master Thesis

MASTER'S DEGREE IN AUTOMATIC CONTROL AND ROBOTICS

Design of a test system for embedded processor boards

MEMORY

Autor: Carlos Alonso Barbero
Advisor: Alexandre Perera Lluna

Convocatòria: , 2018

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Abstract

One of the most important part during the manufacturing process of an embedded system board is the final test. This process will guarantee that each of its parts that the board contains works correctly and ensure the embedded system board is ready.

The complete test system is composed by different test that will check all the parts that compose the embedded board. This parts goes from the internal chip to external peripherals. These test need to be robust and completely autonomous.

The test process is often used as the last part of the manufactured process and an operator is the responsible of set up manually the necessary connection and supervise the progression and results.

In this project it will be proposed a test system which reduce as much as possible the testing time in order to reduce the production cost of each board. This system will be designed for some particular embedded system boards but it will be adaptive to simply export it to new embedded system boards.

Content

List of Figures

List of Tables

1 Introduction

Nowadays the electronic industry is one of the most important around the word. It is growing up every day more and more. One of the most popular field inside the electronic industry is the embedded field.

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors are manufactured as components of embedded systems.

Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.

Nowadays it is usually to find embedded systems as “single-board computers”. A single-board computer is a complete computer built on a single circuit board, with microprocessor, memory, input/output pins, peripheral interfaces and other features required of a functional computer. Single-board computers were made as demonstration or development systems, for use as embedded computer controllers. Many types of home computers or portable computers integrate all their functions onto a single printed circuit board.

Figure 1: Single-Board Computer based on iMX6 from ISEE company

Single-board computers are quite extended in our days. There are a lot of application where these embedded systems are present: industrial, educational, medicine, robotics, etc.

1.1 Motivation

The competence between embedded system companies is really high due to the easy access to the embedded industry and the low capital necessary to start manufacturing or developing embedded system boards. The key point in order to excel above the rest of the companies is to reduce the cost the maximum possible without losing quality.

The test process at factory guarantee that the final embedded system board is going to be completely functional. The better and exhaustive the process the more time it takes to completely test an embedded board and the more time the operator use to test all the produced boards at factory. This basically means that it is needed an in-depth test process in the minimal possible time in order to reduce the costs.

This project seeks to create an exhaustive test that ensure that each component of the embedded system board is working correctly even in a stress situation in the minimal possible time. This test will guarantee that each embedded system board that pass the complete test is going to be able to work appropriately in any situation including the most demanding conditions.

1.2 Scope

An embedded system consists of heterogeneous layers including hardware, HAL (hardware abstraction layer), OS kernel and application layer. Interactions between these layers are the software interfaces to be tested in an embedded system.

One of the most fascinating aspects of embedded development is that the development is presented in both fields: hardware and software. This aspect is one of the most important strengths of the embedded systems because it gives almost a total freedom to develop the desired project with all details in the embedded system molding the hardware to fulfill all the necessities and using the software to complete this process. This advantage that these systems presents is one of its weakness too. The development of the hardware and the software need to be precise and concrete. This concept could be a little bit controversial. Let's explain it with a little real example that, in fact, happened in the company where this project was developed.

During the development of a single-board computer, some prototypes were manufactured in order to test them and start the production of this board. During the test procedure it was needed to design a new software (u-boot bootloader and Linux Kernel) adapted to this new board. This process is really complex because it is needed to set in the software all the hardware specifications. Is a long process where it is necessary to deal with the specifications of each different chip producer datasheets in order to fulfill all of them. It is important to know

that the chips and components that form part of the single-board computer are usually from different manufacturers: the processor form NXP, the PMIC form TI, the eMMC from WD, etc.

Figure 2: Embedded Chip common producers

One of the most complicate problem that happened during the testing of this prototype was related with the audio. Apparently, the audio was not working in these prototypes. When this kind of problems appear it is necessary to follow an analytic process checking all the components that form part or are related with the problematic component. After some time, it was detected that the problem was that the i2c communication between the audio codec and the processor was not working. After more and more research it was detected that the pull-up resistors of this i2c bus were in the limit compare with the datasheet specifications. It was a problem of both fields, software and hardware. The hardware design presented a pull-up resistor and the software set another internal pull-up resistor and the combination in parallel of both results in a final pull-up resistor insufficient to the i2c bus. The solution was easily modifying it by software and the audio system starts to work correctly.

The previous example was looking for explain the complicate that is to design a complete embedded system and the precision and concretion needed in order everything works perfectly.

2 State-of-the-art

The factory test of Single-Board Computers is completely extended in today's world. It is the only way to check that the product that is going out to the market is working correctly. It tests the functionality and the performance of each board.

There is not a standard of a test at factory of a Single-Board Computer. Almost all of this projects are property of each company and are not public. This project is a redesign of the test created by the company ISEE 2007 S.L that will improve the performance and result in order to speed up the test time and decrease the false positives.

ISEE is a specialist engineering company in the Embedded Systems founded in 2006 by Agustí Fontquerni and Manel Caro on the basis of dedication, honesty and knowledge to offer complete embedded solutions that help industries improving their production level, lowering costs and reducing time to market of their products. (*Who is ISEE, 2016)*

The experienced gained over all these years of hard work, is a backup that support its solutions through a well experienced team of engineers always committed to deliver tailor made and unique solutions for every need requested. ISEE products offer advanced function and open source software at industry-leading prices in low or high volume.

When ISEE developed and started manufacturing its first Single-Board Computer, it appears the necessity of build a complete test at factory in order to ensure the quality of its product.

This test has been improving during the years up now.

3 Objectives

Many of today's products contain quite complex components for information collection, processing and exchange. Besides, many of those components come in packages whose tens or hundreds of pins are either very narrow pitch or not even visible anymore. For many products, quality is an issue, not only for cost reasons but also for customer satisfaction; losing a reputation might be costly at the end or even lead to bankruptcy of a company. **The main objective is to design an efficient but comprehensive factory test system that avoid any not correct board to go out to the market.** In order to do so, it is important to divide this extensive main objective in to several simple objectives:

1- Identify each component that form part of a prototype of a Single-Board Computer.

The first step is to identify and describe all the different components that are essential in the correct working of an embedded system and analyze its normal performing in order to know how the works and how to correctly test them.

2- Check the most common failures of each component.

The second step is to analyze the most common ways of failures presented on each component. This will allow to build a concrete test for each component preventing common failures at the same time that check the correct performance of each component.

3- Design an adapted particular method to completely test each component.

It is important to develop a concrete test process for each component. The test process of two completely different components is not necessary the same. For instance, it is not the same the test process needed to check the video system than the test process needed to check the NAND flash. In the first case it is needed to check that a concrete image or video is being displayed and in the second case it is necessary to check it the NAND is able to record and show data inside it at the same time that there are not more bad blocks than expected.

4- Build a specific bootloader and Linux Kernel for this test procedure.

In order to build a test system, it is necessary to build a particular bootloader and operating system optimized according this test process. The objective of the bootloader and operating system is to be completely compatible with the different test components. This means to have the correct drivers mounted and ready. Furthermore, it will be interesting to have some tools

that speed up the test process such as memory testers, audio players, video players, stress tools, etc.

5- Develop a software that perform automatically all the necessary test for each different board.

It is important to develop and set up a software system that test the different components and report the results to the user. The software need to be optimized the maximum possible in order to reduce the test time as most as possible. This test need to be adaptive by itself, being able to detect the model of the target board and apply the custom test that correspond to this particular board

6- Build a complete system that allows to follow each test step of each board.

It is necessary to check all the steps of the test process of each board in order to have an exhaustive following of each one and prevent errors and mistakes at factory. It is necessary to set up a database in order to identify each board with an UUID and the date and result of each test.

7- Design a test station in order to speed up the test process at factory.

Last but not least, it is necessary to design a test station in order to speed up the test preparation. This test station has to be modular to add or remove test depending of the board. Furthermore, all the parts of this test station need to be easy to assembly in order that a factory operator will be able to set up as quick as possible.

All these objectives look for two important pillars on this project:

- **To minimize the testing time** in order to reduce the global cost of the manufacturing process.
- **To exhaustively test each component** in order to prevent any incorrect Single-Board Computer to go out to the market reducing the cost that this can generate.

4 Methodology

4.1 Archetype of Embedded Single-Board Computer

The first essential step is to define the archetype of an embedded Single-Board Computer in order to build a modular adaptive test system. In order to illustrate this archetype, it will be used one of the board selected to use in this project: IGEPv2 DM3730 *(IGEP V2 DATASHEET, 2016)* developed by ISEE 2007 S.L:

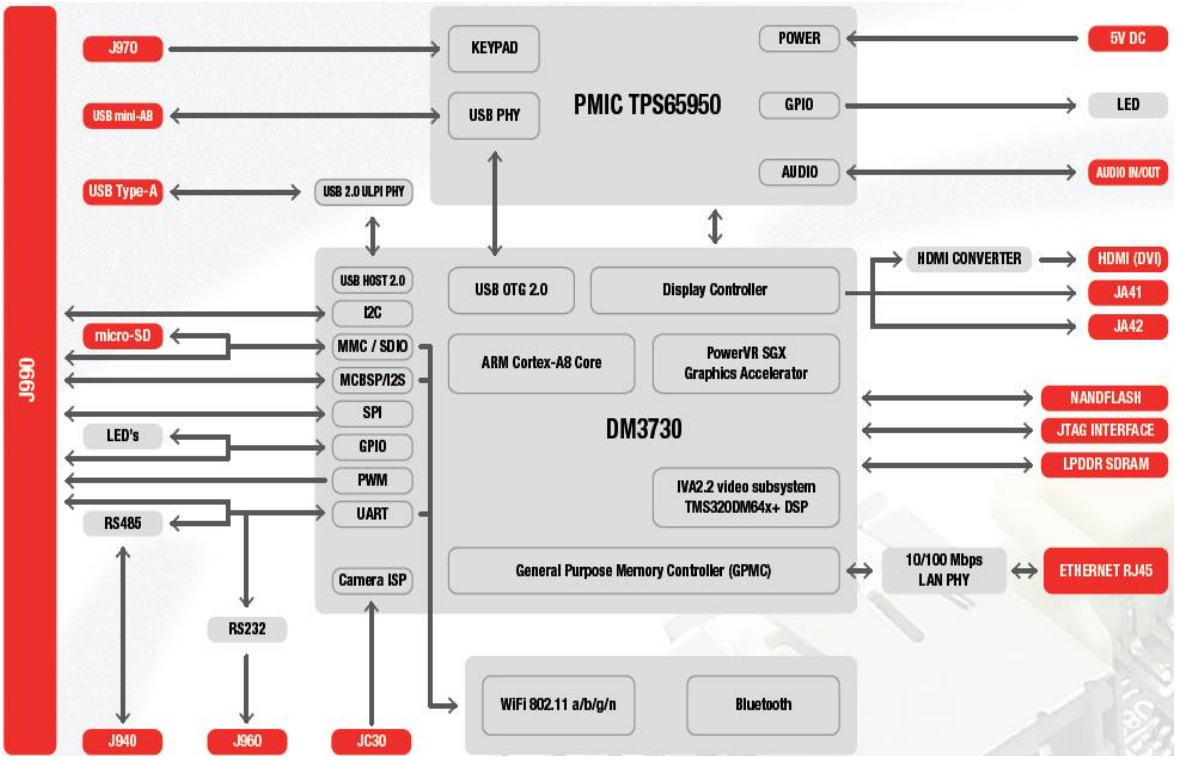


Figure 3: Block Diagram of the IGEPv2 DM3730

The **IGEPv2** is an embedded processor development board based on Texas Instruments OMAP3530/DM3730 processor. It components can perfectly represent the archetype of an embedded Single-Board Computer. The main components are:

- Texas Instruments OMAP3530/DM3730 processor
- RAM 512 MBytes
- NAND Flash 512 Mbytes
- On board micro-SD socket
- Debug RS232 & JTAG
- Connection to single +5V power supply.

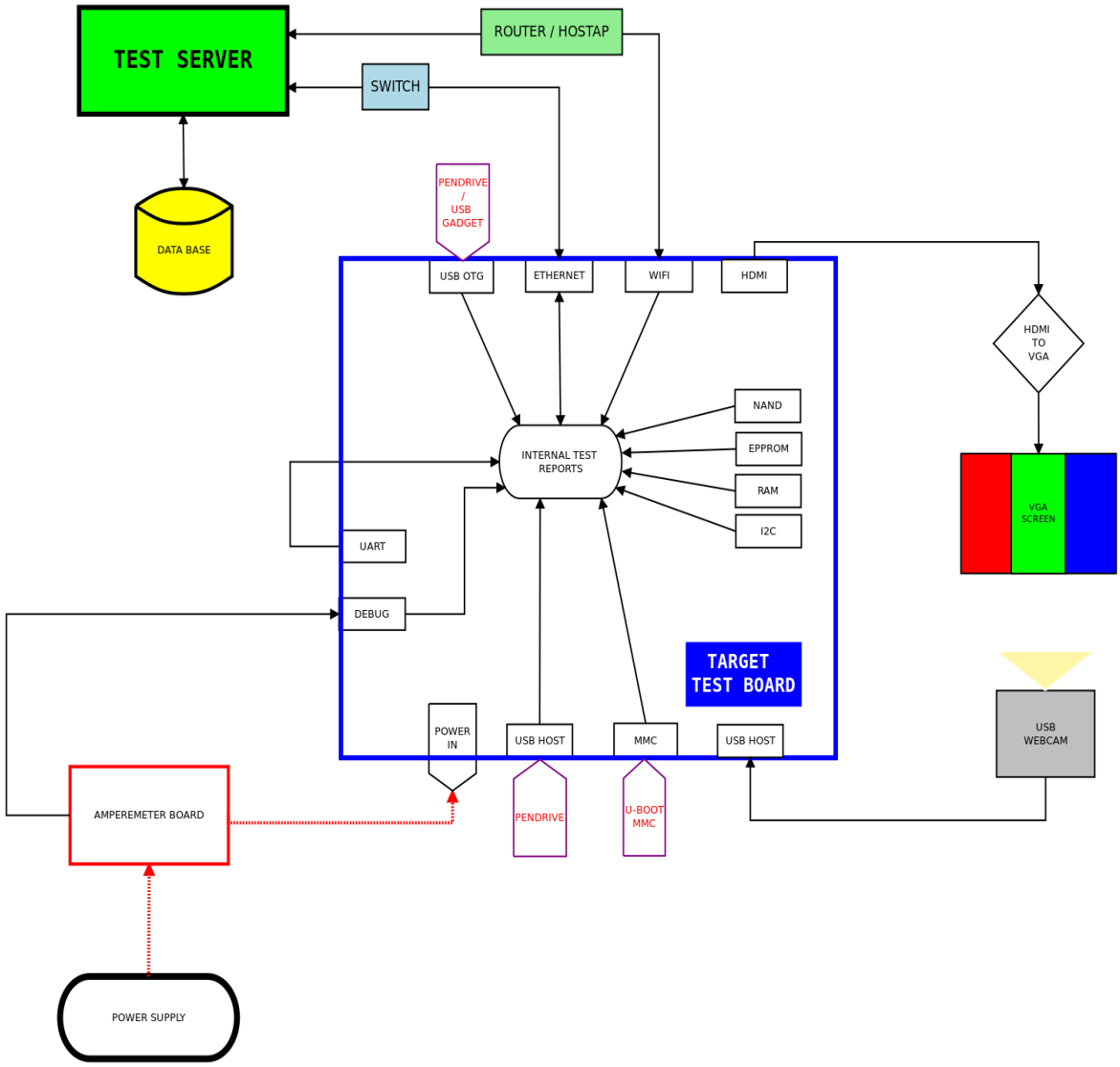
- USB OTG 2.0 on mini AB connector
- USB HOST 2.0 HS
- Ethernet 10/100 Mb
- Audio Stereo Input Jack & Output Jack
- Wi-Fi & Bluetooth
- DVI Video Output
- TFT Interface 24 bit
- LED Indicators

Figure 4: Top and Bottom view of the IGEPv2 board

4.2 Critical Testable Elements and Test on Linux

After define the archetype of an embedded Single-Board Computer it is necessary to start defining all the elements that form part of the system and are going to be tested. It is necessary to define and describe all the elements correctly and the test process that can be followed in Linux operating system to test each one.

