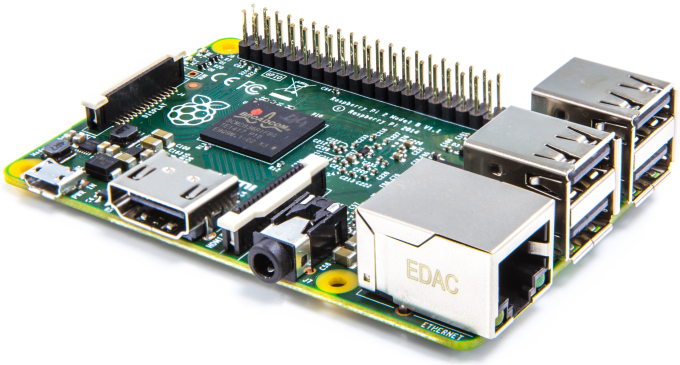
Simulink Defined Radio with Raspberry Pi2



Raspberry Pi

Telecommunication Experience

Ver. 1.1
May, 2016

Gianni Pasolini*, Alessandro Bazzi°, Mirko Mirabella+

WiLab - Wireless Communications Lab

* Dept. of Electrical, Electronic, and Information Engineering "Guglielmo Marconi"
University of Bologna, Italy

° Institute of Electronics, Computer and Telecommunication Engineering
National Research Council, Italy

+ CNIT - Italian Inter-University Consortium for Telecommunication

This work has been funded by MathWorks in the framework of the *Academic Support A#: 1-2073488447*.

Corresponding author
Gianni Pasolini, Univ. of Bologna (Italy).
Phone +39 051 2093553
Email: gianni.pasolini@unibo.it

Acknowledgments
This activity has been carried out at the Wireless Communications Lab (WiLAB) of the University of Bologna/National Research Council. The authors wish to thank the WiLAB Director, Prof. Oreste Andrisano, for his comments and suggestions and for providing all technical facilities.

The authors wish to thank Stefano Olivieri (MathWorks Italy) for his helpful suggestions and support.

Document versions
Version 1.0, October 2015: first release of this document.
Version 1.1, May 2016: Besides other minor modifications, the new Chapter 8 has been added, dealing with the QPSK modulation.
Note: the latest version of this document can be found, along with all the Simulink projects, at *www.simulinkdefinedradio.com*

Contents

Chapter 1

Raspberry Pi2 and Simulink configuration

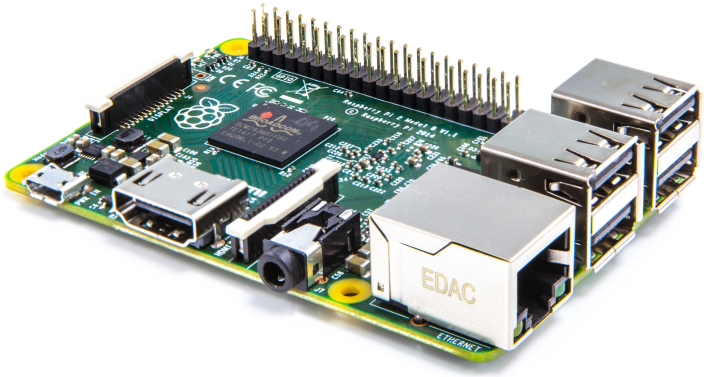


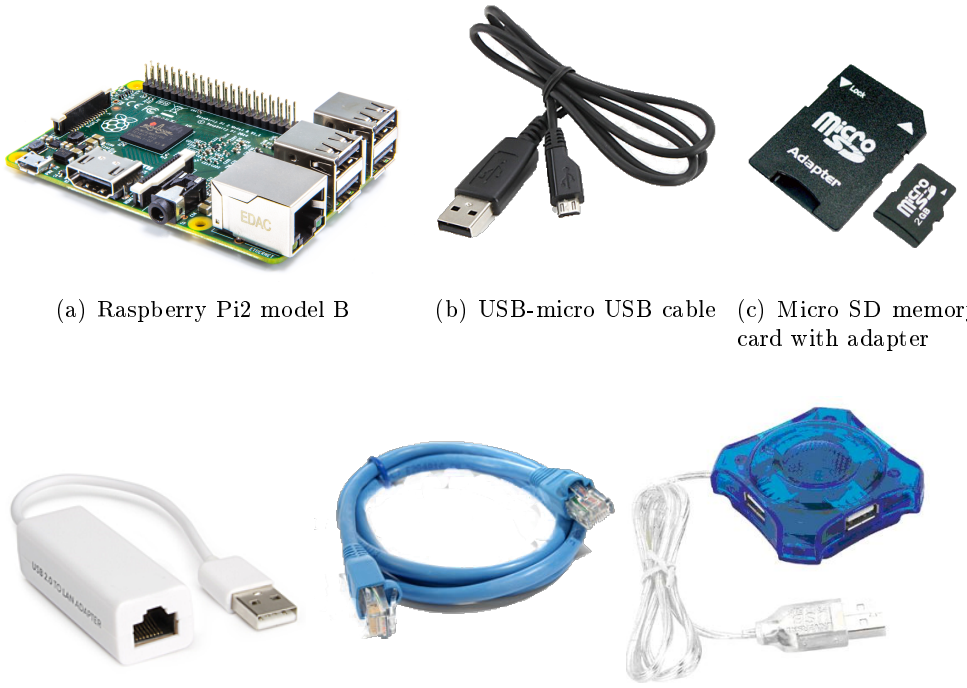
Figure 1.1: Raspberry Pi2 model B

In order to connect the small and cheap Raspberry Pi2 single-board computer (Fig:1.1) with MATLAB and Simulink, some preliminary operations must be performed. The configuration procedure described in this chapter allows the communication between the PC hosting MATLAB/Simulink and the Raspberry Pi2 board, without the need to add a keyboard, a monitor, and a mouse to the Raspberry Pi2, with evident benefits in terms of system setup.

Please note that, although the concepts described in this document are of general validity, small changes in the graphical representation or available options may be observed for versions of MATLAB that are different from the R2015a used in this document.

1.1 Hardware Support Packages

The growing availability of low cost prototyping boards, microcontrollers and single-board computers, suggested Mathworks to develop specific *Hardware Support Packages* that expand the functionalities of MATLAB and Simulink, allowing them to interface with a variety of devices developed by third-party vendors.



(a) Raspberry Pi2 model B

(b) USB-micro USB cable

(c) Micro SD memory card with adapter

(d) USB to LAN converter

(e) Ethernet cable

(f) USB Hub

Figure 1.2: Basic equipment

In this chapter we describe, in particular, the procedure to install the *Raspberry Pi Support Package*, that extends both MATLAB and Simulink libraries and updates the Raspberry Pi™operating system (OS).

The prerequisites and the equipment required to successfully complete the operation are detailed in the following section.

1.2 Equipment

The installation of the *Raspberry Pi Support Package* requires a PC equipped with MATLAB/Simulink as well as an SD memory-card slot. The following items are also necessary:

- Nr.1 Raspberry Pi2 model B (Fig.1.2(a));
- Nr.1 USB-micro USB cable for the power supply (Fig.1.2(b));
- Nr.1 micro SD memory-card, with an SD adapter if needed by the PC (Fig.1.2(c));
- Nr.1 USB to LAN converter, needed in some cases, as detailed in Section 1.5 (Fig.1.2(d));
- Nr.1 Ethernet cable (Fig.1.2(e));
- Nr.1 USB hub, needed in case the PC has only two USB ports (Fig.1.2(f)).

Moreover, a Mathworks account is needed (free registration at www.mathworks.com).

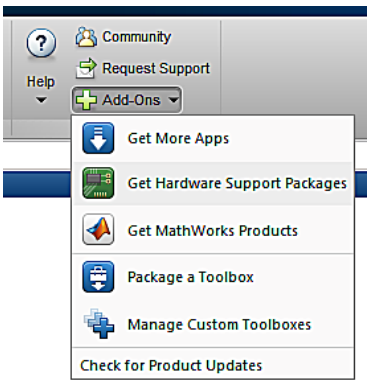


Figure 1.3: Get the Hardware Support Packages

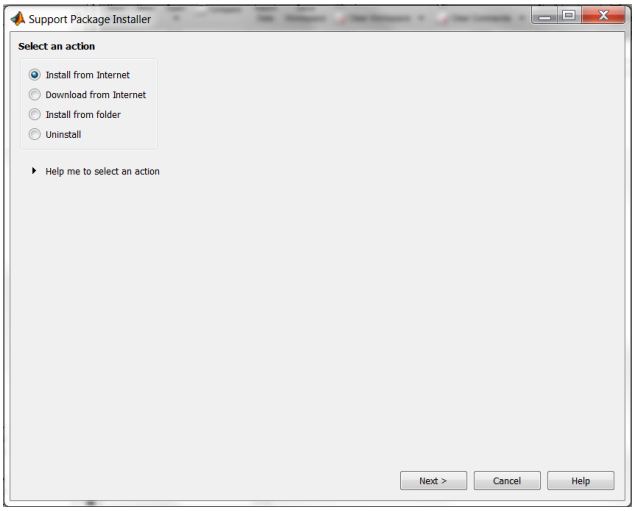


Figure 1.4: Select an action

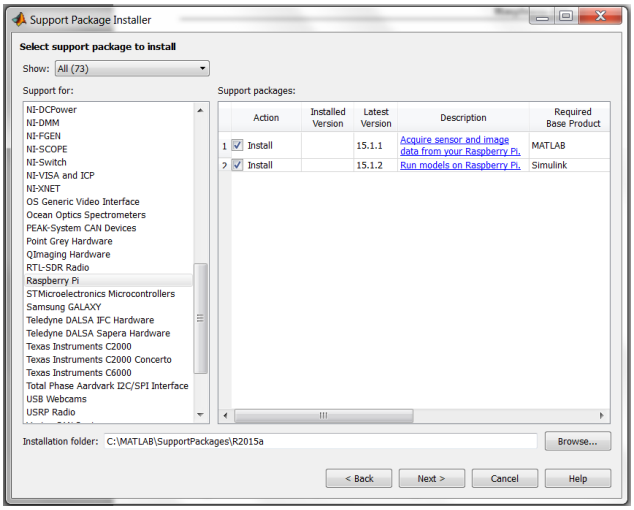


Figure 1.5: Select support package to install

1.3 Hardware Support Package Installation

The *Raspberry Pi Support Package* can be downloaded and installed directly in the MATLAB environment through **Add-Ons→Get Hardware Support Packages**, as shown in Fig. 1.3.