In the following image it can be seen the general structure of the test and the elements that compose the system:



4.2.1 Current Consumption

The current consumption test is basically test that the board current consumption is between the desired values (maximum and minimum). This test is critical because a current consumption out of this bounds means that the board is not correctly assembly or there is broken component and obviously the board can't go out to the market and need to be repaired.

In order to check this current consumption, it has been decided to use an external element that will be called "amperemeter". The idea is to supply the board through this amperemeter and use a shunt resistor to measure the potential difference between terminals an in this way the current consumption by dividing it between the shunt resistor value.

A shunt is a device which allows electric current to pass around another point in the circuit by creating a low resistance path. An ammeter shunt allows the measurement of current values too large to be directly measured by a particular ammeter. In this case the shunt is placed in parallel with the moving coil galvanometer, thus all of the current to be measured will flow through it. In order not to disrupt the circuit, the resistance of the shunt is normally very low. The voltage drop across the shunt is proportional to the current flowing through it and as its resistance is known, a voltmeter connected across the shunt can be scaled to directly display the current value.

$$I_{consumption} = \frac{V_{shunt}}{R_{shunt}}$$

In order to get this voltage from the shunt resistor it is necessary to use an analogic to digital (ADC) converter and another element to transform send this value to a CPU and can be analyze by software. After some research it have been decided to use a chip that has everything of this inside and in fact, it is mainly designed to be used as an ammeter. This chip is the INA233 from Texas Instruments.

The INA233 device is a current, voltage, and power monitor with an I 2C-, SMBus-, and PMBus-compatible interface that is compliant with digital bus voltages from 1.8 V to 5.0 V. The device monitors and reports values for current, voltage, and power. The integrated power accumulator can be used for energy or average power calculations. Programmable calibration value, conversion times, and averaging when combined with an internal multiplier enable direct readouts of current in amperes and power in watts. *(INA233 Datasheet, 2017)*

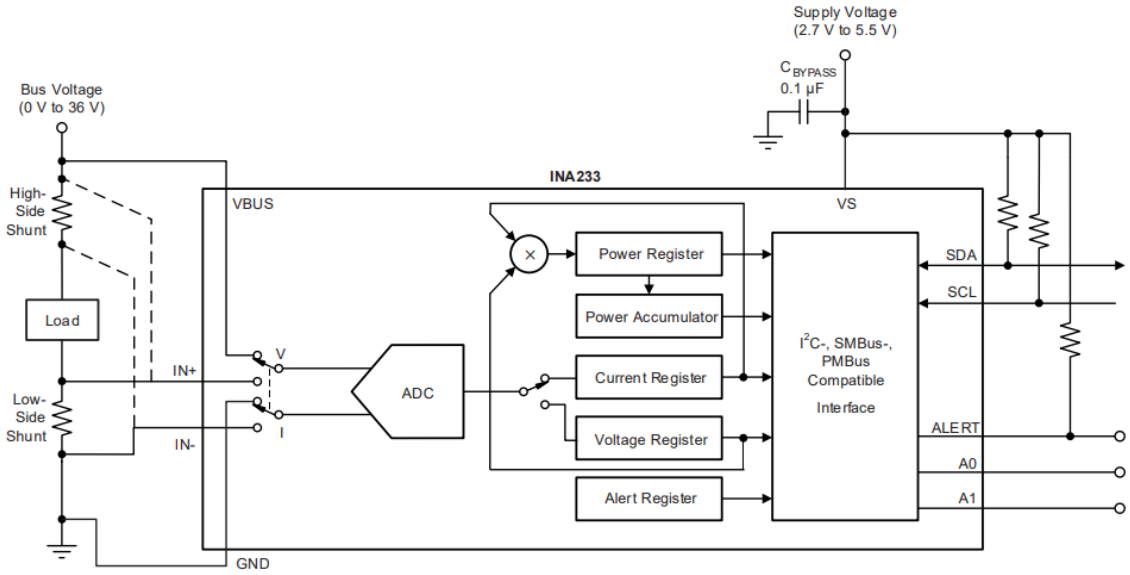


Figure 5: INA233 Sensing Application (http://www.ti.com)

This chip has several programmable registers that are really useful. In the following table they are described:

Table 1: INA233 Registres

CODE	NAME	FUNCTION
03h	CLEAR_FAULTS	Clears the status registers and rearms the black box registers for updating
12h	RESTORE_DEFAULT_ALL	Restores internal registers to the default values
19h	CAPABILITY	Retrieves the device capability
4Ah	IOUT_OC_WARN_LIMIT	Retrieves or stores the output overcurrent warn limit threshold
57h	VIN_OV_WARN_LIMIT	Retrieves or stores the input overvoltage warn limit threshold
58h	VIN_UV_WARN_LIMIT	Retrieves or stores the input undervoltage warn limit threshold
6Bh	PIN_OP_WARN_LIMIT	Retrieves or stores the output overpower warn limit threshold
78h	STATUS BYTE	Retrieves information about the device operating status
79h	STATUS_WORD	Retrieves information about the device operating status
7Bh	STATUS_IOUT	Retrieves information about the output current status
7Ch	STATUS_INPUT	Retrieves information about the input status
7Eh	STATUS_CML	Retrieves information about the communications status
80h	STATUS_MFR_SPECIFIC	Retrieves information about the manufacturer specific device status
86h	READ_EIN	Retrieves the energy reading measurement
88h	READ_VIN	Retrieves the measurement for the VBUS voltage

89h	READ_IN	Retrieves the input current measurement, supports both positive and negative currents
8Bh	READ_VOUT	Mirrors READ_VIN
8Ch	READ_IOUT	Mirror of READ_IN for compatibility
96h	READ_POUT	Mirror of READ_PIN for compatibility with possible VBUS connections
97h	READ_PIN	Retrieves the input power measurement
99h	MFR_ID	Retrieves the manufacturer ID in ASCII characters (TI)
9Ah	MFR_MODEL	Retrieves the device number in ASCII characters (INA233)
9Bh	MFR_REVISION	Retrieves the device revision letter and number in ASCII (for instance, A0)
D0h	MFR_ADC_CONFIG	Configures the ADC averaging modes, conversion times, and operating modes
D1h	MFR_READ_VSHUNT	Retrieves the shunt voltage measurement
D2h	MFR_ALERT_MASK	Allows masking of device warnings
D4h	MFR_CALIBRATION	Allows the value of the current-sense resistor calibration value to be input. Must be programed at power-up. Default value is set to 1.
D5h	MFR_DEVICE_CONFIG	Allows the ALERT pin polarity to be changed
D6h	CLEAR_EIN	Clears the energy accumulator
E0h	TI_MFR_ID	Returns a unique word for the manufacturer ID
E1h	TI_MFR_MODEL	Returns a unique word for the manufacturer model
E2h	TI_MFR_REVISION	Returns a unique word for the manufacturer revision

This registers allows to calibrate the chip depending, the overcurrent alert signal, etc. The problem is that this registers are volatile and it will necessary to use another external element that use i2c bus to program the chip and read the results. A first option could be to use the board that is being tested to communicate with the chip. The problem is that the current consumption test is the most critical test and if the target is not working correctly and present an overcurrent it will impossible to program the overcurrent register and alert the user using the alert signal, for example. For this reason, it going to be used an external element, similar to an Arduino Nano: the ST NUCLEO-F042K6. *(ST NUCLEO-F042K6 Datasheet, 2017)*

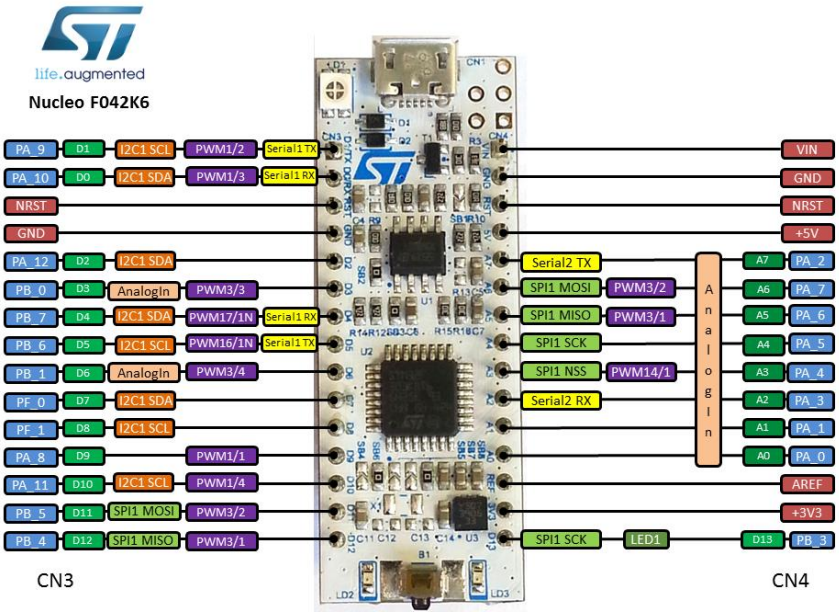


Figure 6: ST NUCLEO-F042K6 board (http://www.monsterelectronics.com)

This board was chosen because of its price and its low power consumption. It is easy to program and has the possibility to communicate with the INA233 through i2c bus and with the board through serial communication and in this way program the overcurrent register to alert the factory operator in case of an overcurrent and communicate the current consumption result to the tested board in order to be analyzed. In the following schema, it is represented the communication process:

The final schematic of the amperemeter is the following four blocks:

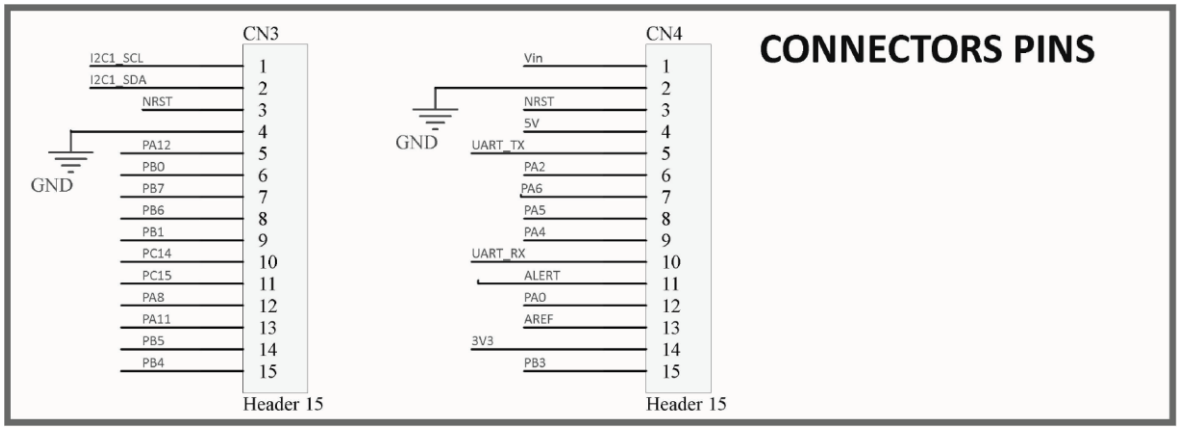


Figure 7: Amperemeter Connector pin with ST module

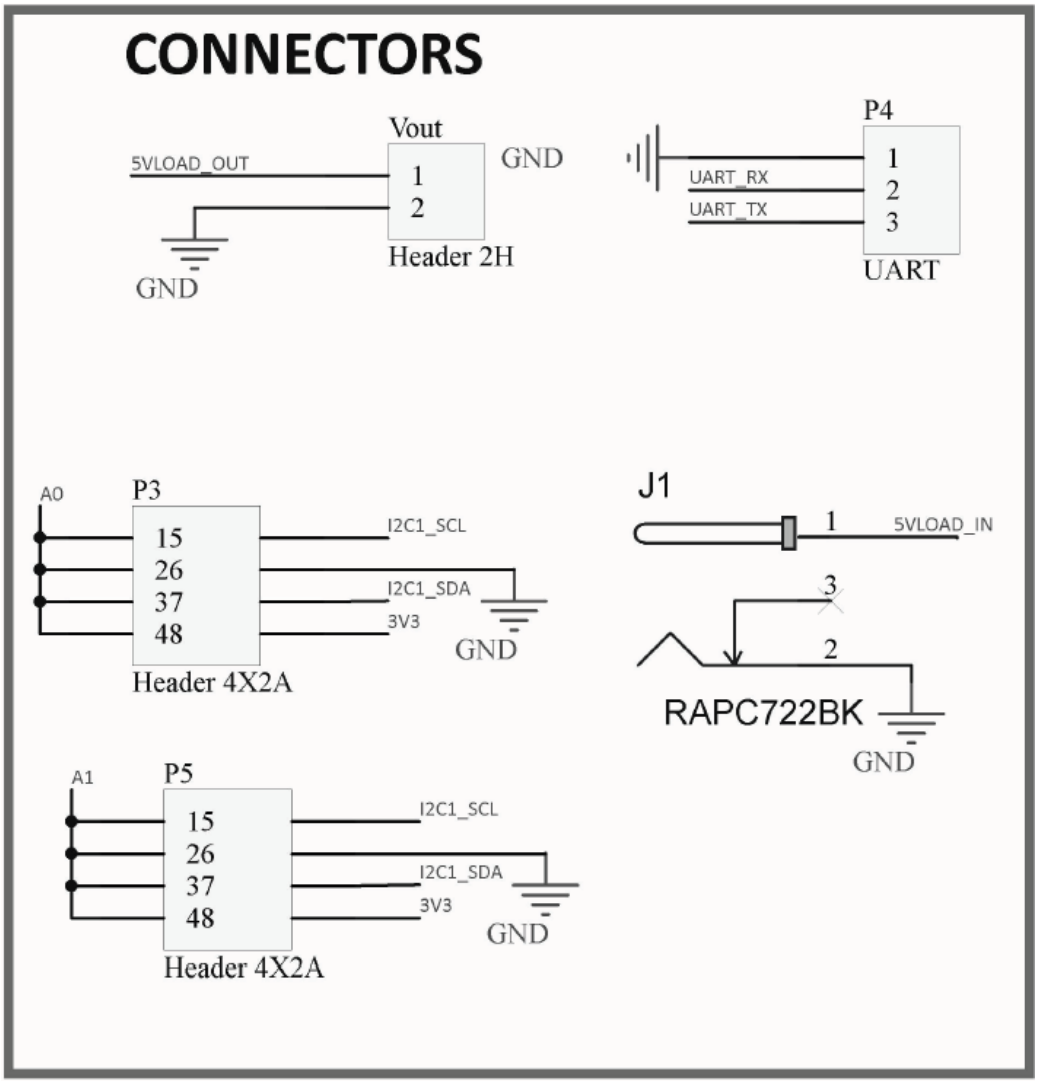


Figure 8: Amperemeter external connector

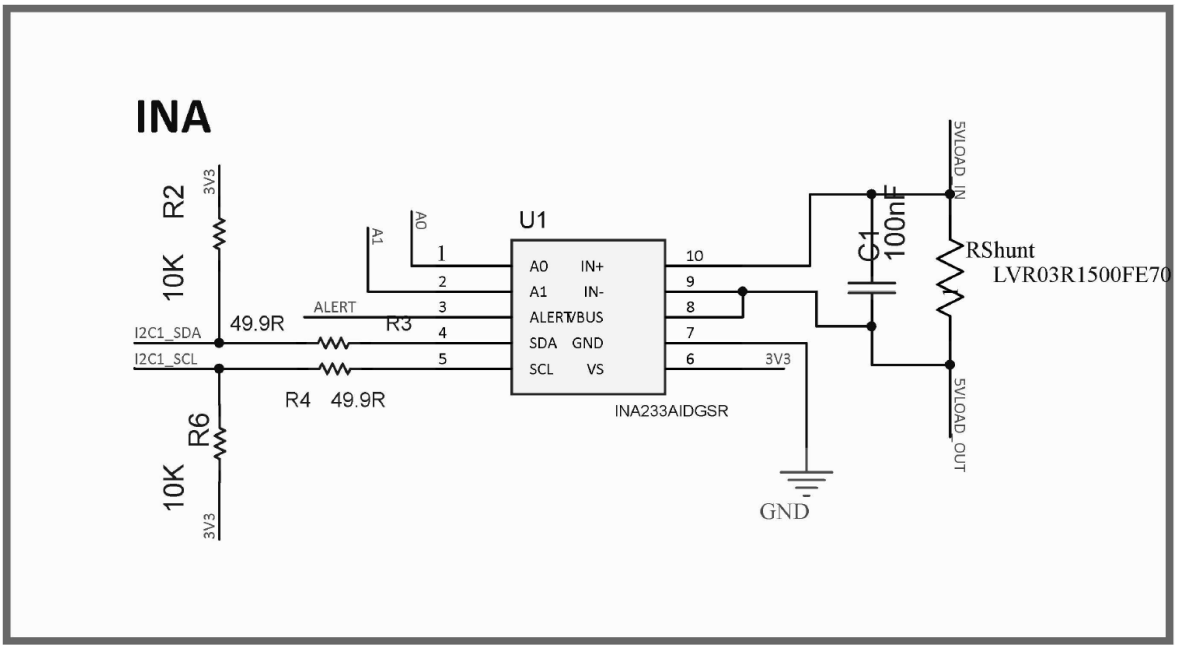


Figure 9: Amperemeter INA233

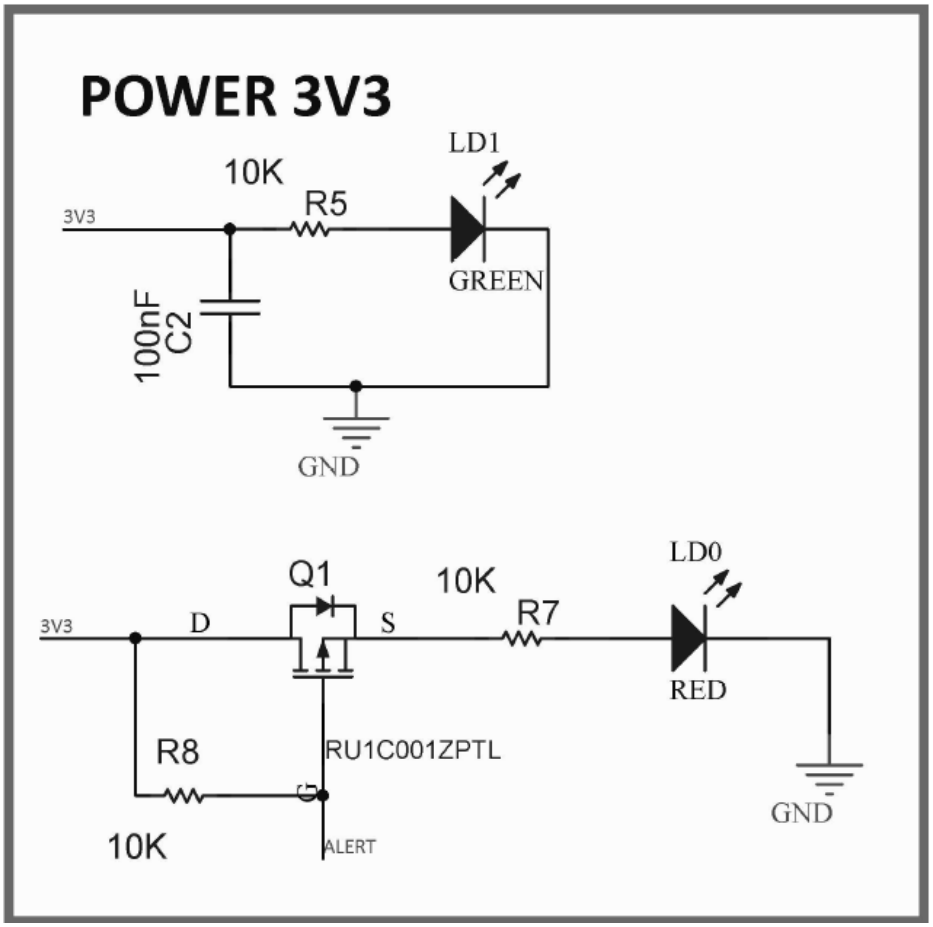


Figure 10: Amperemeter power

4.2.2 Ethernet Connection

Ethernet is the most widely installed local area network (LAN) technology. Ethernet is a link layer protocol in the TCP/IP stack, describing how networked devices can format data for transmission to other network devices on the same network segment, and how to put that data out on the network connection.

Ethernet defines two units of transmission, packet and frame. The frame includes not just the "payload" of data being transmitted but also addressing information identifying the physical "Media Access Control" (MAC) addresses of both sender and receiver, VLAN tagging and quality of service information, and error-correction information to detect problems in transmission. Each frame is wrapped in a packet, which affixes several bytes of information used in establishing the connection and marking where the frame starts.

In the case of Single-Board Computer, the Ethernet technology is using mostly for communications. This communication has to be stable and without interferences. The test has to be able to, in a brief way, evaluate this communication and determine its robustness. It can be differentiated between two classes of Ethernet variants. It is important to well know this variant and it specs in order to build an appropriate test. These variants are generally:

Table 2: Ethernet variants evaluated

Technical Name	Common Name	Speed [Mbit/s]
100BASE-TX	Fast Ethernet	100
1000BASE-T	Gigabit Ethernet	1000

The intention of this test is to use a robust tool that could perform a quick test of this connection. One of the best alternatives is use the free tool called "iperf". Iperf is a widely used tool for network performance measurement and tuning. It is significant as a cross-platform tool that can produce standardized performance measurements for any network. Iperf has client and server functionality, and can create data streams to measure the throughput between the two ends in one or both directions. Typical Iperf output contains a time-stamped report of the amount of data transferred and the throughput measured. *(IPERF, 2018)*

The main idea is to set a server, that could be any device with a Linux OS and a Gigabit Ethernet RJ-45 port, and use iperf tool to set it as a server for the test. We will connect the RJ-

45 cable from the server to a switch or a router that could play the role of a switch. Then we will connect the board RJ-45 to the switch and will use the iperf tool to test the connection and report the results.

The instruction on Linux to obtain the final speed result is:

```
iperf -t2 -c 192.168.2.97 | grep Mb | awk -F MBytes '{print$2}' | cut -d'
' -f3
```

4.2.3 Micro SD card interface

The Micro SD card is used to boot the board. It contains the boot program that it firstly loaded by the processor.

A Micro SD is a type of removable flash memory card used for storing information. SD is an abbreviation of Secure Digital, and Micro SD cards are sometimes referred to as µSD or uSD. The cards are used in mobile phones and other mobile devices.

It is the smallest memory card that can be bought; at 15 mm × 11 mm × 1 mm (about the size of a fingernail), it is about a quarter of the size of a normal-sized SD card. There are adapters that make the small Micro SD able to fit in devices that have slots for standard SD, Mini SD, Memory Stick Duo card, and even USB. But, not all of the different cards can work together. Many Micro SD cards are sold with a standard SD adapter, so that people can use them in devices that take standard SD but not Micro SD cards.

This test is very simple. If we have load the kernel successfully it will mean that the MMC is working correctly.

If we want to perform an exhaustive extra test, stressing the MMC we can use *iozone* software. We can use this command in this way *(IOzone , 2016)*:

```
-R  Generate Excel report

-l #    Lower limit on number of processes to run

-u #    Upper limit on number of processes to run

-r #    record size in Kb
      or -r #k .. size in kB
      or -r #m .. size in MB
      or -r #g .. size in GB

-s #    file size in Kb
      or -s #k .. size in kB
      or -s #m .. size in MB
      or -s #g .. size in GB

-F filenames    for each process/thread in throughput test
```

An example of the command could be:

```
iozone -R -l 1 -u 1 -r 4k -s 20m -F /tmp_iozone_file.txt | tee -a
/tmp/iozone_report.txt
```

It will generate a report in the `/tmp/` folder

4.2.4 EEPROM

EEPROM (also E2PROM) stands for electrically erasable programmable read-only memory and is a type of non-volatile memory used in computers, integrated in microcontrollers for smart cards and remote keyless system, and other electronic devices to store relatively small amounts of data but allowing individual bytes to be erased and reprogrammed.

In our case the EEPROM use the i2c interface to communicate with the processor. The i2c BUS and the address that the EEPROM is using are important parameters to know in order to use the EEPROM.

This test is very simple. It is only necessary to have the Linux EEPROM drive correctly installed and configured

First we have to locale the EEPROM file path, we can do by writing this:

```
cd /sys; find . -iname eeprom
```

Then we can go to this location and read from the EEPROM file:

```
cat eeprom
```

We can write by writing:

```
echo "TESTING-EEPROM" > eeprom
```

Then we can read if it has been successfully written

```
cat eeprom | hexdump -c
```

4.2.5 Internal Memory Device

Usually the Single-Boards Computer present two main internal storage devices to keep the data after the board is disconnected from the power supply: **Flash Memory** (typically NAND Flash) and **eMMC** Depending on the chip processor and the design it will be being used one or the other.

The main idea is to flash the boot software and the Linux OS on the internal storage device

and check if it is correctly flashed. The test will consist in the flashing process into the internal storage device and after that check the integrity of the of data that has been flashed. Depending on the type of the internal memory device this process will be different.

NAND Flash

NAND flash also uses floating-gate transistors, but they are connected in a way that resembles a NAND gate: several transistors are connected in series, and the bit line is pulled low only if all the word lines are pulled high. NAND flash might address by page, word and bit.

In addition, NAND flash is typically permitted to contain a certain number of faults. Manufacturers try to maximize the amount of usable storage by shrinking the size of the transistors. This effect is mitigated in some chip firmware or file system drivers by counting the writes and dynamically remapping blocks in order to spread write operations between sectors; this technique is called wear leveling. Another approach is to perform write verification and remapping to spare sectors in case of write failure, a technique called *bad block* management (BBM). For portable consumer devices, these management techniques typically extend the life of the flash memory beyond the life of the device itself, and some data loss may be acceptable in these applications.

UBIFS (UBI File System, more fully Unsorted Block Image File System) is a filesystem for unmanaged flash memory devices. UBIFS works on top of an UBI (unsorted block image) layer, which is itself on top of a memory technology device (MTD) layer.

UBIFS presents some characteristics that make it the best option to storage data into the NAND Flash:

- UBI/UBIFS scales to large flash sizes better than others filesystems (JFFS2 for example)
- Good fault tolerance, via a number of features
- Built-in on-the-fly compression
- Good runtime performance

The process followed in Linux is the following (using **mtd-utils** package form Texas Instruments) *(MTD Utilities , 2013):*

- Test each NAND partition and mark the bad blocks:

```
nandtest -m /dev/mtd0
nandtest -m /dev/mtd1
nandtest -m /dev/mtd2
nandtest -m /dev/mtd3
```

- Erase each NAND partition:

```
flash_eraseall /dev/mtd0
flash_eraseall /dev/mtd1
flash_eraseall /dev/mtd2
flash_eraseall /dev/mtd3
```

- Now the first loader (MLO) can be flashed in the first partition /dev/mtd0. It is necessary to flash it four times in the first 4 blocks of the NAND:

```
nandwrite -p -s [address] /dev/mtd0 [your_MLO]
```

- Then the **u-boot.img** can be flashed into the second partition /dev/mtd1:

```
nandwrite -p /dev/mtd1 [your_u-boot.img]
```

If some environment is necessary, it can be flashed in the third partition.

- Finally, the rootfs have to be copied using the third partition as an **UBIFS**.
    - First preparing the partition:

```
ubiformat /dev/mtd3
ubiattach -p /dev/mtd3
ubimkvol /dev/ubi0 -N filesystem -m
```

    - Then, copying the files:

```
mkdir /tmp/flashroot
mount -t ubifs ubi0:rootfs /tmp/flashroot/
cd /tmp/flashroot/
tar -xf -C .
cd /
umount /tmp/flashroot
ubidetach -p /dev/mtd3
```

EMMC

The term eMMC is short for "embedded Multi-Media Controller" and refers to a package consisting of both flash memory and a flash memory controller integrated on the same silicon die. The eMMC solution consists of at least three components - the MMC (multimedia card) interface, the flash memory, and the flash memory controller.

The eMMC architecture integrating the flash memory controller in the same package simplifies

the application interface design and frees the host processor from low-level flash memory management. This benefits product developers by simplifying the non-volatile memory interface design and qualification process, resulting in a reduction in time-to-market as well as facilitating support for future flash device offerings.

In other words, eMMC memory hides the complexities of flash memory technology in a convenient "plug-and-play" package. Obviously, this reduces time and effort for developers.

Secondly, eMMC eliminates the need to develop interface software for all types of NAND memory by integrating the embedded controller into the memory chip and providing an easy-to-use memory solutions package for high-speed data transmissions by devices, such as mobile phones. It also eliminates the need for a memory expansion slot by stacking several memory functions vertically, resulting in a very small footprint for the memory devices.

Further, this design makes it possible to select suppliers for subcomponents from a wider base, which results in increased revenues with lower time-to-market.

The flashing process is easier that in the case of a NAND Flash. It is only necessary to partition the device and copy the files. The process that can be used in Linux is the following (in this case it has been tested using a iMX6 based system):

- Create the partitions using **sfdisk**:

```
sfdisk -D -H 255 -S 63 /dev/<eMMC_DEVICE> << THEEND
1,8,0x0C,*
9,,,-
THEEND

mkfs.vfat -F 32 /dev/<eMMC_DEVICE>p1 -n boot
RFS_UUID=`cat /proc/sys/kernel/random/uuid`
mkfs.ext3 -U "$RFS_UUID" "/dev/<eMMC_DEVICE>p2" -L rootfs
```

- Copy the bootloader and the kernel in the first partition:

```
mkdir -p /tmp/<eMMC_DEVICE>p1
mkdir -p /tmp/<eMMC_DEVICE>p2
mount -t vfat /dev/<eMMC_DEVICE>p1 /tmp/mmcblk2p1
mount -t ext3 /dev/<eMMC_DEVICE>p2 /tmp/mmcblk2p2
dd if= of=/dev/<eMMC_DEVICE> bs=512 seek=2 2> /dev/null
cp <kernel_image> /tmp/<eMMC_DEVICE>p1
```

- Copy the rootfs in the second partition:

```
cp –av <rootfs_root>/* /tmp/eMMC_DEVICE>p2
```

4.2.6 Wi-Fi interface

Wi-Fi is a technology for wireless local area networking with devices based on the IEEE 802.11 standards. Wi-Fi compatible devices can connect to the Internet via a WLAN and a wireless access point. Such an access point (or hotspot) has a range of about 20 meters (66 feet) indoors and a greater range outdoors.

To connect to a Wi-Fi LAN, a device has to be equipped with a wireless network interface controller. The combination of computer and interface controller is called a station. For all stations that share a single radio frequency communication channel, transmissions on this channel are received by all stations within range.

The Wi-Fi test has to check to main points:

- The connectivity.
- The range.

The first point, the connectivity, is basically check that the Wi-Fi system is able to stablish connection with a net and start transmitting data. If everything works correctly, it will be ensured that the Wi-Fi interface is working correctly, except for the second point, the range.

Testing the range is the same that checking the Wi-Fi antenna. The basic process is to place the Wi-Fi device far away from the Wi-Fi source. If the signal intensity, in *dbm* is known to this distance the test will consist on checking if the signal intensity that the target test board provide is near the expected signal intensity. If the antenna is broken or bad assembled this intensity will not be correct.

4.2.7 HDMI

HDMI (High-Definition Multimedia Interface) is a proprietary audio/video interface for transmitting uncompressed video data and compressed or uncompressed digital audio data from an HDMI-compliant source device, such as a display controller, to a compatible computer monitor, video projector, digital television, or digital audio device. HDMI is a digital replacement for analog video standards.

The Single-Board Computers used to have this interface. In order to test this interface, it is necessary to display a well-known image and check that the display shows the image correctly. This image does not have to be any image, it has to be an image that display the three main colors red, green and blue in order to check that all the channels are correct and if the

connector is correctly assembled. The image that is going to be used is the following:

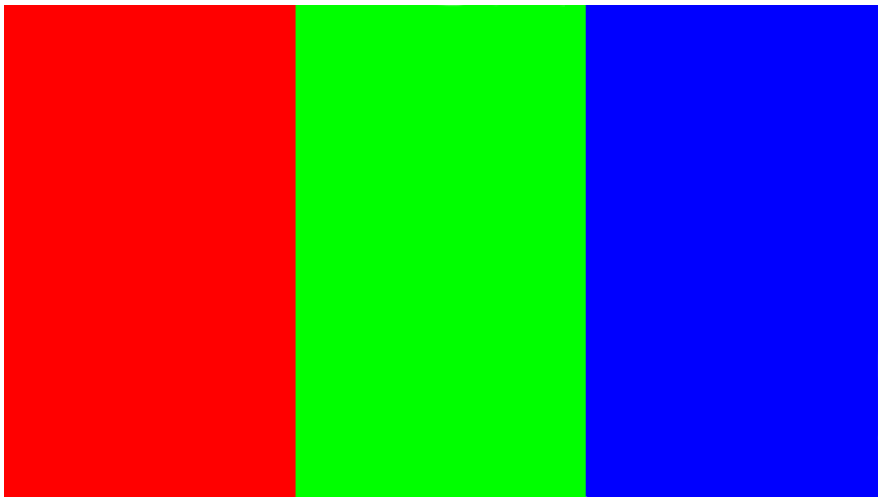


Figure 11: Test pattern image used to test the HDMI

4.2.8 USB Interface

USB (Universal Serial Bus) is the most popular connection used to connect a computer to devices such as digital cameras, printers, scanners, and external hard drives. USB is a cross-platform technology that is supported by most of the major operating systems. USB is a hot-swappable technology, meaning that USB devices can be added and removed without having to restart the computer. USB is also “plug and play”. When you connect a USB device to your PC, the operating system should detect the device and even install the drivers needed to use it.

Another USB feature is the use of direct current (DC). In fact, several devises use a USB power line to connect to DC current and do not transfer data. Example devices using a USB connector only for DC current include a set of speakers, an audio jack and power devices like a miniature refrigerator, coffee cup warmer or keyboard lamp.

USB On-The-Go, often abbreviated to USB OTG or just OTG, is a specification. Use of USB OTG allows those devices to switch back and forth between the roles of host and device. For instance, a mobile phone may read from removable media as the host device, but present itself as a USB Mass Storage Device when connected to a host computer.

In other words, USB OTG introduces the concept of a device performing both **master and slave roles**, whenever two USB devices are connected and one of them is a USB OTG device, they establish a communication link. The device controlling the link is called the master or host, while the other is called the slave or peripheral.

The initial role of each device is defined by which mini plug a user inserts into its receptacle. In most of the cases of Single-Board Computers use to keep at least one USB as HOST role and one USB as OTG role by joining or not the USB ID pin to ground. The test of the USB interface will be vary depending on the role of the USB.

USB HOST

The test of the USB interface with the role of HOST will consist into plug in a pendrive USB with a created text file and check if the system is able to open this file and read the content. A more exhaustive test could be performed by using the iozone tool. For example:

```
iozone -R -l 1 -u 1 -r 4k -s 50m -F /run/media/sda1/tmpdummyfile.txt | tee
-a /tmp/iozone_results.txt
```

USB OTG

The test of the USB interface with the role of OTG will consist into use the USB Ethernet Gadget. The Linux kernel has a class of drivers called USB Gadgets that allow you to use USB as a transport for a number of different protocols like serial, virtual file systems and Ethernet devices.

The USB OTG connector supports using the Ethernet Gadget to create a networking device over USB. From the client's (and host's) perspective this driver simply appears as another Ethernet device, enabling us to have Ethernet access on the board by hooking them up to a desktop, so this USB slave device can communicate with a properly configured USB host.

It should have the usbnet module available in the Linux kernel. On most distributions, this is the case by default.

In order to test it, is only matter of activating the net over USB and use the *iperf* tool in order to test the communication. Another option is to connect the board with the HOST computer and stablish a SSH communication. A little example could be:

In the file `/etc/network/interfaces`, it should be added:

```
iface usb0 inet dhcp
```

Awake the device and check the results:

```
ifconfig usb0 up

ifconfig
```

```
usb0      Link encap:Ethernet  HWaddr 66:cc:f2:be:51:b6
          inet addr:192.168.7.10  Bcast:192.168.7.255  Mask:255.255.255.0
          inet6 addr: fe80::64cc:f2ff:febe:51b6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:44 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2496 (2.4 KB)  TX bytes:8794 (8.7 KB)
```

Figure 12: USB Ethernet gadget connection

4.2.9 UART

A universal asynchronous receiver-transmitter (UART) is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. The electric signaling levels and methods are handled by a driver circuit external to the UART. One or more UART peripherals are commonly integrated in microcontroller chips.

The universal asynchronous receiver-transmitter takes bytes of data and transmits the individual bits in a sequential. At the destination, a second UART re-assembles the bits into complete bytes. Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information (bits) through a single wire or other medium is less costly than parallel transmission through multiple wires.

The UART usually does not directly generate or receive the external signals used between different items of equipment. Separate interface devices are used to convert the logic level signals of the UART to and from the external signaling levels, which may be standardized voltage levels, current levels, or other signals.

Communication may be simplex (in one direction only, with no provision for the receiving device to send information back to the transmitting device), full duplex (both devices send and receive at the same time) or half duplex (devices take turns transmitting and receiving).

In order to test the UART it is necessary to stablish communication with another UART device. The simplest case is to communicate two Single-Board Computers through its UARTs. A text-based modem control and terminal emulation program have to be used. In the following case it is used the software *Minicom*. It is important to cross the lanes as the following image shows:

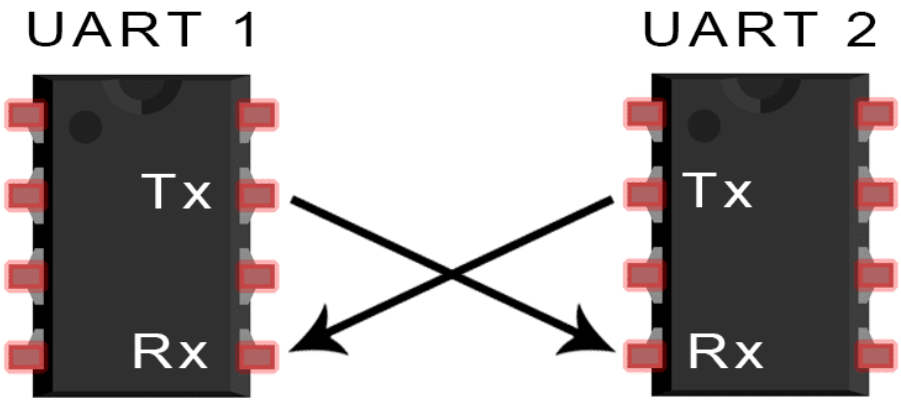


Figure 13: Two UARTs connection diagram (http://www.circuitbasics.com/)

In both boards the following command has to be written and the communication will be stablished:

```
minicom -s 115200 /dev/<your_uart_port>
```

4.2.10 I2C

I2C is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus. It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

It could be found several devices connected to the i2c bus. Some examples are:

- PMIC
- EEPROM
- AUDIO
- CAMERA

The objective of this test is to check if the i2c bus is working correctly and there exist communication with devices attached to it.

In this case we will use the i2c-tools to interact with the i2c bus of the board. We are interested in detect and check the devices in the correct bus and address. We start from the point that for every board we would know where each i2c slave device is connected and its address.

In order to check that the i2c interface is correctly working it is necessary to check all the i2c

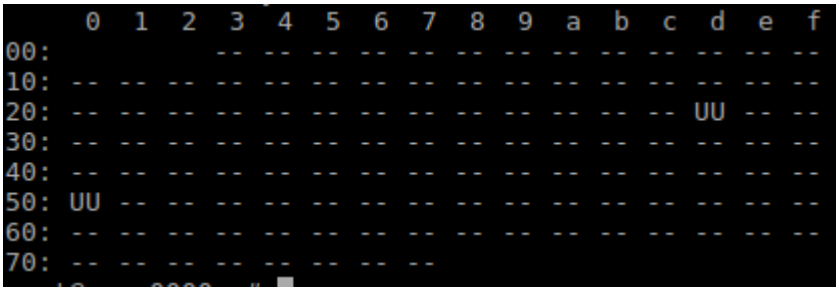
bus presented in the board. For example, there are a PMIC and an EEPROM on the i2c1 and RTC in the i2c2. The following process show how can be tested that the buses are correctly working and it is able to stablish communication with the slaves:

i2C channel 1

Check that the direction 0x2f (PMIC) and 0x50 (EEPROM) are busy (UU). To do that we use first the command:

```
i2cdetect -r 1
```

, and obtain:

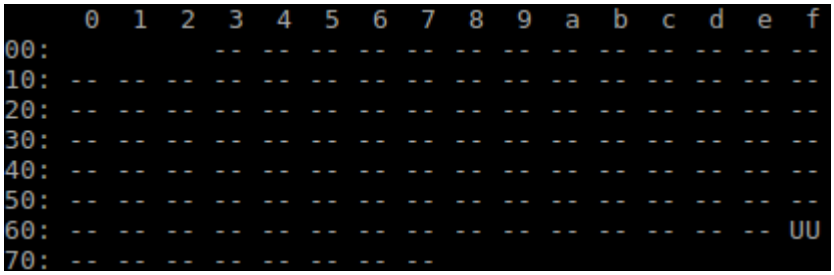


i2C channel 2

Check that the direction 0x2f (RTC) is busy (UU). To do that we use first the command:

```
i2cdetect -r 2
```

, and obtain:



4.2.11 Audio Interface

Usually, the boards have audio platforms. There are several modules that can contain audio interface. We can difference this audio interface as inputs and output.

The most common modules that contain audio interfaces in the boards are the HDMI and the normal audio interface (usually using 3.5mm Audio Jack connectors)

The easiest way to test the audio on Linux is to use the ALSA software. Advanced Linux Sound Architecture (ALSA) is a software framework and part of the Linux kernel that provides an application programming interface (API) for sound card device drivers. *(Advanced Linux Sound Architecture Wiki, 2018)*

The easiest way to test the audio is to create a loop-back between an audio input and an audio output. If the board has two jack connector, one for an input and one for an output, it would be only needed a male-male jack stereo audio cable connected between the audio input and the audio output creating a loop-back.

Once the loop-back is created, we can play a known audio file meanwhile it is recorded by the audio input and check that the file played is the same that the file recorded. The best file to use is a dtmf file. Dual-tone multi-frequency signaling (DTMF) is an in-band telecommunication signaling system using the voice-frequency band over telephone lines between telephone equipment and other communications devices and switching centers. We can decode the recorded file and check the decode output match with the correct one.

There are several ways to generate a known DTMF audio file. One of the easiest is to use the following website: https://www.audiocheck.net/aud iocheck_dtmf.php. There also exist a dree tool to decode DTMF wav files. It is called multimon.

We can use the following commands to test the audio using the previous procedure:

1- Check the available options.