When requested (see Fig.1.4, select the option *Install from Internet* and continue, clicking *Next*.

Once you have selected the support package for Raspberry Pi™, as shown in Fig.1.5, click *Next*. Click then *OK* when the window shown in Fig.1.6 is displayed.

The installation procedure continues with the request to log-in with a registered Mathworks account, as shown in Fig.1.7. Click *Log In* and enter your log-in information in the window shown in Fig.1.8.

Accept the licence conditions shown in Fig.1.9 and click *Next*. Click *Next* once more when the window shown in Fig.1.10 is displayed.

Finally, click *Install* when the confirmation request shown in Fig.1.11 appears.

The procedure continues with the installation of the ***MATLAB Support Package for***

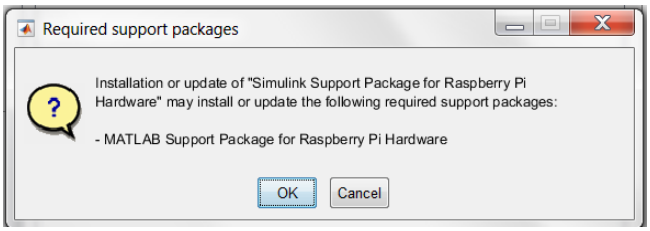


Figure 1.6: Required support package

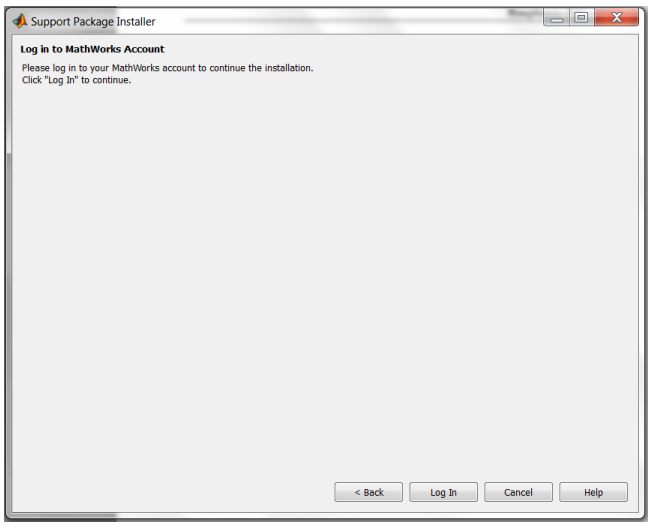


Figure 1.7: Log in to Mathworks Account

Raspberry Pi Hardware and the ***Simulink Support Package for Raspberry Pi Hardware***.

At the end of the installation, the window shown in Fig.1.12 is displayed. MATLAB and Simulink are now ready to interact with Raspberry Pi™devices.

The next step is the installation of the Raspberry Pi™OS on a micro SD card and the configuration of the connection between the PC and the Raspberry Pi™.

Before clicking *Continue* in the window shown in Fig.1.12, thus starting the OS installation, it is necessary to establish how to configure the network connection between the Raspberry Pi™and the PC. This aspect will be discussed in the following section.

1.4 Installation of the Raspberry Pi™OS and network configuration

Communications between MATLAB/Simulink and the Raspberry Pi2 board occur through a network connection, that must be properly set up.

The configuration procedure can be performed during the installation of the Raspberry Pi2 OS, that takes place after the above described *Hardware Support Package* installation, or later, by accessing directly to the network configuration files of the Raspberry Pi2.

The first option, described below, is the fastest one. It allows, in fact, to immediately control the Raspberry Pi2 board through the PC hosting MATLAB/Simulink once the OS is installed, with no need to access the Raspberry Pi2 configurations with additional peripherals, such as monitor, keyboard and mouse.

Coming back to the installation procedure, click *Continue* in the window shown in Fig.1.12 to start the OS installation.

When the window shown in Fig.1.13 appears, select *Raspberry Pi (Simulink)* and click *Next*.

The first thing to do is to choose the Raspberry Pi™ model. The procedure explained hereafter is valid for all the Raspberry Pi™ models that can be selected, although in this document we will

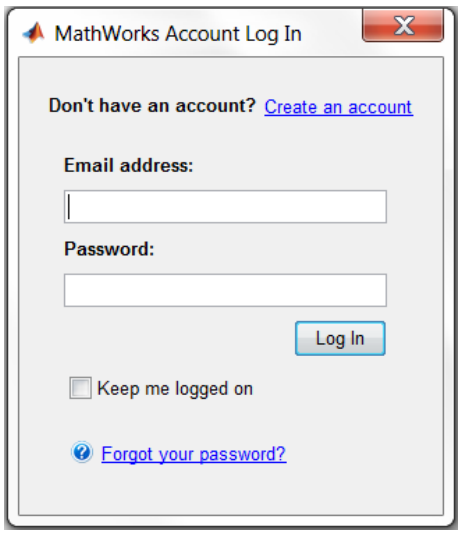


Figure 1.8: Mathworks Account Log In

proceed with the installation of the Raspberry Pi2 OS, as shown in Fig.1.14.

Clicking *Next* you will get to the window shown in Fig.1.15, regarding the configuration of the network connection between the PC and the Raspberry Pi™. According to what has been previously said, the choice will be *Manually enter network settings*, as shown in Fig.1.15. The procedure detailed in the following will directly modify the Raspberry Pi2 system files containing the network parameters. This ensures the immediate use of the hardware once the installation is concluded, without further configurations.

The window shown in Fig.1.15 also requires to enter the *Host name* that identifies the device being installed. As an example, we chose the *Host name* **Raspberrypi-TLC1**, but the choice is absolutely arbitrary[1].

In addition, an IP address must be manually selected. The choice of the IP address to be assigned to the Raspberry Pi2 depends on the availability of free addresses within the local-area-network (LAN): with the command ***PING #IPaddress***, executed in the Windows command interpreter *CMD*, it is possible to check the actual availability of an IP address. The procedure here described adopts the configuration shown in Table 1.1.

IP ADDRESS	**169.254.0.3**
NETWORK MASK	**255.255.0.0**
DEFAULT GATEWAY	**169.254.0.1**

Table 1.1: Choice of Raspberry Pi2's network parameters

The final objective is to create a local network between the PC and the Raspberry Pi2. If a network switch is available, more than one Raspberry Pi2 can be simultaneously controlled once each device has been properly set up in terms of IP address. In the example given in the table

[1]The *Host name* assigned in this phase will be displayed in the *prompt* of the Raspberry Pi2 Linux Shell, that will be described later. By default, MATLAB and Simulink will assign the device a predefined *Host name*, corresponding to its IP address.

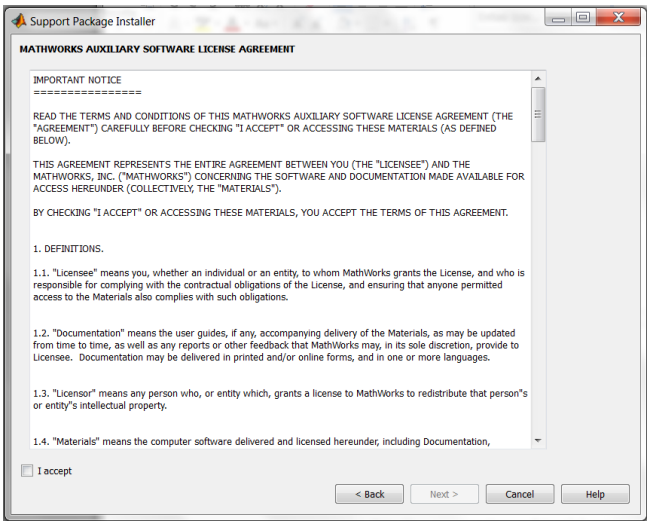


Figure 1.9: License agreement

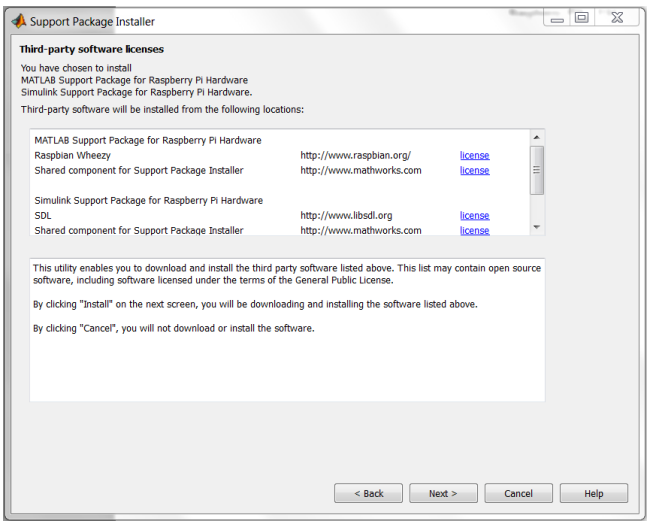


Figure 1.10: Third-party software licenses

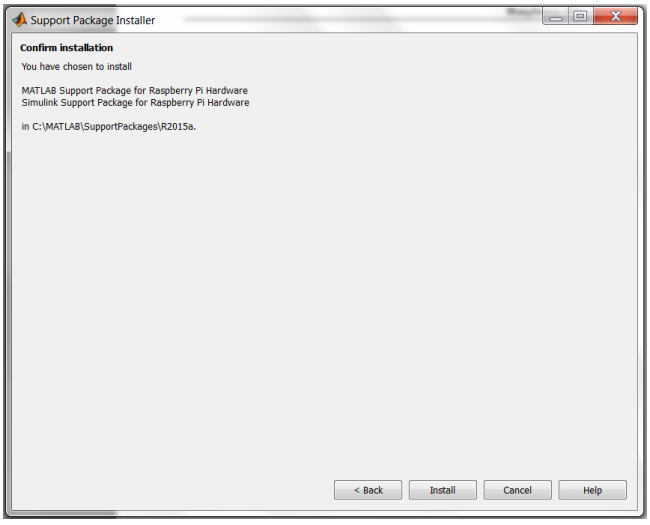


Figure 1.11: Confirm installation

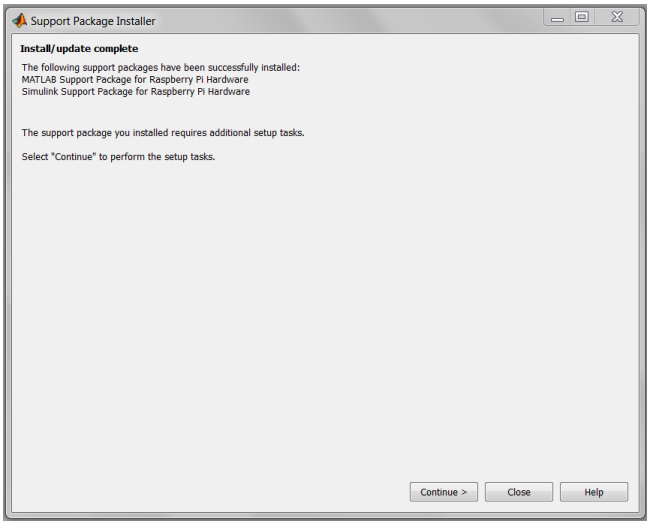


Figure 1.12: Install/update complete

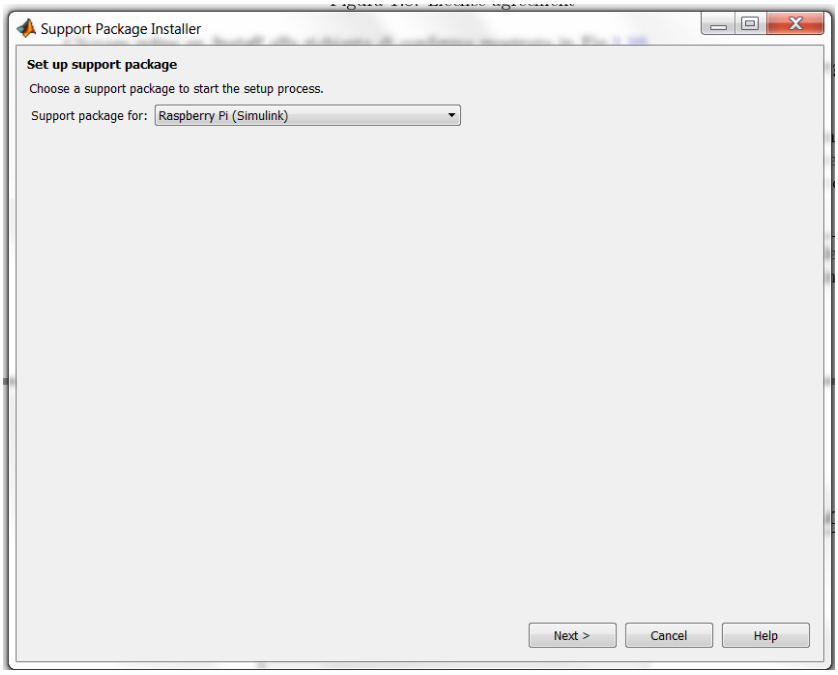


Figure 1.13: Set up support package

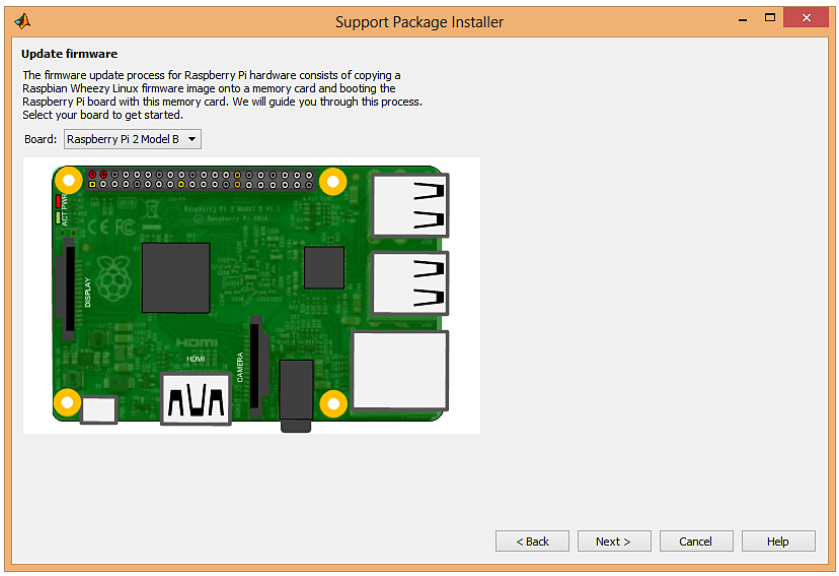


Figure 1.14: Choice of the Raspberry Pi2 model

1.2, the addresses assigned to three Raspberry Pi2 have been reported. As can be noticed, the network mask and the default gateway are the same.

Clicking *Next* in the window shown in Fig.1.15, the window depicted in Fig.1.16 appears, which requires to insert the micro SD card (with the SD adapter, if necessary) in the appropriate slot of the PC. Clicking *Next*, the window shown in Fig.1.17 is displayed, requesting a confirmation to write in the memory card. Clicking *write* the Raspberry Pi2's OS installation finally

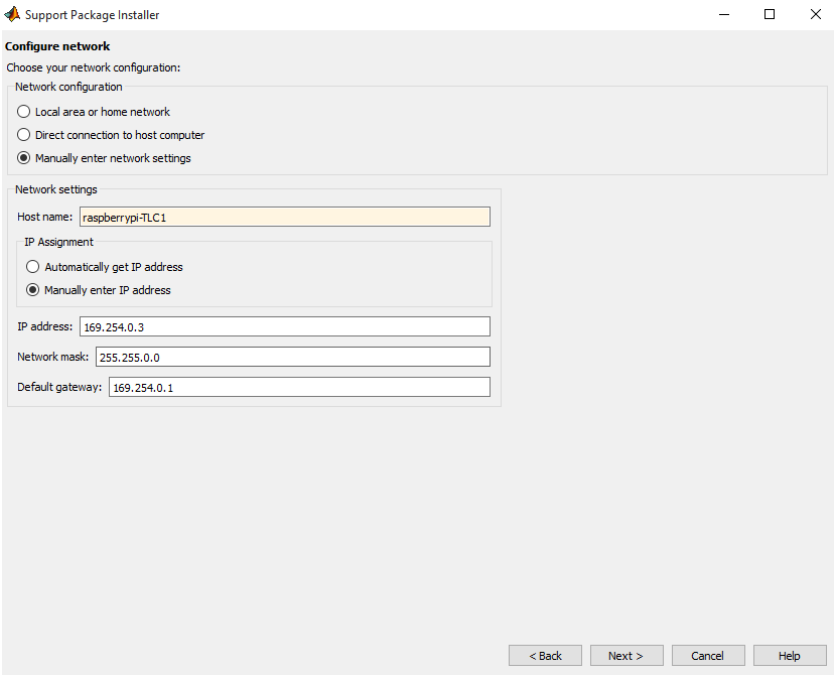


Figure 1.15: Network Configuration

Device	IP ADDRESS	NETWORK MASK	DEFAULT GATEWAY
Raspberry - TLC2	**169.254.0.4**	255.255.0.0	169.254.0.1
Raspberry - TLC3	**169.254.0.5**	255.255.0.0	169.254.0.1
Raspberry - TLC4	**169.254.0.6**	255.255.0.0	169.254.0.1

Table 1.2: Choice of network parameters for several Raspberry Pi2

starts, with the adopted network settings, on the micro SD card (Fig.1.18). This step might require some tens of minutes.

Once the operation is completed, the wizard suggests the steps to follow in order to connect the Raspberry Pi2 to the PC (Fig.1.19). Before clicking *Next*, and proceeding with the next step of the wizard, it is necessary can be necessary to install and set up an USB to Ethernet converter, as explained in Section 1.5. The use of an USB to Ethernet converter is required any time a Ethernet port is not available on the PC.