Here we will find the state [ON/OFF] of each device, the level, etc. The name that it will be presented here is the same that the name it has been given in the system device tree:

```
amixer
```

2- Select target device

```
amixer -c 0 sset '<Target Device>' <parameter>
```

3- Play and record the dtmf file:

```
aplay <target-file> & arecord -r <desired-frequency> -d <duration> <output-file>
```

4- Decode the recorded file

```
multimon -t wav -a DTMF <target-wav-file>
```

5 System Description

The test system will be completely made to measure. From the bootloader to the Linux kernel. In this chapter it will be described the software parts that form part of the test system.

5.1 General Structure

The test system is basically a piece of software running in a Linux operating system. The key point in here, is that the operating system, bootloader, Linux kernel and root file system, are made to measure. By doing this, we can create a powerful and efficient system.

For example, if we want to test the correct performing of the CPU we can build the kernel to allow us to overclock it and test its different clock frequencies.

5.2 Operating System

It can be understood the operating system as the combination of the bootloader, the kernel and the root file system. In the following image it can be seen a brief schema of the workflow of an operating system and the files that represent it:

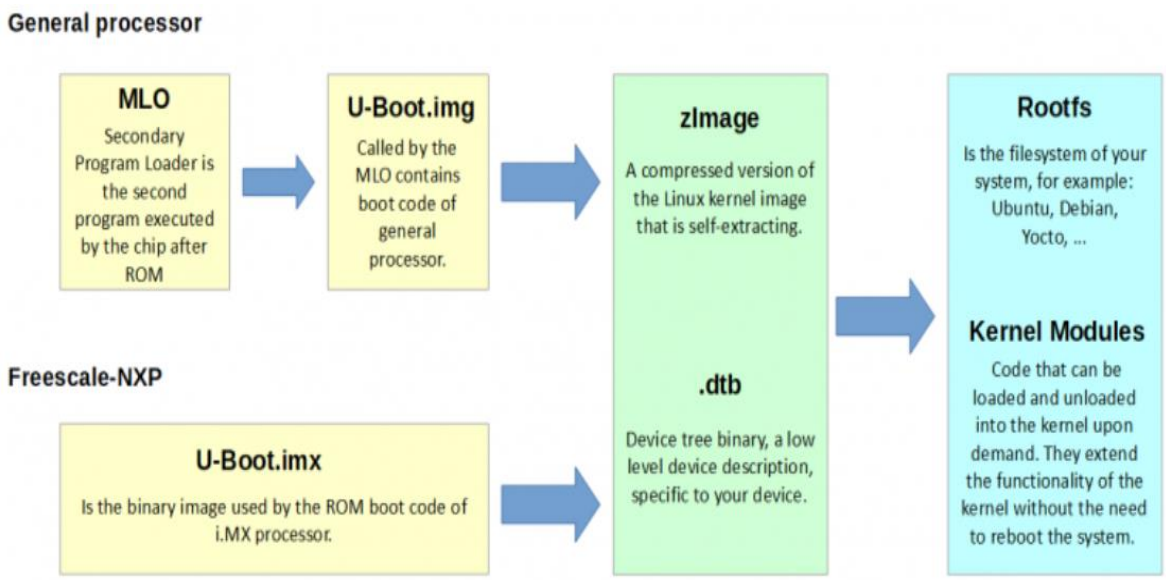


Figure 14: Operating system workflow

5.2.1 U-Boot

U-Boot is an open source, primary boot loader used in embedded devices to package the

instructions to boot the device's operating system kernel. It is available for a number of computer architectures, including obviously ARM architecture.

U-Boot is both a first-stage and second-stage bootloader. It is loaded by the system's ROM or BIOS from a supported boot device, such as an SD card, SATA drive, NOR flash (using SPI or I²C), or NAND flash. If there are size constraints, U-Boot may be split into stages: the platform would load a small SPL (Secondary Program Loader), which is a stripped-down version of U-Boot, and the SPL would do initial hardware configuration and load the larger, fully featured version of U-Boot. Regardless of whether the SPL is used, U-Boot performs both first-stage (e.g., configuring memory controllers and SDRAM) and second-stage booting (performing multiple steps to load a modern operating system from a variety of devices that must be configured, presenting a menu for users to interact with and control the boot process, etc.).

U-Boot runs a command-line interface on a serial port. Using the console, users can load and boot a kernel, possibly changing parameters from the default. There are also commands to read device information, read and write flash memory, download files (kernels, boot images, etc.) from the serial port or network, manipulate device trees, and work with environment variables (which can be written to persistent storage, and are used to control U-Boot behavior such as the default boot command and timeout before auto-booting, as well as hardware data such as the Ethernet MAC address).

Unlike PC bootloaders which obscure or automatically choose the memory locations of the kernel and other boot data, U-Boot requires its boot commands to explicitly specify the memory addresses as destinations for copying data (kernel, ramdisk, device tree, etc.) and for jumping to the kernel and as arguments for the kernel. Because U-Boot's commands are fairly low-level, it takes several steps to boot a kernel, but this also makes U-Boot more flexible than other bootloaders, since the same commands can be used for more general tasks. It's even possible to upgrade U-Boot using U-Boot, simply by reading the new bootloader from somewhere (local storage, or from the serial port or network) into memory, and writing that data to persistent storage where the bootloader belongs.

U-Boot has support for USB, so it can use a USB keyboard to operate the console (in addition to input from the serial port), and it can access and boot from USB Mass Storage devices such as SD card readers.

In order to prepare the U-Boot it is necessary to download and compile it for the correct architecture. In order to do it, it can be used a CROSS COMPILER.

Once the U-Boot has been downloaded from the source it can be loaded the default

configuration file, called defconfig (default configuration) that variates in function of each model. In order to load the default configuration of the target board, it is necessary to write the following command:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- [defconfig]
```

After running the defconfig, it is necessary to build the U-Boot by executing this command:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

Once the compile process has successfully finished, the resulting files inside the u-boot root folder can be found.

Depending on the processor the generated files will be different:

- The result for ARM processor based board will be a **MLO** and **u-boot.img**.
- The result for Freescale-NXP processor based will be a **u-boot.imx**.

Finally, it can be copied this file in a boot device, for example, a microSD Memory Card and power on the board.

5.2.2 Kernel

The Linux kernel is an open-source monolithic Unix-like computer operating system kernel. The Linux family of operating systems is based on this kernel and deployed on both traditional computer systems such as personal computers and servers, usually in the form of Linux distributions, and on various embedded devices.

The Linux kernel API, the application programming interface (API) through which user programs interact with the kernel, is meant to be very stable and to not break userspace programs (some programs, such as those with GUIs, rely on other APIs as well). As part of the kernel's functionality, device drivers control the hardware; "mainlined" device drivers are also meant to be very stable. However, the interface between the kernel and loadable kernel modules (LKMs), unlike in many other kernels and operating systems, is not meant to be very stable by design.

In order to compile the Linux kernel, it is necessary to build and load a configuration. The following steps can be followed in order to compile the Linux Kernel using a Cross Compiler.

First of all, the default configuration (defconfig) has to be loaded. It can be done by writing the

following command:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- [your_defconfig]
```

Once the default configuration has been loaded, it is time to compile the Linux Kernel. There are three important elements to compile:

- Image: The kernel image. There are three several formats. Generally, it will be used the zImage (a compressed version of the Linux kernel image that is self-extracting)
- DTBs: Device tree binary, a low level device description, specific to your device.
- Modules: Kernel Modules, pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system.

In order to compile this three elements, it is necessary to write:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage dtbs modules
```

Then, it will find the resulting compiled Kernel in the arch/arm/boot folder:

- **zImage**: located in arch/arm/boot/zImage
- **DTB**: located in arch/arm/boot/dts/.dtb. Depending of the board the dtb file will be different.

Finally, the last step is to install the modules inside the desired rootfs. It can be done by typing:

```
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- INSTALL_MOD_PATH=
modules_install
```

For example, if the rootfs is located in an external storage device mounted on the */media folder*:

```
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
INSTALL_MOD_PATH=/media/rootfs/ modules_install
```

5.3 Test Software

The idea behind this project is to perform a complete functional test of the board that runs automatically when the user switch on the board. At this point, the next step is to develop a software application that will be able to run on Linux operating system.

The language that will be used in order to develop this application is Python. Python is an interpreted high-level programming language for general-purpose programming. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems, including Linux operating system.

Figure 15: Python language logo (www.python.org)

In addition, in order to manage the data, the test control and the implementation it will be used PostgreSQL database. PostgreSQL is an object-relational database management system with an emphasis on extensibility and standards compliance. As a database server, its primary functions are to store data securely and return that data in response to requests from other software applications, in this case Python software. It can handle workloads ranging from small single-machine applications to large Internet-facing applications with many concurrent users. It is also available for Linux and another different operating system.

Figure 16: PostgreSQL database logo (www.postgresql.org)

5.3.1 Python 3 Software

The test software implement the Linux function detailed in the precious chapter into Python language. It will be used Python 3.5 version because the support of python 2.7 is ending on 2020.

GENERAL STRUCTURE

The first step is to define the general structure of the Python software project. There will be four main folders:

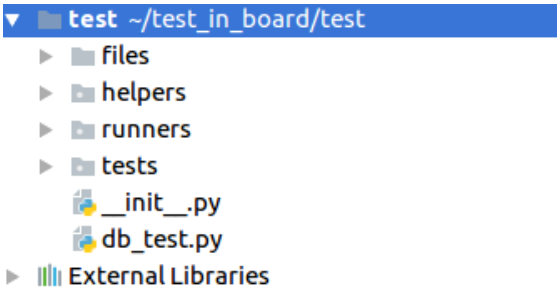


Figure 17: General structure of the software project

As it can be seen, there are three programs in the main folder there is a program: *db_test.py*. This file is the main program of the software application. It is the one that is automatically executed in the INIT of Linux.

The workflow of this program is the following:

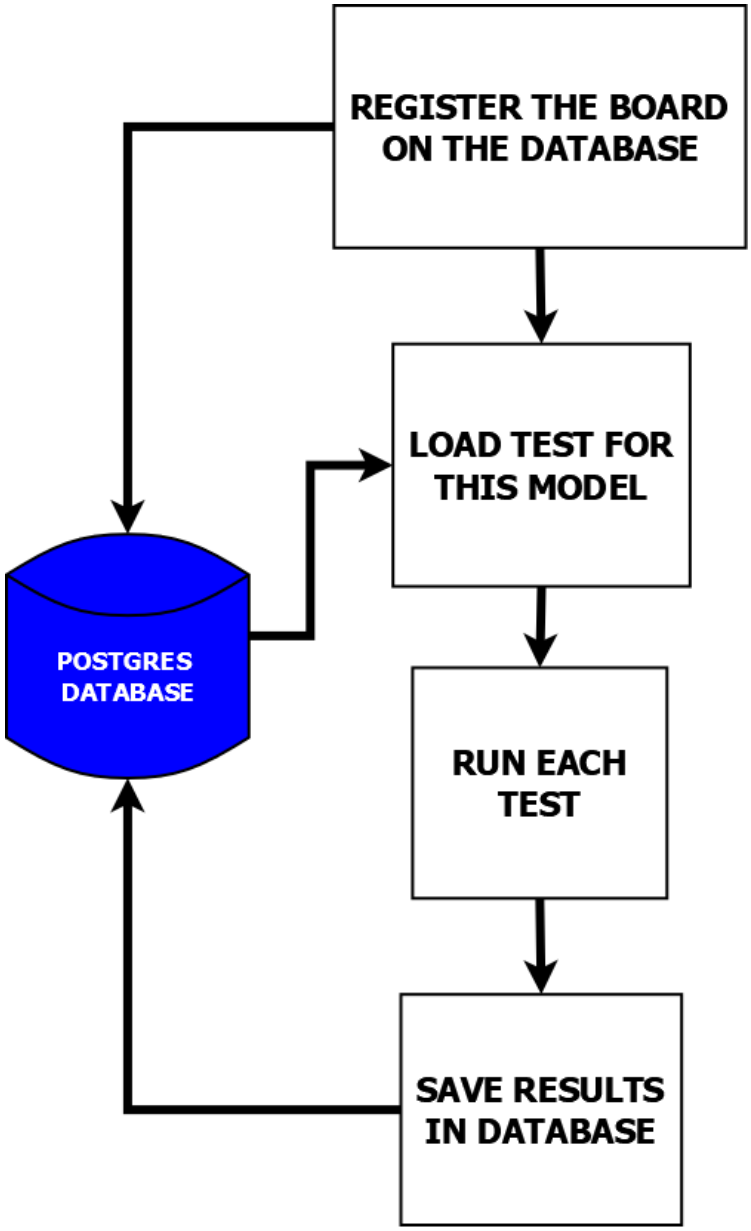


Figure 18: Main program workflow

The first folder that appear in the project is the "*files*" folder. In this folder it will be found all the

external files that are needed to perform the test, for example the *dtmf* file used in the audio test. In this folder it also will be found xml files used for diverse tasks, for example, configure the setup of the database.

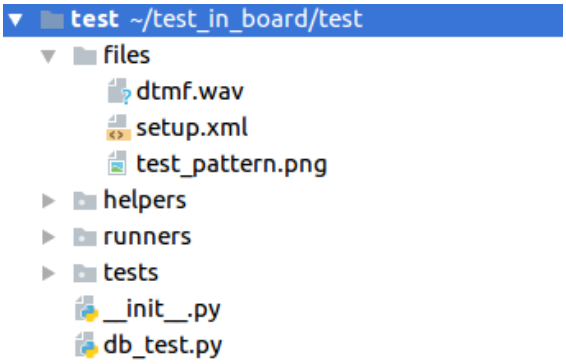


Figure 19: "files" folder expanded

The following folder is the "*helpers*" folder. Inside this folder there are all the programs that are not a dedicated test program and are used as external help functions for the rest of the program. For example, there is a function to read the unitary processor id of a board or a function to send the commands to the database.

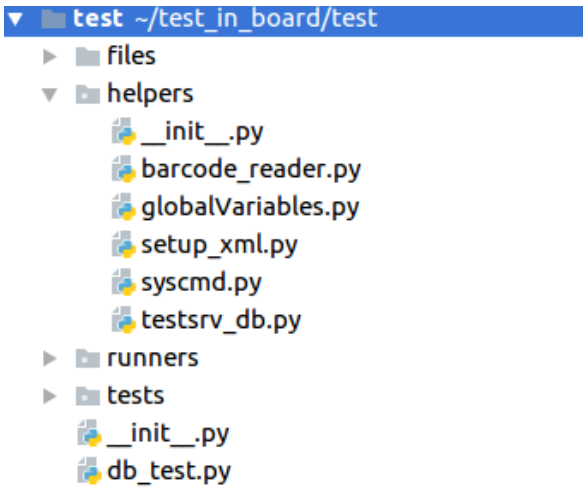


Figure 20: "helpers" folder expanded

The next folder is the "*runners"* folder. This folder has inside the programs that allow to add and organize the different tests (loaded from the database) of the board in a class where all the test will have the same structure: name, result and message.

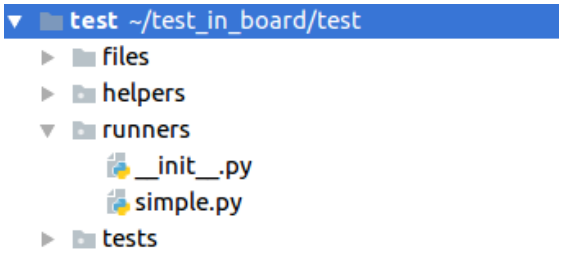


Figure 21: "runners" folder expanded

Finally, the "*tests*" folder will contain all the different unitary test that exists in the project. As it has been said before, these tests are implementation of the proposed Linux test.

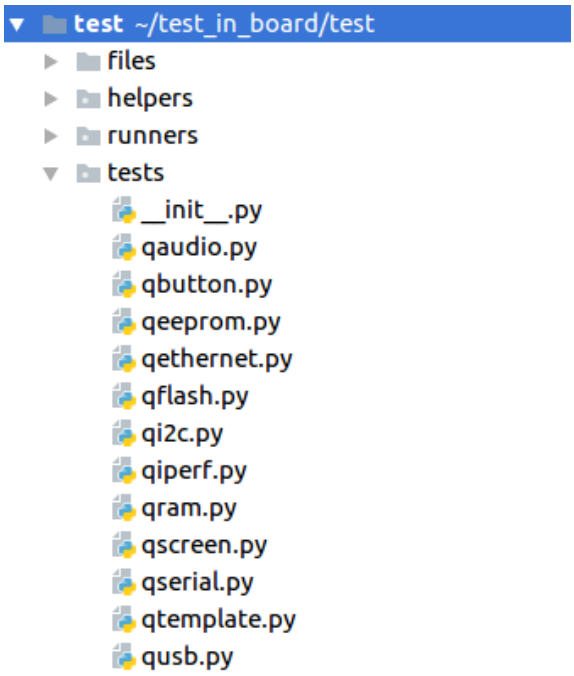


Figure 22: "tests" folder expanded

LIBRARIES

One of the advantages of using Python is the huge variety of libraries than can be used to interact with the system. Now, it will be described the libraries that are used in this project.

Unittest

The *unittest* unit testing framework was originally inspired by JUnit and has a similar flavor as major unit testing frameworks in other languages. It supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework. *(Unit testing framework, 2018)*

The *unittest* module provides a rich set of tools for constructing and running tests. In this project, the *unittest* library is not used to check the code but it is used to organize the different tests and its arguments.

subprocess

The *subprocess* module allows to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. This module intends to replace several older modules and functions, for example the *os.system* that presents several limitations when it is used to execute demanding system commands. *(Subprocess management, 2018)*

xml.etree.ElementTree

The Element type is a flexible container object, designed to store hierarchical data structures in memory. The type can be described as a cross between a list and a dictionary. *(The ElementTree XML API, 2018)*

Each element has a number of properties associated with it:

- a tag which is a string identifying what kind of data this element represents (the element type, in other words).
- a number of attributes, stored in a Python dictionary.
- a text string.
- an optional tail string.
- a number of child elements, stored in a Python sequence

psycopg2

Psycopg is the most popular PostgreSQL database adapter for the Python programming language. Its main features are the complete implementation of the Python DB API 2.0 specification and the thread safety. It was designed for heavily multi-threaded applications that create and destroy lots of cursors and make a large number of concurrent INSERTs or UPDATEs.

Psycopg 2 is mostly implemented in C as a libpq wrapper, resulting in being both efficient and secure. It features client-side and server-side cursors, asynchronous communication and notifications, COPY support. Many Python types are supported out-of-the-box and adapted to matching PostgreSQL data types; adaptation can be extended and customized thanks to a flexible objects adaptation system. *(PostgreSQL database adapter for Python, 2016)*

pySerial

This module encapsulates the access for the serial port. It will be used to communicate and test the different serial devices in a deep way. (*pySerial's documentation, 2015)*

TEST TEMPLATE

It has been exposed that the test file programs inside the "*tests*" folder are implementations of the different proposals used in Linux. Using python, it is necessary to use an especial library to execute complex Linux commands. The aim of this is to replicate the test proposed in the previous chapter and create functions that automatically analyze the results given and return a result. All the different tests, except the HDMI and the Serial test) will use the following structure:

1- Import the modules and dependencies