1.5 Installation and configuration of the USB to LAN converter

Communications between the PC and the Raspberry Pi2 take place through a network (Ethernet) connection. This entails that a direct connection between them occupies the Ethernet port of the

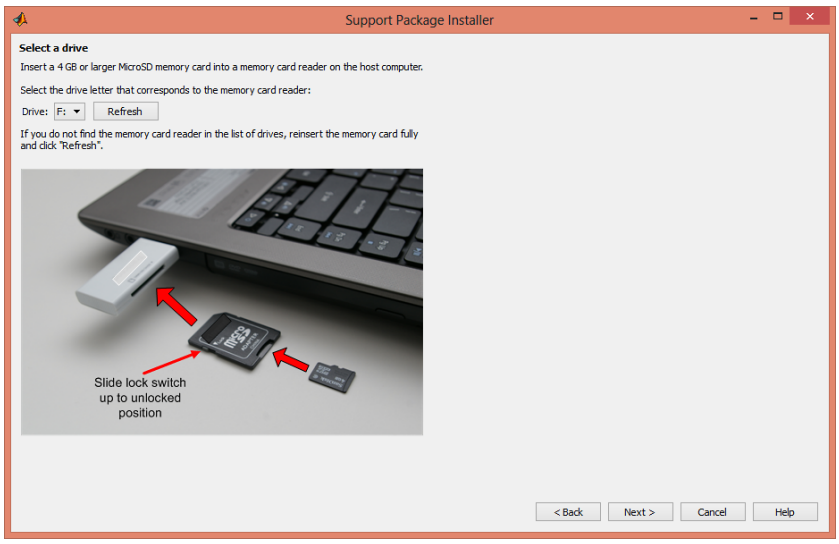


Figure 1.16: OS Installation

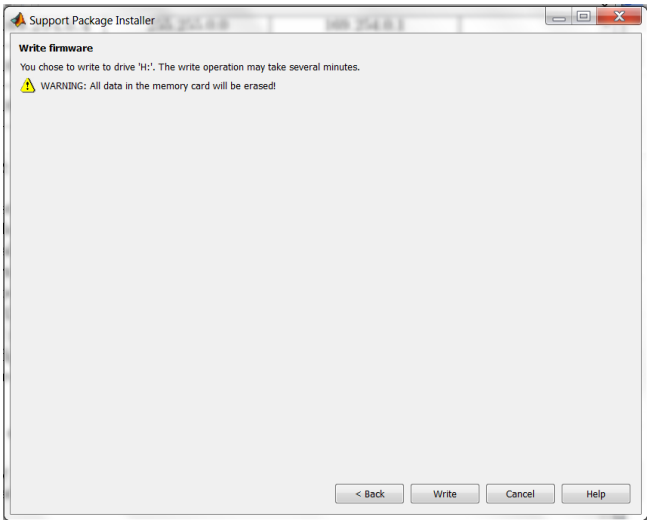


Figure 1.17: Write firmware

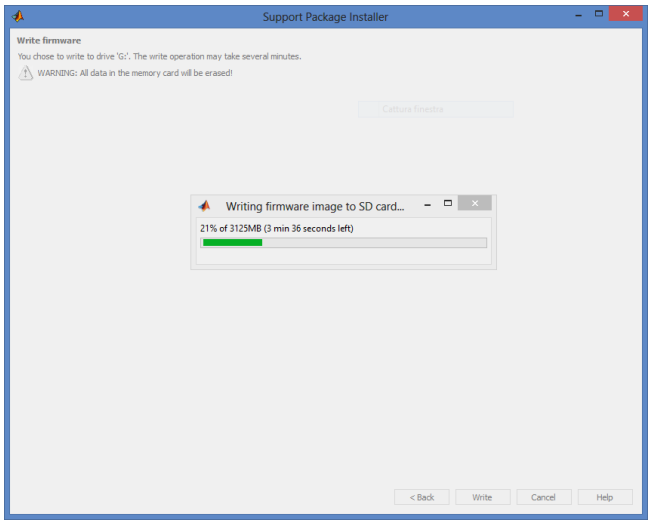


Figure 1.18: Write firmware

PC, which could be prevented from connecting to the local network as well as to the Internet. In order to connect the Raspberry Pi2 directly to the PC avoiding this problem, an USB 2.0 to LAN converter can be used, like the one shown in Fig.1.20.

By using the converter, it is possible to create a local area network between the Raspberry Pi2 and the PC, with no need to occupy its Ethernet port.

Once the driver of the device has been installed (a step that is not always needed and depends from the specific device and PC OS), it is necessary to manually enter the IP address assigned to the converter. The choice of the address must be consistent with the subnet mask and gateway previously assigned to the Raspberry Pi2. To set the IP address, it is necessary to

- Connect the converter to an USB port of the PC;

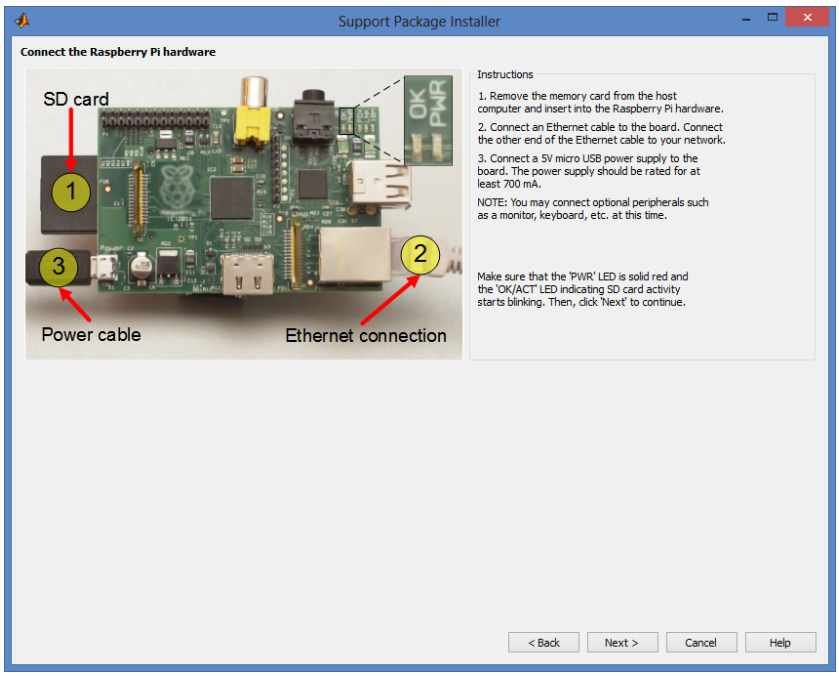


Figure 1.19: Connect the Raspberry Pi™hardware

Figure 1.20: USB to LAN converter

- Access the network configuration window through ***Control Panel*> *Network and Internet*>*Network Connections***;
- Identify the network card to be configured (corresponding to the USB to LAN converter);
- Modify the **TCP/IPv4** properties, as shown in Fig.1.21.

Fig.1.21 also shows example addresses for the configuration of the device.

1.6 Raspberry Pi2 Power On

Once the USB 2.0 to LAN converter has been configured, it is possible to test the proper functioning of the system. After inserting the micro SD card with the OS in the Raspberry Pi2 slot and after connecting the PC according to the scheme shown in Fig.1.22, the Raspberry Pi2 can be powered on.

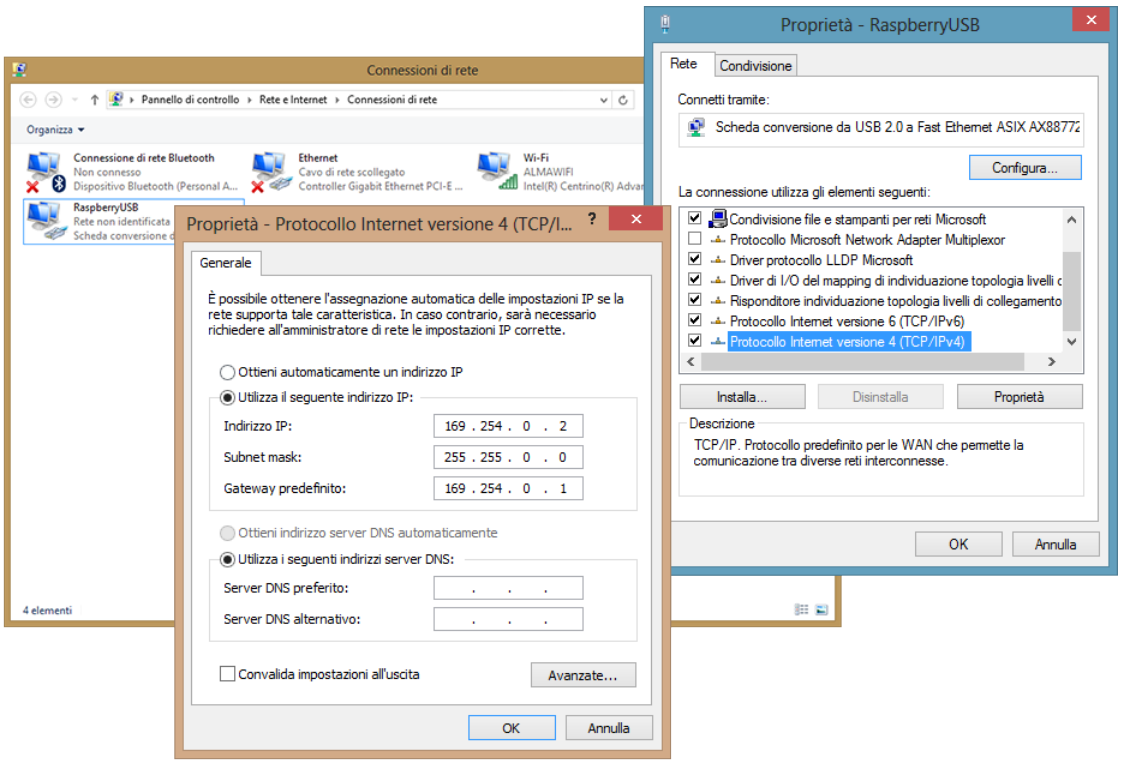


Figure 1.21: Configuration of the USB2.0 to LAN converter

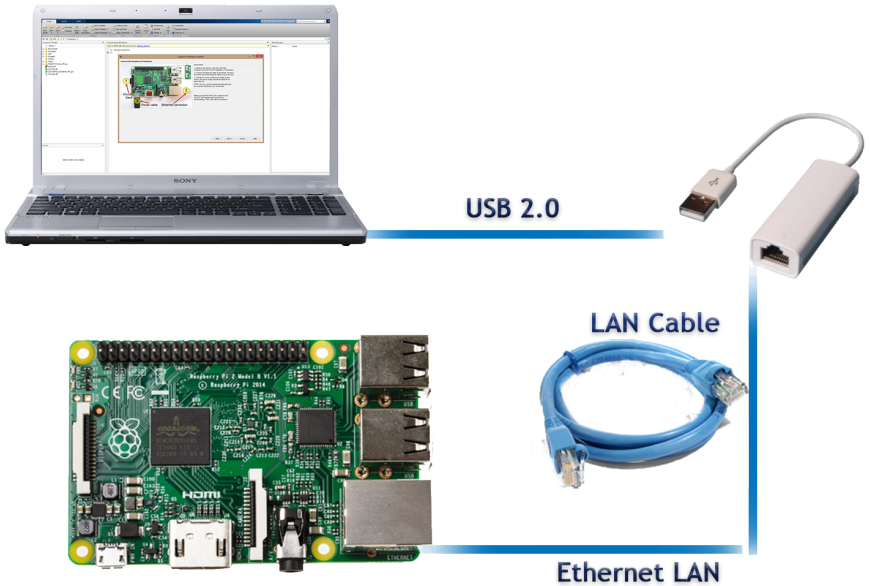


Figure 1.22: Connection scheme

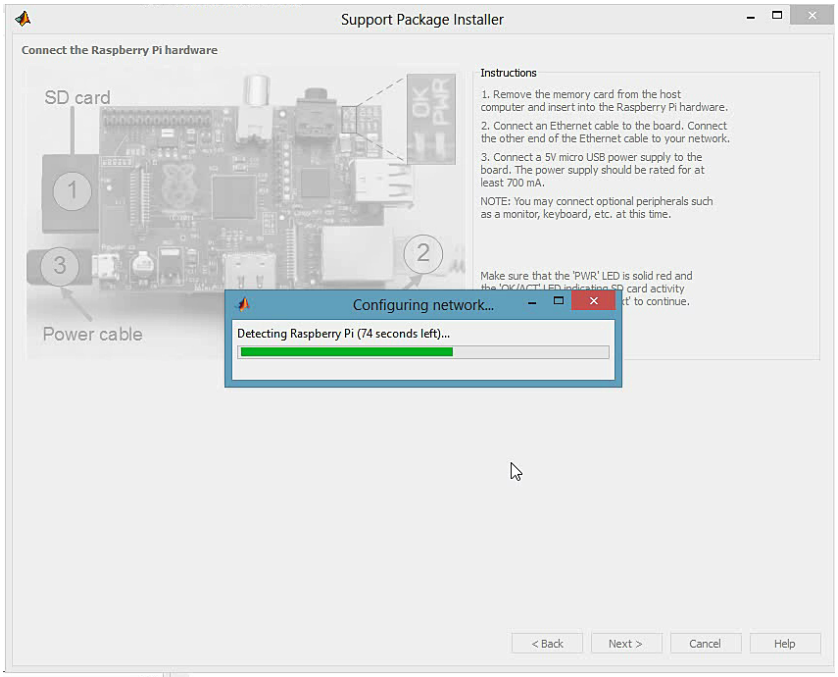


Figure 1.23: Network parameters test

The power supply can be done either using an AC/DC converter with micro USB connector, like those used for cell phones, or through a cable with USB-microUSB connectors, thus taking the power directly from one of the USB ports of the PC.

Please be aware, however, that the first option can cause some problems due to the disturbances generated by the AC/DC converter. The occurrence and entity of this phenomenon depends on the quality of the AC/DC converter. In any case, for this reason the second option might be preferred.

Once the Raspberry Pi2 is initialised, it is possible to test the connection between the device and the PC. For this purpose it is necessary to continue the installation procedure previously interrupted, by clicking *Next* in the window shown in Fig.1.19. The installation will continue with the hardware research phase (Fig.1.23), leading to the window shown in Fig.1.24. By starting the connection test (click *Test connection*) it is possible to have a confirmation of the successful connection (Fig.1.25).

Clicking *Next*, the installation procedure is completed (Fig.1.26). From now on, it is possible to interact with the Raspberry Pi2 directly within MATLAB, as explained below.

1.7 Controlling the Raspberry Pi2 through MATLAB

In order to test within MATLAB the connection between the PC and the Raspberry Pi2 and to get information about host name, user name, password and active build directory [2], the command below can be executed in the MATLAB command window:

[2] The build directory is the Raspberry Pi2 folder where all the files generated by MATLAB/Simulink are stored.

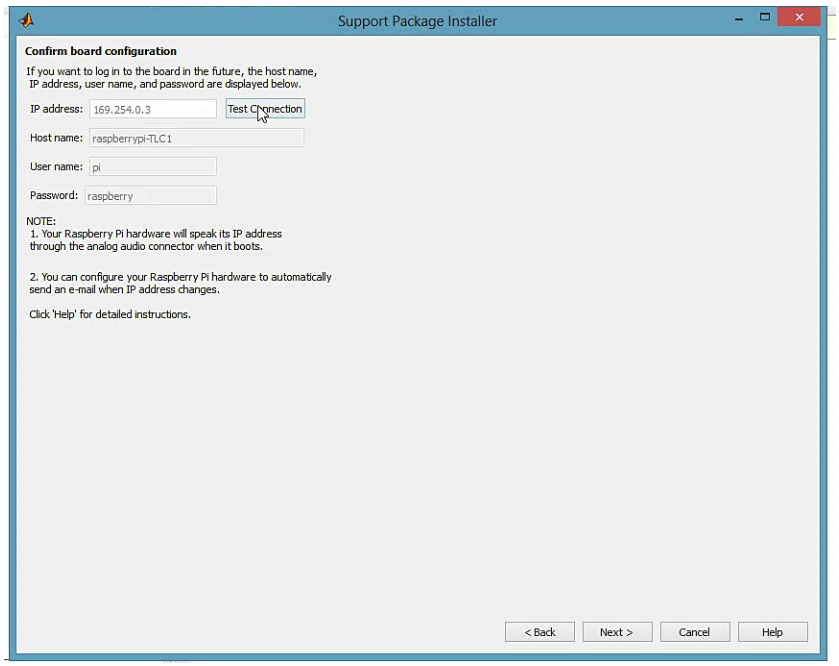


Figure 1.24: Confirm board configurations

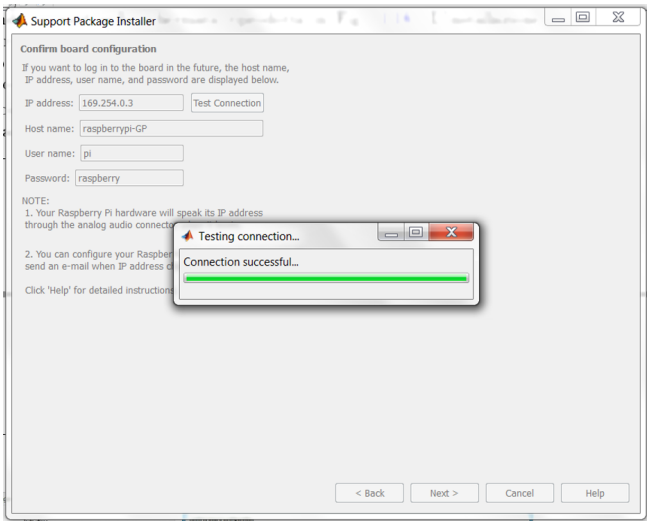


Figure 1.25: Connection successful

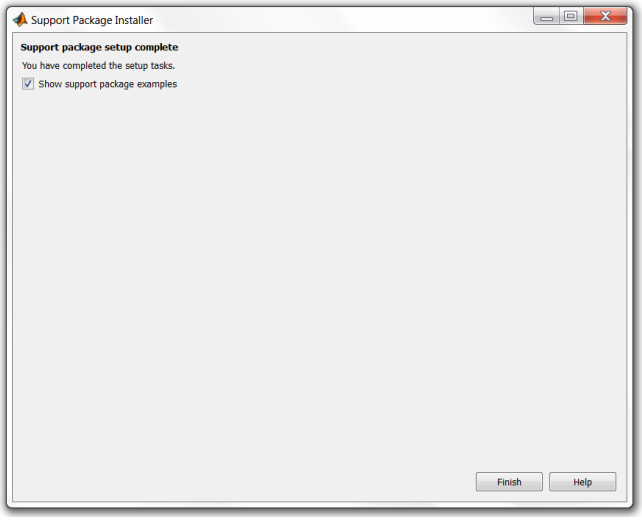


Figure 1.26: Support package setup complete

```
raspberrypi
```

Listing 1.1: Get info from the Raspberry Pi™board

The result is shown in Fig.1.27. The default settings are:

- HostName: Raspberry Pi2's IP address;

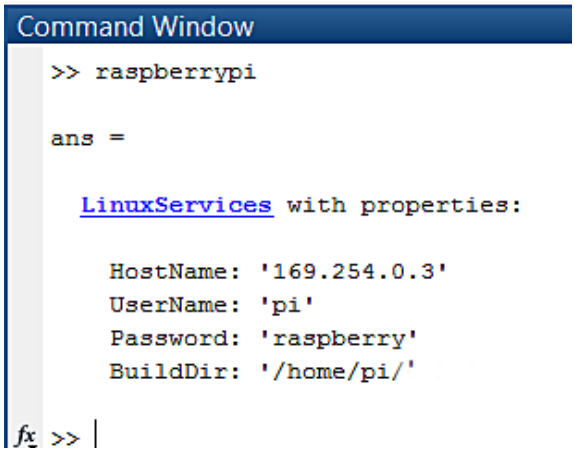


Figure 1.27: Information about connected device

- UserName: ***pi***;
- Password: ***raspberry***;
- Build directory: ***/home/pi/***.