```
1   #Import necessary modules
2   from helpers.syscmd import SysCommand
3   import unittest
```

2- Define the test class

```
6   # Define a class
7   class Qtemplate(unittest.TestCase):
8       # Initialize the variables
9       __variable1 = "Value-a"
10      __variable2 = "Value-b"
11      #....
12      __variablen = "Value-n"
```

3- Initialize the class and execute the desired function inside the class

```
14      # Class initializer
15      def __init__(self, testname, testfunc, input1=None, inputn=None):
16          # Doing this we will initialize the class and
17          # later on perform a particular method inside this class
18          super(Qtemplate, self).__init__(testfunc)
19          self.__testname = testname
20          self.__input1 = input1
21          self.__inputn = inputn
22          self._testMethodDoc = testname
```

4- Execute the desired command

```
24      # class execution
25      def execute(self):
26          str_cmd = "corresponding command string"
27          command = SysCommand("command-name", str_cmd)
28          if command.execute() == -1:
```

5- Analyze the returned results from the command

```
29              self.__raw_out = command.getOutput()
30              # Generate a result from the command output
31              if self.  raw out != "OK":
```

6- Give a result of the test

```
31              if self.__raw_out != "OK":
32                  self.fail("failed: test failed because...")
33          else:
34              self.fail("failed: executing command")
```

5.3.2 POSTGRESQL Database

The PostgreSQL database is used to storage the test data. It is composed by a combination of tables and functions. Firstly, it is important to give a clear definition of this elements:

> **A table** in a relational database is much like a table on paper: It consists of rows and columns. The number and order of the columns is fixed, and each column has a name. The number of rows is variable. When a table is read, the rows will appear in an unspecified order, unless sorting is explicitly requested. Each column has a data type. The data type constrains the set of possible values that can be assigned to a column and assigns semantics to the data stored in the column so that it can be used for computations.
>
> PostgreSQL includes a sizable set of built-in data types that fit many applications. Users can also define their own data types. Most built-in data types have obvious names and semantics. Some of the frequently used data types are integer for whole numbers, numeric for possibly fractional numbers, text for character strings, date for dates, time for time-of-day values, and timestamp for values containing both date and time.

> User-defined functions in programming PostgreSQL language are routines that accept parameters, perform an action, such as a complex calculation, and return the result of that action as a value. The return value can either be a single scalar value or a result set.
>
> They allow modular programming. It can be created the function once, store it in the database, and call it any number of times in the program. User-defined functions can be modified independently of the program source code. They also allow faster execution. User-defined functions reduce the compilation cost of the code by caching the plans and reusing them for repeated executions. This means the user-defined function does not need to be reparsed and re-optimized with each use resulting in much faster execution times. (PostgreSQL Documentation, 2018)

There are four main tables created and its associated functions that allow to get and extract data from them using relational indentificators between tables:

BOARD TABLE

This table will contain the different unique identification off each table and the time stamp of its register. In this way each board will be stored and will be easily accessible by its unique identification in order to identify it in other tables.

TEST DEFINITATION TABLE

This table will contain all the different test that exist. It will be given a unique identification in order to identify them in other tables.

TEST RESULTS

This table will contain the historical of all the test results that have been register in the database. This data will be represented by the board unique identification, the test identification, the timestamp and the result of each test with its observations.

TEST PER MODEL

This table will contain the test that have to be applied to each model of Single-Board Computer contemplate. For example, if two models are similar but one has Wi-Fi support and the other not, the table will define different tests for the two models.

5.4 OpenCV and Display Test

In order to check the correct performance of the HDMI in an automatic way, without no human interaction, it is necessary to develop a special test. This test is quite different from the other ones because it is a complex algorithm that used OpenCV libraries for python.

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

By mean of using OpenCV and Python it will be developed an application that will take a picture of a pattern image that is being displayed in the test screen through HDMI port. Then, it will process this picture and detect that the picture taken is similar to the image that is being displayed.

INSTALLING OPENCV 3.5 IN ARM PRROCESSOR BOARDS

One of the first problem that appear when it is necessary to work with the last version of OpenCV and the last version of Python in an ARM platform is the installation. When it is being used a typical platform, like a standard computer, the last version this software can be easily installed using the standard installer (pip, apt, …). In the case of this project, in order to make OpenCV 3 work with Python 3.5 it is necessary to compile it, and set the compilation flags correctly. *(OpenCV documentation, 2018)*

After downloading OpenCV last version from the official web page, this steps have to be followed in other to get this software working on ARM processor board:

Building OpenCV from Source Using CMake

Create a temporary directory, where it will be put the generated Makefiles, project files as well the object files and output binaries and enter there.

Using *cmake* it can be configured build flags, prefix, etc. For example:

```
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local ..
```

The parameters that are important are:

- PYTHON3_EXECUTABLE = <path to python>

- PYTHON_INCLUDE_DIR = /usr/include/python<version>
- PYTHON3_NUMPY_INCLUDE_DIRS = /usr/lib/python<version>/dist-packages/numpy/core/include/

Finally, the last step is to build everything:

```
make

make  install
```

HDMI COMMON DEFECTS

In order to develop an automatic algorithm that detect defects in the HDMI system it is necessary to identify the most common defects that can be present. These defects are based in the past experience with similar boards and in some research about each element that compose the HDMI video system.

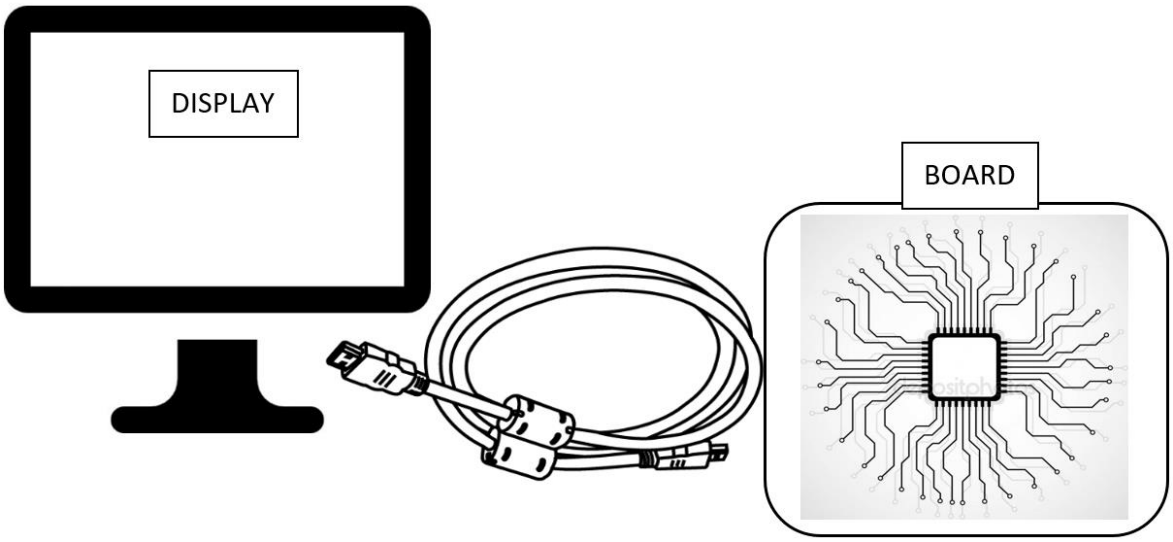


Figure 23: HDMI test elements

Assuming that the rest of the equipment are working correctly (the display and the cable) the defects would be present in three main elements inside the processor board:

- The HDMI connector. A defective connector or a bad soldering could generate crossing lines, short circuit and interferences
- The PCB. If the PCB has defects, it could generate short circuit or crossing lines.

- The processor. Generally, in modern processor the HDMI line is direct. It is not necessary to use other chips like video codec or converters. If the processor is not correctly soldered or presents any factory defect the HDMI signal will be corrupted.

After defining the possible elements that can create problem in the HDMI test, it will be described the possible outputs that the pattern detection algorithm has to detect. This failure will be represented starting from the reference test image pattern that is composed by three colored bars with the three main colors: red, green, blue.

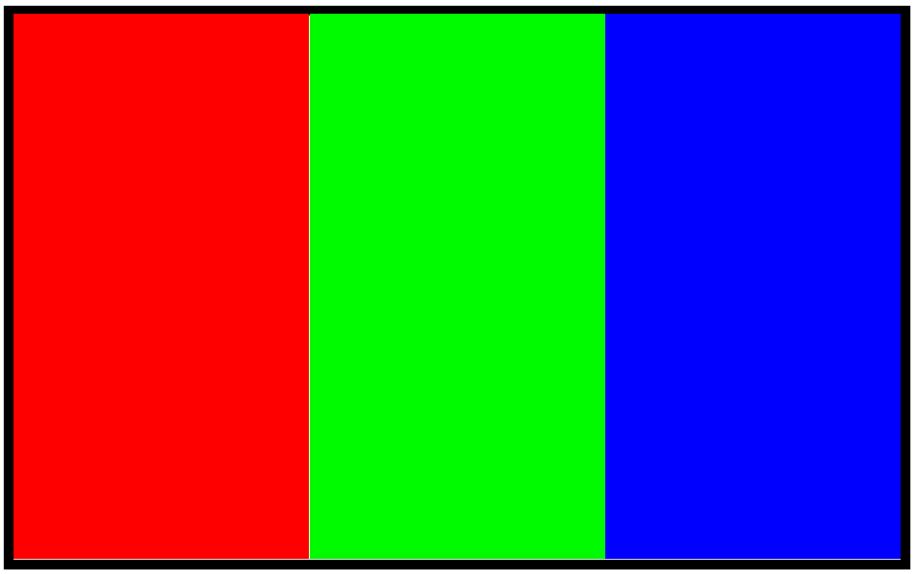


Figure 24: Test image pattern

1. The first failure that have to be detected is that the image is **complete black or white**. That will mean that the system is not working at all or is not able to display the image.

Figure 25: Complete black/white example

2. Another problem that have to be detected is that the system **could not display concrete color**. It will mean that some of the HDMI lines are not working correctly.

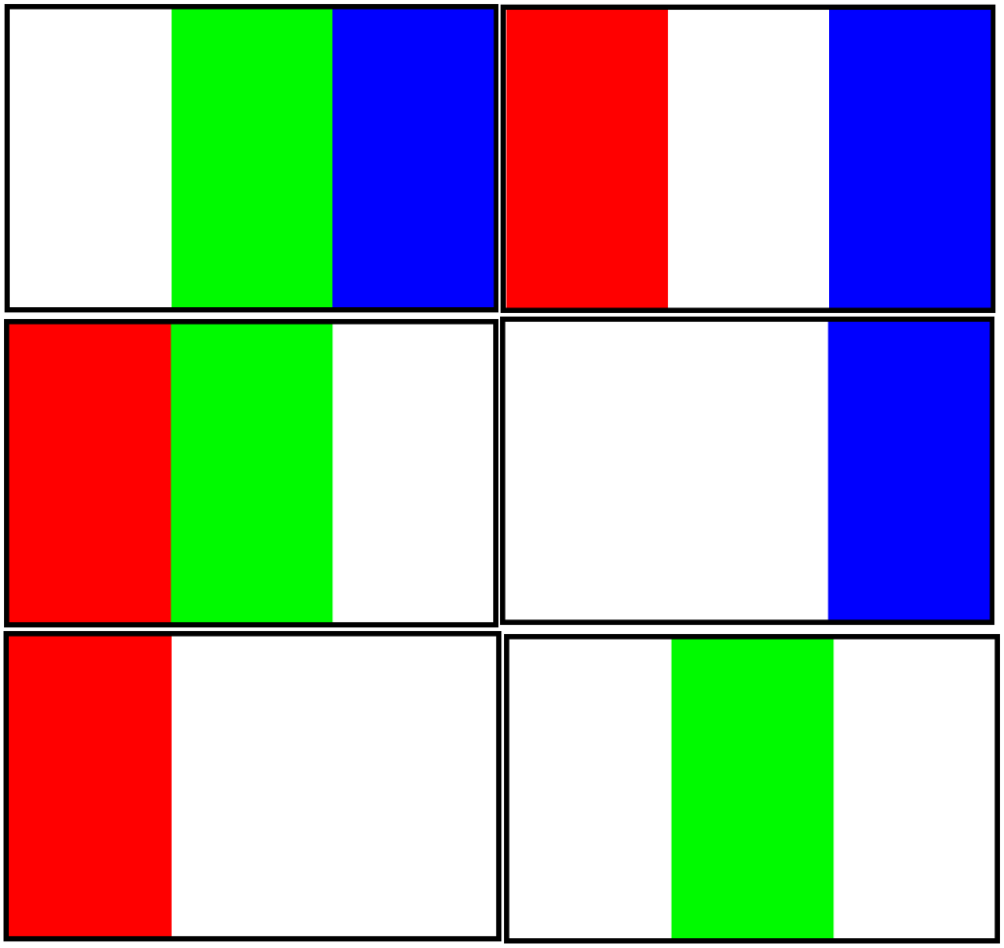


Figure 26: Color not displayed example

3. The next failure is that the system **changes the position** of the color. This will mean that there are crossing lines somewhere in the system.

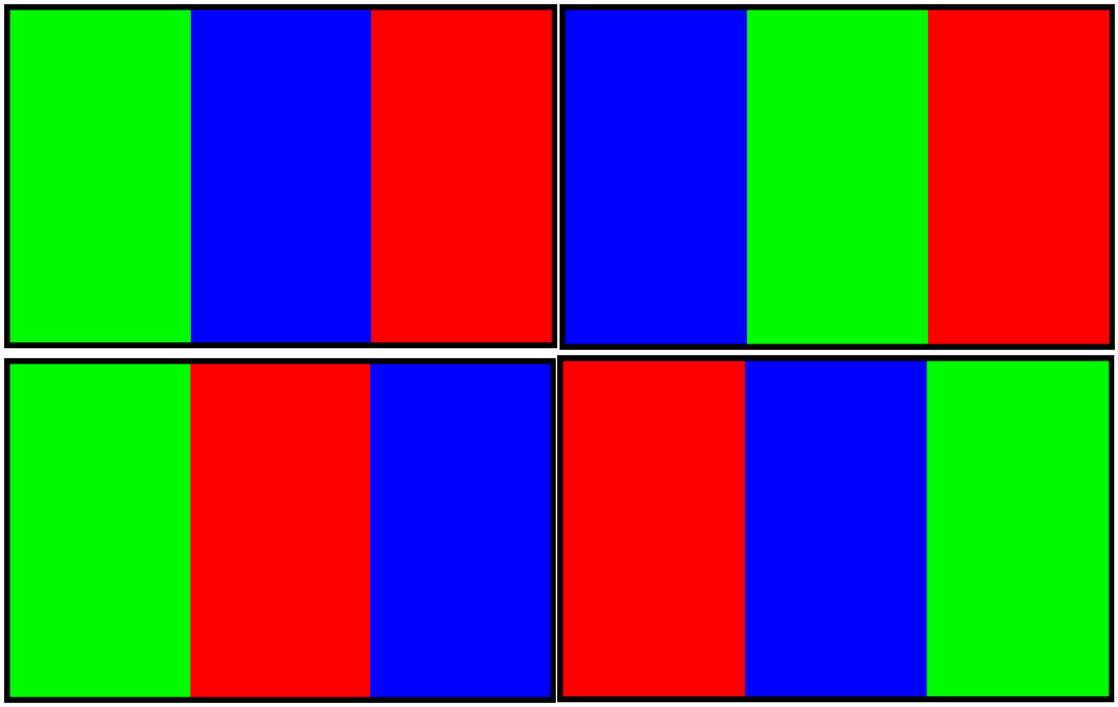
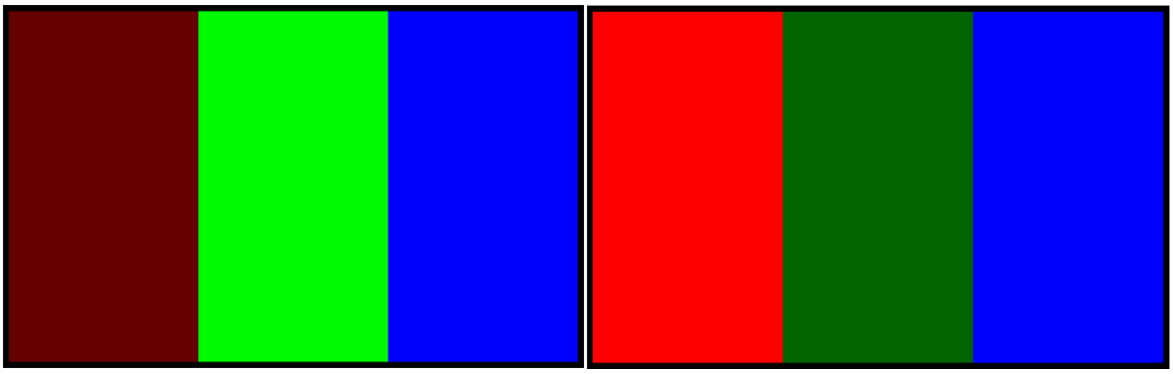


Figure 27: Color position change example

4. The last problem is that the intensity of some of all the colors are really low. This is generally due to a bad soldering in the low bit color pad of the connector.



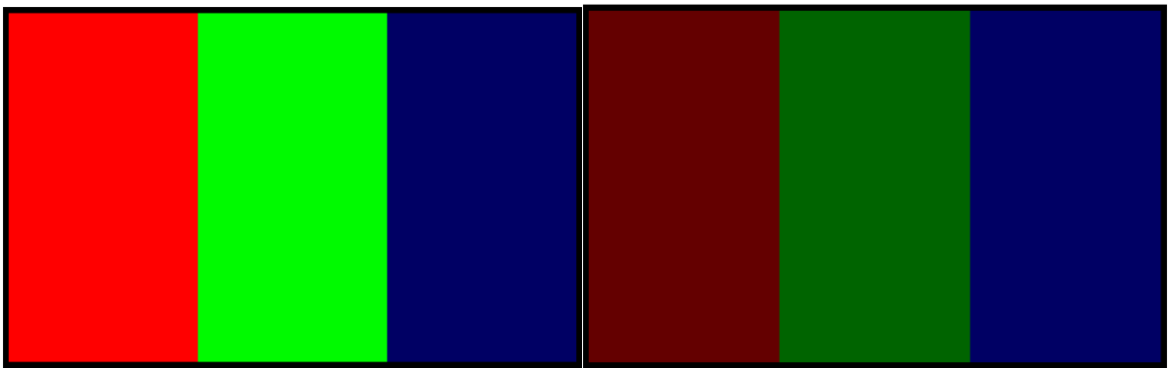


Figure 28: Color gradient failure

PATTERN DETECTION ALGORITHM

The pattern detection algorithm has the mission of detect all the previous failures and give a concrete result (True or False) and a corresponding description of the possible failure, if exist. The algorithm will have some main functions and the general procedure will be the following:

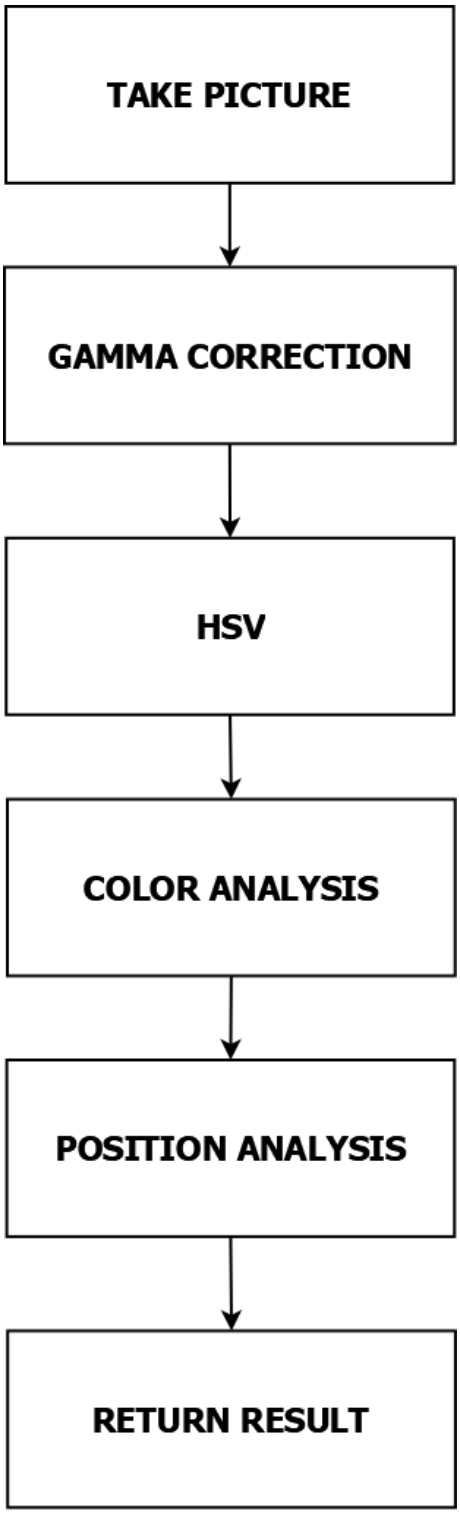


Figure 29: Pattern detect algorithm

In order to understand better the algorithm, it will be given a brief description of each part of the algorithm:

Take picture.

Using the class *VideoCapture* it can be taken a picture from the desired camera device. Class for video capturing from video files, image sequences or cameras. The class provides C++ API for capturing video from cameras or for reading video files and image sequences.

To capture a video or picture, it is needed to create a VideoCapture object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera. Normally one camera will be connected (as in my case). You can select the second camera by passing 1 and so on. After that, it can be captured frame-by-frame.

Gama correction.

In order to avoid problem with the illumination that can alter the test results, it have to correct the image using a Gamm correction. It will be taken as reference the green color in the middle of the image and select the gamma factor that make this green color better.

Gamma correction is also known as the Power Law Transform. First, the image pixel intensities must be scaled from the range [0, 255] to [0, 1.0]. From there, it been obtained the output gamma corrected image by applying the following equation *(OpenCV Gamma Correction, 2015):*

$$O = I^{\frac{1}{G}}$$

Where **I** is the input image and **G** is the desired gamma value. The output image **O** is then scaled back to the range [0, 255].

Gamma values lower than 1 will shift the image towards the darker end of the spectrum while gamma values > 1 will make the image appear lighter. A gamma value of **G=1** will have no affect on the input image:

Figure 30: Gamma correction example (www.pyimagesearch.com/)

By using OpenCV and Python it can be implemented a function to perform this gamma correction:

```
import numpy as np
import argparse
import cv2

def adjust_gamma(image, gamma=1.0):
    # build a lookup table mapping the pixel values [0, 255] to
    # their adjusted gamma values
    invGamma = 1.0 / gamma
    table = np.array([((i / 255.0) ** invGamma) * 255
        for i in np.arange(0, 256)]).astype("uint8")

    # apply gamma correction using the lookup table
    return cv2.LUT(image, table)
```

Figure 31: Gamma correction function

HSV.

On a computer, color can be represented in many formats. However, in this tutorial, we will be strictly concerned with only RGB and HSV.

With RGB, a pixel is represented by 3 parameters, blue, green, and red. Each parameter usually has a value from 0 – 255. For example, a pure blue pixel on your computer screen would have a B value of 255, a G value of 0, and a R value of 0.

With HSV, a pixel is also represented by 3 parameters, but it is instead Hue, Saturation and Value.

Unlike BGR, HSV does not use the primary color to represent a pixel. Instead, it uses hue, which is the color or shade of the pixel.

The saturation is the intensity of the color. A saturation of 0 is white, and a saturation of 255 is maximum intensity. Another way to think about it is to imagine saturation as the colorfulness of a certain pixel. Value is the simplest of the three, as it is just how bright or dark the color is.

HSL (hue, saturation, lightness) and HSV (hue, saturation, value) are two alternative representations of the RGB color model, designed to more closely align with the way human vision perceives color-making attributes. In these models, colors of each hue are arranged in a radial slice, around a central axis of neutral colors which ranges from black at the bottom to white at the top. The HSV representation models the way paints of different colors mix together, with the saturation dimension resembling various shades of brightly colored paint, and the value dimension resembling the mixture of those paints with varying amounts of black or white paint. HSV will be give better results in order to classify the colors.

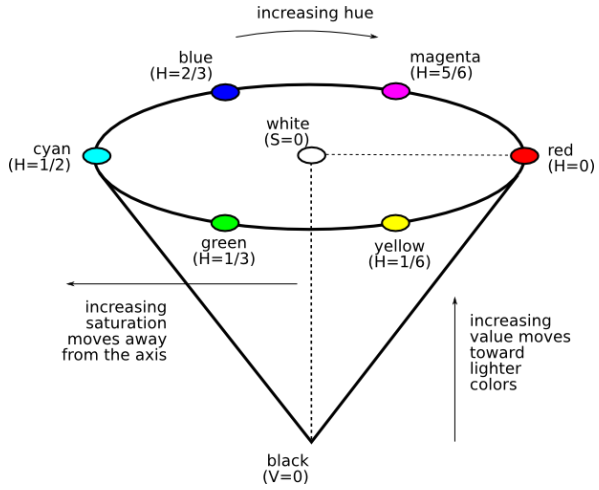


Figure 32: HSV diagram (www.infohost.nmt.edu)

HSV model is characterized by:

- The hue (H) of a color refers to which pure color it resembles. All tints, tones and shades of red have the same hue.

- Hues are described by a number that specifies the position of the corresponding pure color on the color wheel, as a fraction between 0 and 1. Value 0 refers to red; 1/6 is yellow; 1/3 is green; and so forth around the color wheel.

- The saturation (S) of a color describes how white the color is. A pure red is fully saturated, with a saturation of 1; tints of red have saturations less than 1; and white has a saturation of 0.

- The value (V) of a color, also called its lightness, describes how dark the color is. A value of 0 is black, with increasing lightness moving away from black.

In order to change to HSV color space using OpenCV, it can be use the function:

```
cv.CvtColor(image, outputHSV, cv.CV_BGR2HSV)
```

Color analysis.

In order to detect the colors in the captured image and can check if the image pattern has been correctly displayed, it is necessary to detect the red, green and blue colors in the image.

In order to do that it is necessary to define the bound for the three colors and apply a mask to the main image. Doing this, it will be obtained a result black and white image where the white part will be the desired color of the applied mask. For example, if the red mask is applied, the image result would be:

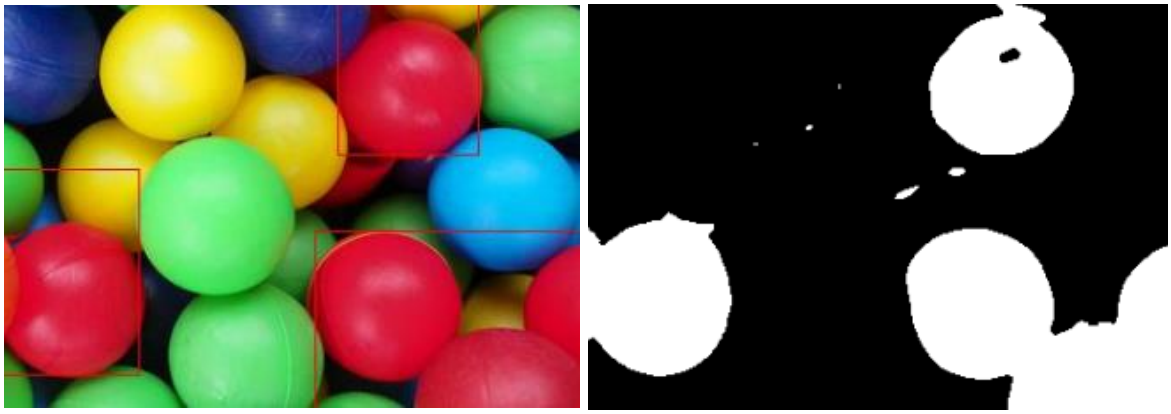


Figure 33: Red color detection result

Position detection.

It is necessary to detect that all the colors are in the right position. This is quite simple. It is only necessary to select little squares in the middle of each of the three bars and check that the color of this square are the expected.

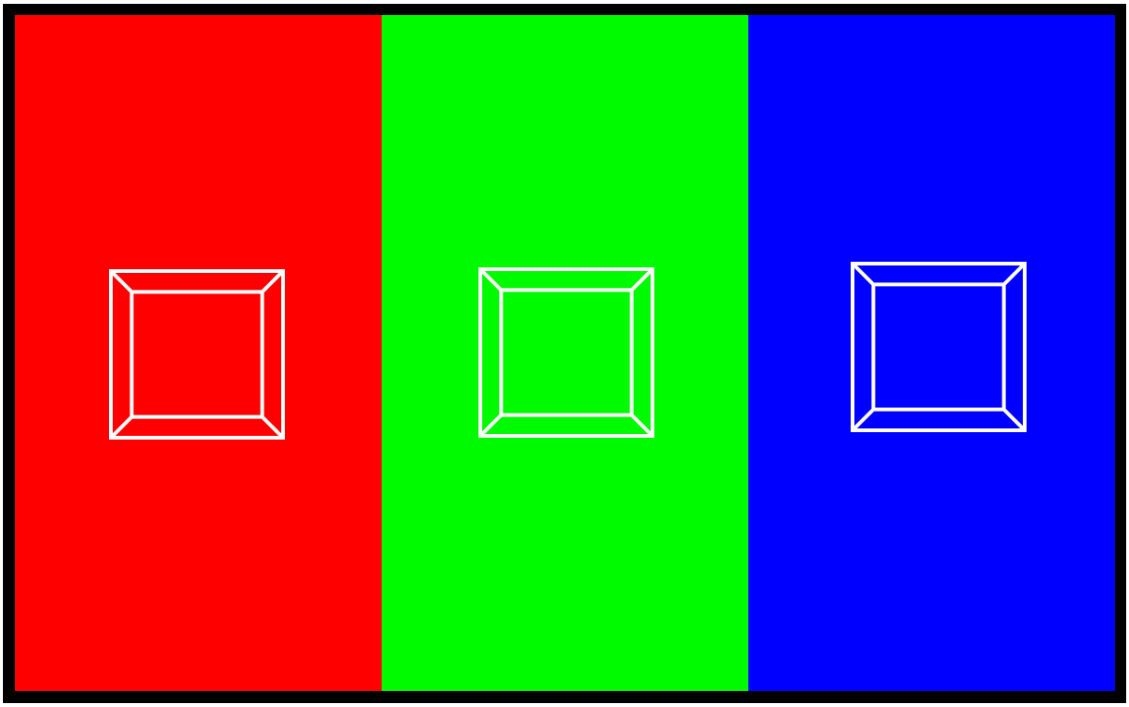


Figure 34: Position detection squares

Return results.

If the colors are correct and in the correct position it will be returned a **True**. In the order case it will be returned a **False** and the reason of the fail.

6 Environmental impact

This chapter deal with the commitment that the developed project faces with the community. It is intended to present the impact that this work proposes in a social environment.

The Environmental Impact Assessment of projects is an instrument enabling the preservation of natural resources and the defense of the environment by introducing the environmental variable into decision-making on projects which are predicted to have a significant impact on the environment.

In this regard, there is a valid and updated legislation on a national level. This Law describes the strategic environmental assessment of plans and programs the evaluation of **projects' environmental impact** in a single regulation. Thereby establishing a similar scheme for both procedures and unifying its terminology.

No significant impact has been done to the environment other that the one derived from the power consumption of the building, the computer and the embedded systems used to develop this project. In the following table it is briefly described the power consumption.

Table 3: Environmental Impact

EQUIPMENT	POWER [kW]	TIME [h]	UNITS	ENERGY [kWh]
Computer	0.350	900	1	315
Embedded system (full)	0.05	900	2	45
Display	0.1	900	1	90
Building	20	900	1	18000
			TOTAL	**18450**

7 Budget

The work developed in this thesis is not a product to be sold and have no direct economic benefits. This thesis had the aim to investigate new strategies to test embedded board systems. It has been developed thanks to the contribution of ISEE 2007 S.L. This company has facilitated the installations and needed material and in return they can use this thesis and all its contents to implement and upgrade their current test at factory.

7.1 Cost of Material

The only cost of the material derived from the project may refer to the material used, the embedded systems and the computer.

Table 4: Material Cost

DESCRIPTION	**UNITARY COST (taxes included)**	**UNITS**	**COST [€]**
Computer	2000	1	2000
Display	100	1	100
Embedded Systems	500	2	1000
		TOTAL	**3100**

7.2 Cost of Human Resources

The cost of personal is only related with student and the director of the thesis.

Table 5: HHRR cost

DESCRIPTION	UNITARY PER HOUR	HOURS	COST [€]
Director	40	30	1200
Student	10	900	9000
		TOTAL	**10200**

7.3 Cost of Energy

The cost of the energy is related with the power consumption of the elements that form part of the project and the cost of the energy off the building used to develop this project.

The current average cost of the energy is around 0.13 € per kWh.

Table 6: Energy Cost

DESCRIPTION	ENERGY [kWh]	€ / kWh	ENERGY [kWh]
Computer	315	0.13	40.95
Building	18000		304.2
Embedded Systems	45		5.85
Display	90		11.7
		TOTAL	**362.7**

7.4 Total Cost

The total cost will be the result of the partial cost sum.

Table 7: Total Cost

DESCRIPTION	COST
Cost of Material	3100
Cost of HHRR	10200
Cost of Energy	362.7
TOTAL COST	**13662,7**

8 Planning & Scheduling

This project is an extended project of 900 hours. It has been lasted almost 6 months, working approximately 160 hours per month. In this chapter it is explained the plan and schedule followed in this project.

8.1 Project research

The project research and gather information lasted around one month. In this period, it has been defined the project parts, selecting the methods, the programming languages and the operating system. It has been done a research of the different elements and gadgets necessary for the test. Furthermore, it has been done some research looking for the best option for the amperemeter solution.

8.2 Amperemeter design

The amperemeter design has been another important part of the project. After the research and selected the INA233 as the amperemeter core. After this it have been needed another month in order to design the schematic of the complete amperemeter system, the software that read the current consumption and send it through serial port. Some test with different boards and shunt values to select the best option and adjust the calibration values.

8.3 HDMI test building

The HDMI pattern detection computer vision system has been important in this project. It has taken another month. Half of this month has been the software building using OpenCV and Python, deciding the necessary function and structure. The other half month, have been used to implement this software in an ARM platform with Linux and optimizing the code. In addiction it has be done some experimental parts in order to adjust the color bounds and parameters.

8.4 Operating system building

The building of the operating system has taken half month. It has been started from a default configuration files, provides by the chip manufacture, and from this it has been configured and compile the rest of the necessary characteristics. An important part of this has been to solve the different compilation errors that derivate from a bad configuration. This have taken some time because some research was necessary for each error.

8.5 Software development

The develop of the complete test software has taken a month approximately. This time has been spent on implementing the different functions from Linux into python and on building the database.

8.6 Testing and modifications

The last month has been spent on testing and adjusting the test system. It has been prepared a test zone and several situations for the test to fail. From this results some modifications have been applied in order to improve the system results and its performance.

9 Conclusions

This project has been a complete great challenge. It has mix a lot of disciplines related with the Master's degree in Automatic Control and Robotics.

The main objectives raised for this project has been completely achieved. Furthermore, there are some extra challenge that has been fulfilled:

Performance

This project was developed in order to improve the performance of an old implemented factory test. The resulting project has passed the old test. It performs more exhaustive test in a better time. Furthermore, it tests more elements that the old application

Robustness

This test system was looking for a robustness in terms of stability and avoid system software crashes. Thanks to the database and the python robustness it has been achieved a robust software that doesn't get stuck and always report to the database.

Modularity

The test system has been created for two Single-Board Computers models but the only thing that is necessary to do when it is wanted to implement this test in other modules is to change the corresponding data in the database. In this way, it has been created a system that can run on any processor board because the critical data are inside the database and Linux operating system does not change from one board to another.

It has been a great experience that has related the concepts with the real world. The three main pillars of this project are closely related with three disciplines of the Master's degree:

Computer vision

The computer vision discipline has been applied mostly in the HDMI pattern detection application. As it has been seen, several concepts used were similar to the concepts explained at class.

Sensors, Instrumentation and Communications

All the development around the amperemeter was closely related with the Sensors, Instrumentation and Communications discipline. It has been used a sensor based on

a shunt resistor, an analogic to digital converter and a i2c and serial communication. These three elements are similar to the explained during the master. The clear understood of concepts like the communication protocol of a i2c bus make easier the develop of an application that use this bus.

Embedded Systems

Almost all the project is related with the Embedded Systems discipline. From the low level programming of embedded systems to the compilation of a full Linux kernel.

9.1 Future work lines

In spite of this test system is completely functional, there are several points that would be improved in the future. Due to the extensity of the project and the time limitations it has been not implemented. Nevertheless, the project has been developed thinking in this improvements in order to make easy its implementation in the future. The possible improvements are the following:

9.1.1 Improve the connection at factory

It is necessary to create a structure to facilitate the connection of all the elements that involve the test, like the cables, USB pen drives, etc. By mean of creating a solid structure and a methodology attached to it, the test will be faster and the factory operator will be less exposed to connection failures.

9.1.2 Create an application for the factory operator

In order to report the results to the factory operator, it is used a HDMI screen. It has a problem, if the HDMI infrastructure is not correctly working the results will be not displayed. Furthermore, the operator has to check the same number of displays as the number of simultaneous test that are carrying on. If a web application is developed in order to give the test results, the factory operator could use a tablet in order to check and follow the test. It will allow and facilitate the simultaneous tests.

9.1.3 Improve the HDMI pattern detection application

The HDMI pattern detection application works correctly. It is robust and it is able to adapt itself to external factors like the illumination. But this application can be improved more and use machine learning in order to teach more failures to the application and decrease the false positive and false negatives. It can be taught by using several images of difficult failures like the color gradient or bad illumination situations.

10 Annexes

There are several code annexes attached to the main project package. In order to not make larger this document it has been taken out of this document. The documents attached are:

- Full Python project code

11 Bibliography

Adrian Rosebrock. (2015). *OpenCV Gamma Correction*. Retrieved from https://www.pyimagesearch.com/2015/10/05/opencv-gamma-correction/

Alsa Project. (2018). *Advanced Linux Sound Architecture Wiki*. Retrieved from https://www.alsa-project.org/main/index.php/Documentation

Federico Di Gregorio, D. V. (2016). *PostgreSQL database adapter for Python*. Retrieved from http://initd.org/psycopg/docs/

Global, T. P. (2018). *PostgreSQL Documentation*. Retrieved from https://www.postgresql.org/docs/

Instruments, T. (2013). *MTD Utilities* . Retrieved from http://processors.wiki.ti.com/index.php/MTD_Utilities

IOzone . (2016). *IOzone Filesystem Benchmark*. Retrieved from http://www.iozone.org/

IPERF. (2018). *Iperf Documentation*. Retrieved from https://iperf.fr/iperf-doc.php#3doc

ISEE. (2016). *IGEP V2 DATASHEET.*

ISEE. (2016). *Who is ISEE*. Retrieved from https://www.isee.biz/about/company/who-is-isee

Liecht, C. (2015). *pySerial's documentation*. Retrieved from https://pythonhosted.org/pyserial/

Python Software Foundation. (2018). *Subprocess management*. Retrieved from https://docs.python.org/2/library/subprocess.html

Python Software Foundation. (2018). *The ElementTree XML API*. Retrieved from https://docs.python.org/2/library/xml.etree.elementtree.html

Python Software Foundation. (2018). *Unit testing framework*. Retrieved from https://docs.python.org/2/library/unittest.html

STMicroelectronics. (2017). *ST NUCLEO-F042K6 Datasheet.*

Team, O. D. (2018). *OpenCV documentation*. Retrieved from https://docs.opencv.org/2.4/index.html

Texas Instruments. (2017). *INA233 Datasheet.*