Remaining in the MATLAB command window, it is possible to access the Raspberry Pi2 settings, directories, and files, by means of a Linux Shell. This is done through the following commands:

```
h=raspberrypi('169.254.0.3');
h.openShell('ssh')
```

Listing 1.2: MATLAB commands to open a Linux Shell

The function *raspberrypi*, invoked with the HostName (corresponding to the IP address) as the only parameter, provides a *handle* "h" associated to the addressed device, that is used to open the corresponding Shell with the command *h.openShell*.

As a result of the operation, the Shell opens requesting the UserName and the Password, as shown in Fig.1.28. Entering *UserName* ***pi*** and *Password* ***raspberry*** you access the Raspberry Pi2 Linux environment in the *Build Directory* ***/home/pi***, as shown[3] in Fig.1.29. To exit the Linux Shell enter the command *exit*.

If you want to access more than one Raspberry Pi2 connected to MATLAB, it is necessary to create different *handles*, as shown in Listing:1.3. The Linux Shell can be invoked using the corresponding *handle*.

```
h1=raspberrypi('169.254.0.4');
h2=raspberrypi('169.254.0.5');
```

Listing 1.3: MATLAB commands to open more than one Linux Shell

[3]Note that the Linux prompt displays the *Host name* assigned to the device during the OS installation (Section 1.4). As previously said, this *Host name* is different from the one used by default by MATLAB, that corresponds to the IP address assigned to the device.

Figure 1.28: Linux Shell

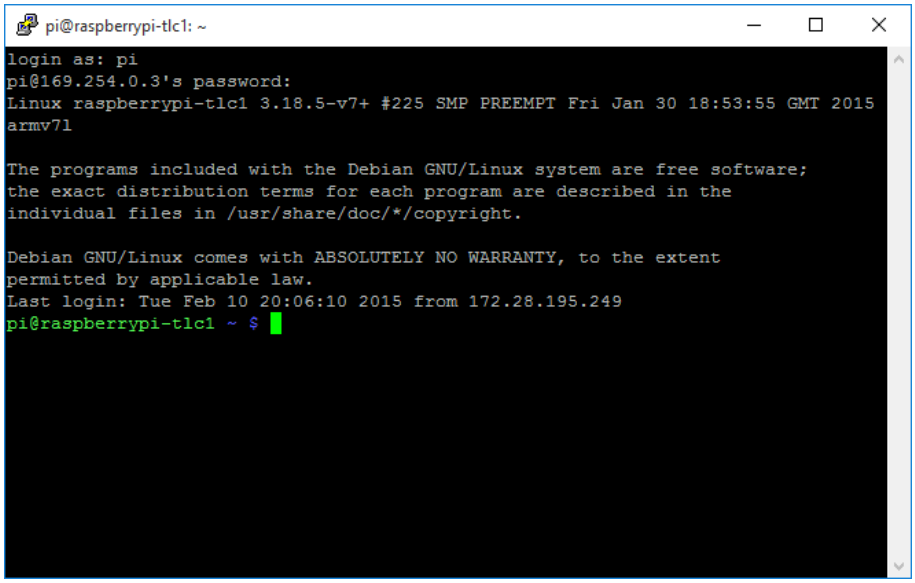


Figure 1.29: Linux environment

The whole installation procedure of a new Raspberry Pi2 can be performed within the MATLAB command window by simply launching the command *targetupdater*:

```
targetupdater
```

Listing 1.4: MATLAB command for new configuration

The Raspberry Pi2's network settings can be checked or modified using, within the Linux Shell, the command:

```
0   sudo nano /etc/network/interfaces
```

Listing 1.5: Linux code to access the network settings

as shown in Fig.1.30, where *sudo* is the Linux command to obtain the rights of the superuser, and *nano* is a text editor (obviously, any other text editor could be used as well). The Raspberry Pi2

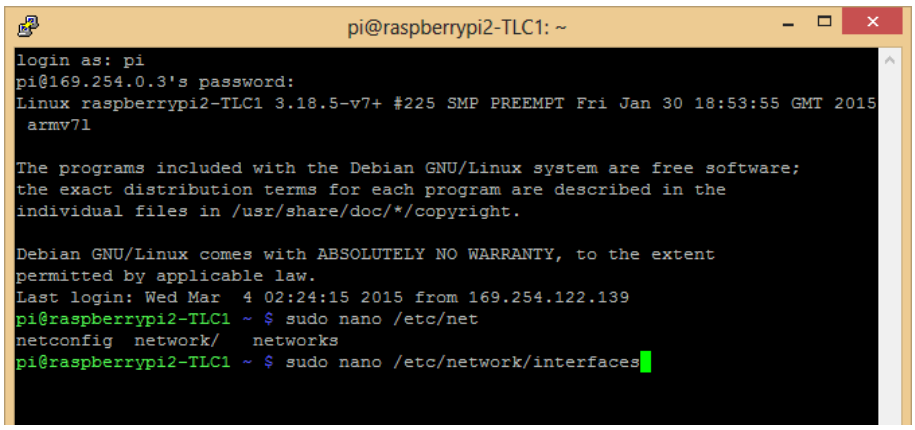


Figure 1.30: Linux Shell. Network configuration

answers opening, in the text editor *nano*, the file containing the network configurations (Listing 1.6).

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 169.254.0.3
netmask 255.255.0.0
gateway 169.254.0.1

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Listing 1.6: Network configuration file

Through the editor it is always possible to modify the Raspberry Pi2's network parameters.

Chapter 2

Sound card configuration

Figure 2.1: 33051D USB sound card

In order to use the Raspberry Pi2 board as a digital signal processing device, an analog input and an analog output are required. The Raspberry Pi2 is equipped with an analog output (headphone output) but not with an analog input. To fill this gap, it is possible to use an external USB sound card similar to the one shown in Fig.2.1, that supplies a microphone input and one more headphones output. In particular, the sound card that we used is equipped with the 33051D chipset.

This cheap device, connected to the Raspberry Pi2's USB port, will thus play a double role: analog-to-digital converter (ADC) for input signals and digital-to-analog converter (DAC) for output signals.

Being the device conceived for audio signals, its sampling frequency is limited to 48000 samples per second, with a resolution of 16 bits per sample. It follows that, in principle, the band of generated and received signals must be within the interval [0, 24 kHz]. In practice, however, the highest frequency that can be reached is in the order of 20 kHz; when this threshold is exceeded, in fact, a significant attenuation, that increases with the frequency, is introduced on the signal.

Please observe that, although the Raspberry Pi2 is equipped with an integrated audio output (headphones output), it is surely preferable to use the analog output provided by the external sound card. The integrated DAC is, in fact, of poor quality, as can be easily understood observing Fig.2.2: The yellow curve represents a 10 kHz sine generated by a Raspberry Pi2 and measured at the integrated output, whereas the red curve represents the same signal measured at the sound card output. The difference is evident, especially when the frequency increases (Fig.2.3).

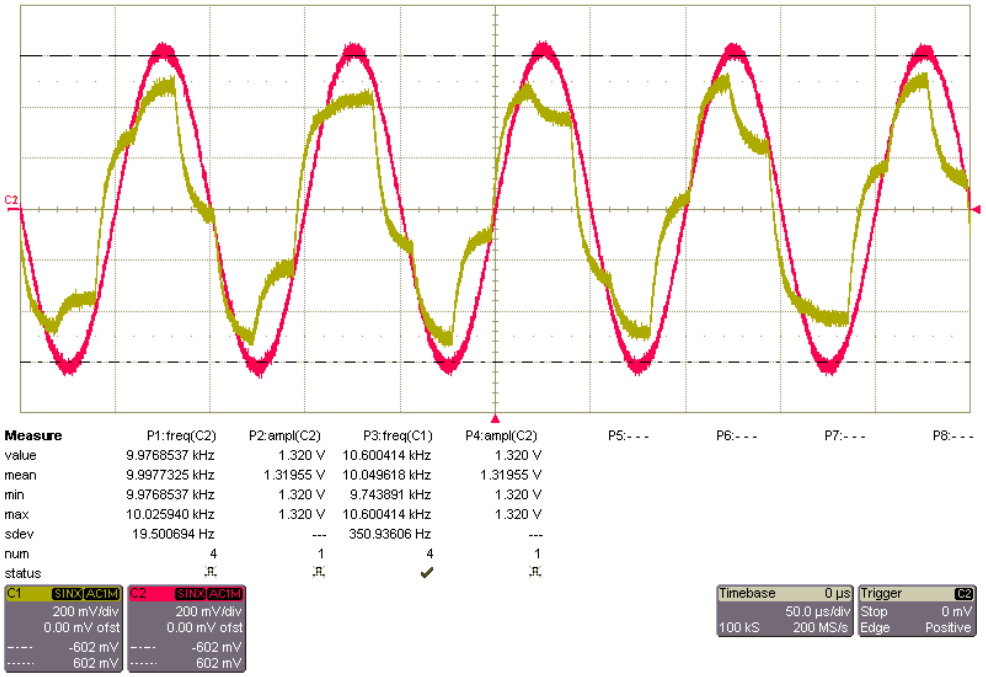


Figure 2.2: Comparison between the external DAC output and the Raspberry Pi2 headphones output. 10 kHz sine wave

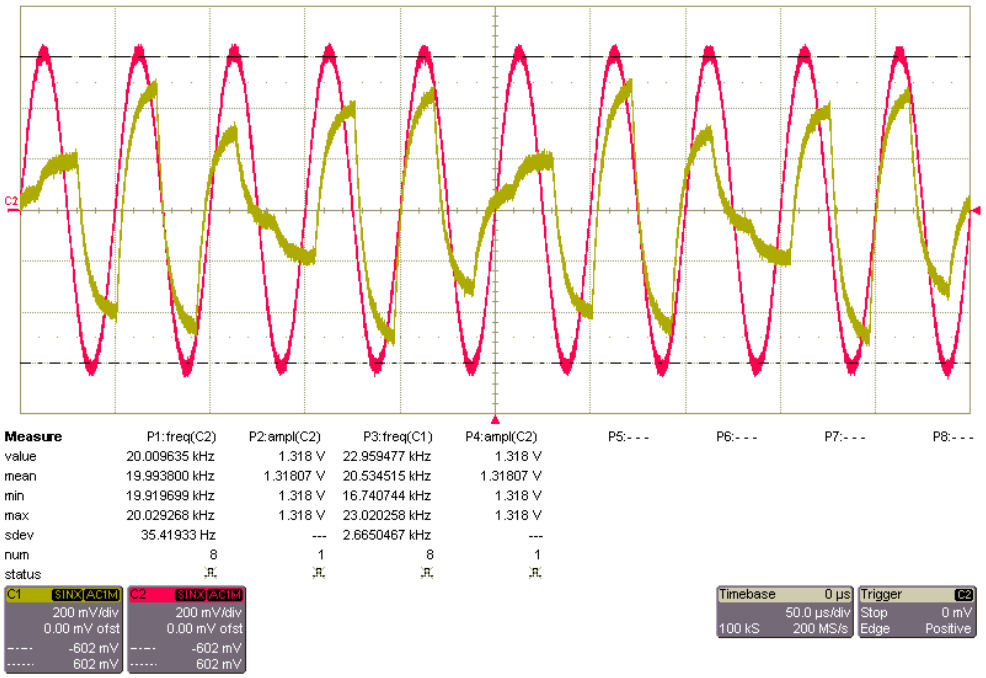


Figure 2.3: Comparison between the external DAC output and the Raspberry Pi2 headphones output. 20 kHz sine wave

2.1 External sound card configuration

The external sound card configuration requires to modify some parameters of the Raspberry Pi2's sound drivers.

The first step is to set this card as the primary audio device. After connecting the sound card to the Raspberry Pi2's USB port, it is possible to check its presence as output device by opening the Linux Shell (Section 1.7) and launching the command:

```
0 aplay -l
```

Listing 2.1: Linux command: ***aplay -l***

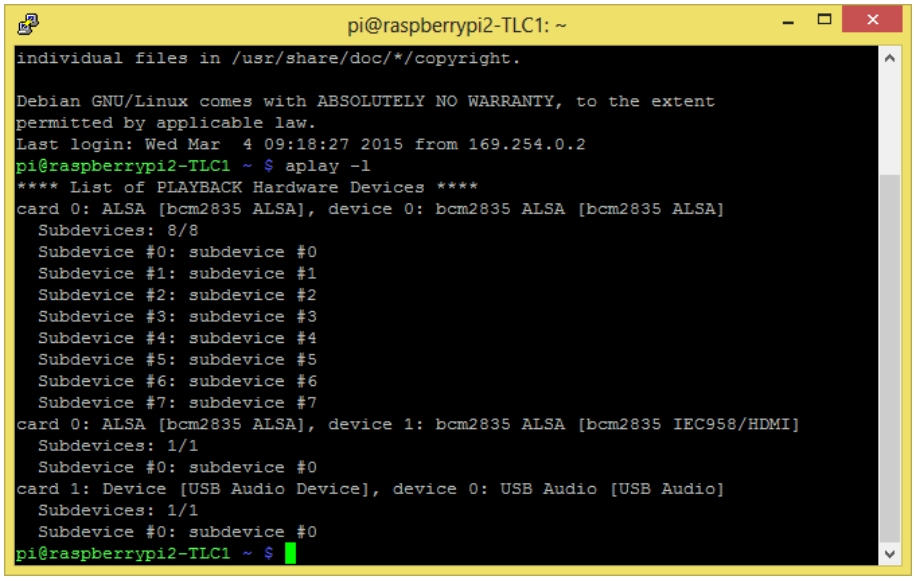


Figure 2.4: Linux *aplay -l* command

As can be seen in Fig.2.4, showing the result of this operation, the primary audio device is the *bcm2835 ALSA*, that is, the internal DAC. This can be easily deduced by its *Card* parameter, which is equal to 0.

The external DAC, labelled as *USB Audio Device*, has instead the *Card* parameter set at 1.

In order to reverse the priority of the two audio devices, it is necessary to modify the *alsa-base.conf* file. For this purpose, open the file *alsa-base.conf* in the text editor *nano* of the Linux Shell through the command

```
0 sudo nano /etc/modprobe.d/alsa-base.conf
```

Listing 2.2: Audio driver configuration

and add the new line (Listing:2.3),

```
#Set to 0 to obtain the loading of the USB Sound Card as first
options snd-usb-audio index=0
```

Listing 2.3: Audio driver modification

as shown in Fig.2.5. Find, then, the line reported in Listing:2.4, that must be commented adding the character "#" (Fig.2.5).

```
# Keep snd-usb-audio from beeing loaded as first soundcard
  options snd-usb-audio index=-2
```

Listing 2.4: Audio driver modification

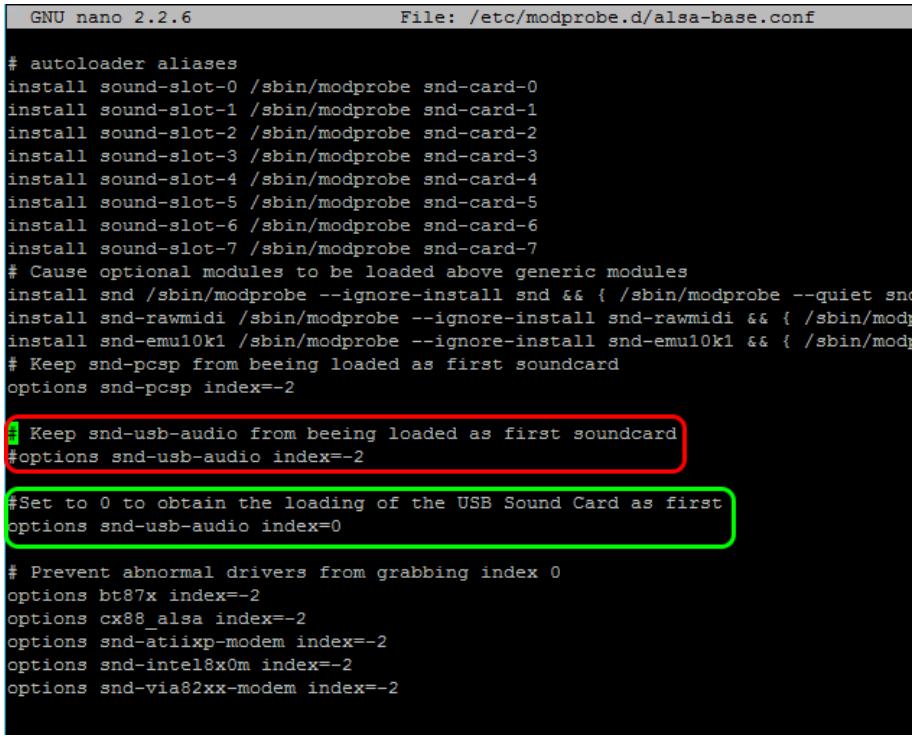


Figure 2.5: File Alsa-Base.conf

For the change to be effective, it is necessary to restart the system through the reboot command (Listing:2.5).

```
sudo reboot
```

Listing 2.5: Linux command: ***reboot***

Once the system is restarted, you can verify that the external sound card is now the primary audio player (Fig.2.6) by checking it through the command in Listing:2.1. Moreover, executing the command in Listing:2.6 you will see that an audio acquisition (CAPTURE) device is also present (Fig.2.6): It is the microphone input, made available by the USB sound card.

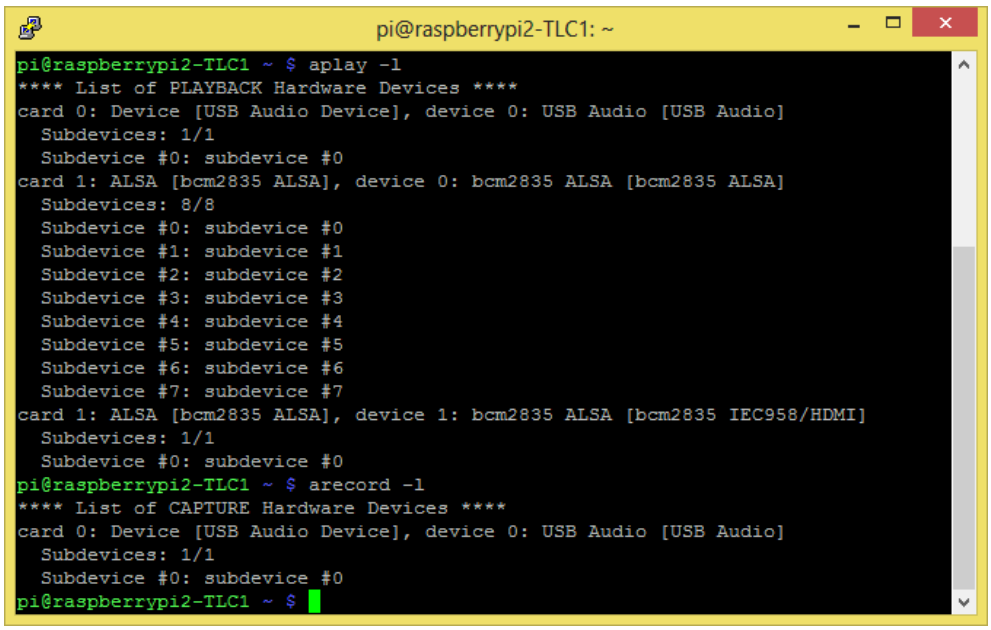


Figure 2.6: ***aplay*** and ***arecord*** outputs

```
0  arecord -l
```

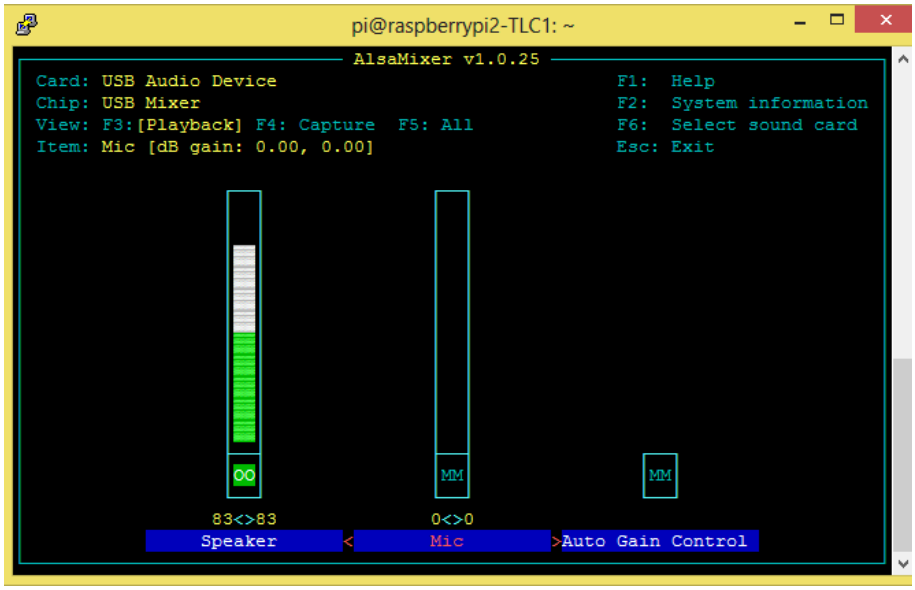
Listing 2.6: Linux command: ***arecord -l***

In order to simultaneously use both the input and the output, it is necessary to check the corresponding sound levels (volumes) through the command in Listing:2.7.

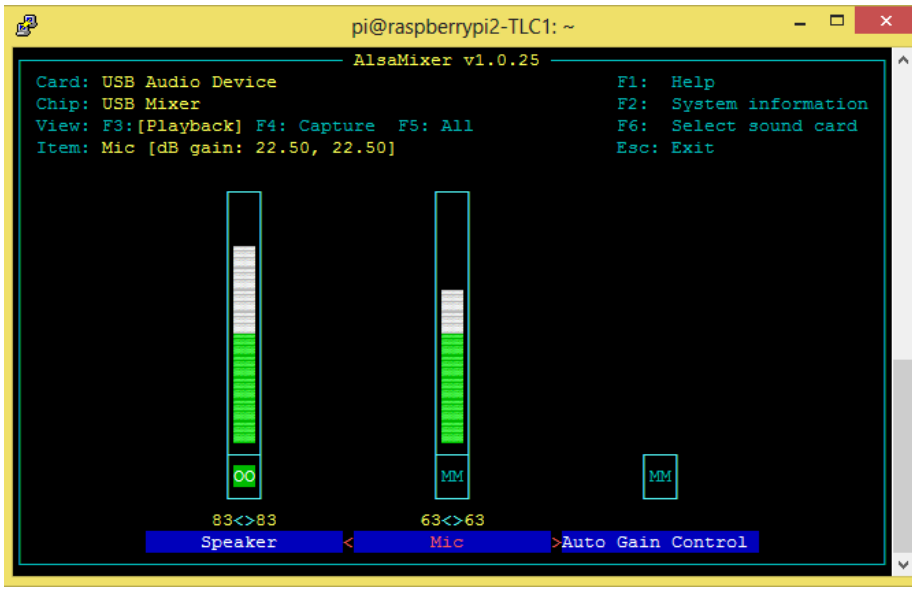
```
0  alsamixer
```

Listing 2.7: Linux command: ***alsamixer***

The *alsamixer* command opens the sound level configuration window (Fig.2.7): It is important to notice that the microphone audio level is set at 0 by default. This means that no signal acquisition can be carried out with the default setting. Of course it is possible to select the entry to be modified by using the arrows in the keyboard and to raise or lower the level for the selected audio device, for both input and output. With the function key F6 it is also possible to access the settings of other sound cards possibly present.



(a) Microphone off

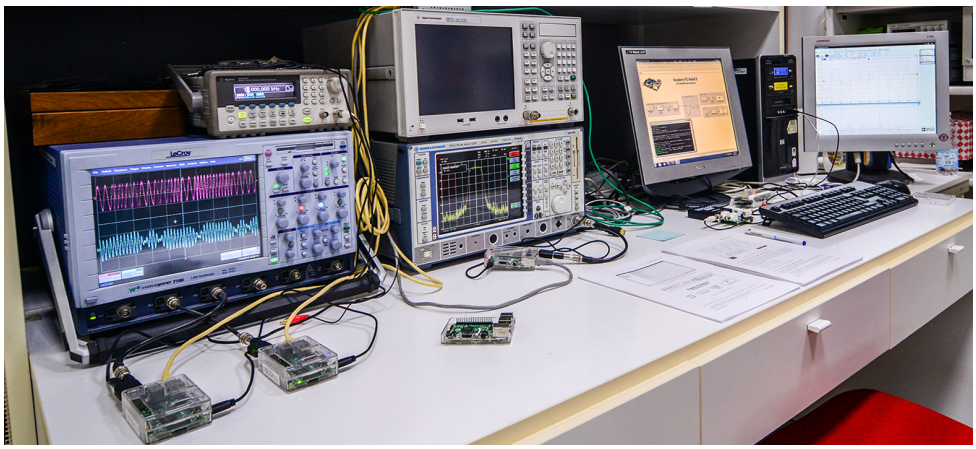


(b) Microphone on

Figure 2.7: Alsamixer

Chapter 3

The workstation



In the following chapters we will describe some didactic experiences concerning the Simulink modelling and the subsequent hardware implementation of telecommunication systems and digital signal processing algorithms.

Apart from the Raspberry Pi2 board, such activities require a PC hosting MATLAB and Simulink, as well as instruments for the generation and analysis of signals in the frequency and time domains. The equipment includes also cables and adapters suitable for interconnecting the Raspberry Pi2 to both the PC and the instruments.

The workstation setup and the equipment required will be discussed in the following sections.

3.1 Personal Computer

The experimental activities that will be presented in the following chapters require, first of all, a PC equipped with MATLAB and Simulink. In particular, the experiences described below have been realized with MATLAB R2015a equipped with the following libraries:

- Communications System Toolbox;
- DSP System Toolbox;
- Data Acquisition Toolbox;

- Fixed-Point Designer;
- Instrument Control Toolbox;
- Signal Processing Toolbox.

3.2 The equipment

The experimental activities described in the following will result in the realization of systems able, in general, to generate and process signals.

The system modelling stage, carried out using Simulink, and the Raspberry Pi2 implementation, will be thus followed by a signal measurement phase, aimed at verifying the correct functioning of the system designed and to confirm, through experimental observations, the theoretical concepts concerning the implemented system.

The equipments required for the measurement campaign are typically available in every didactic lab for electronics and telecommunications, with particular reference to:

- **Signal generator**. It is the classic device used to generate signals with user-defined characteristics. In its simplest versions it can generate periodic signals (sine waves, square pulse trains, ...) and particular aperiodic signals (Gaussian noise, single square pulse, ...). It will be mostly used as a sine wave generator, in order to provide the carrier needed by some of the implemented transmitters, or the input signal to test the digital filtering systems.

Figure 3.1: Signal generator

- **Oscilloscope**. In the majority of cases, the observation of signals will focus on their behaviour in the time domain. This task is performed through an oscilloscope, that will be employed in most of the experimental activities described below.
- **Spectrum analyser**. To investigate the spectrum of a signal it is necessary to perform a frequency domain analysis. The tool required for this kind of investigation is the spectrum analyzer, which is able to display the power distribution of a signal along the frequency axis.

It is worth noting that the signals that will be generated/processed by our systems are within the [0 24 kHz] band, owing to the characteristics of the ADC/DAC described in Chapter 2. For the generation or analysis of such signals there is no need for sophisticated instruments; thus, the basic instruments typically available in a didactic lab are normally sufficient for the experimental activities described below.

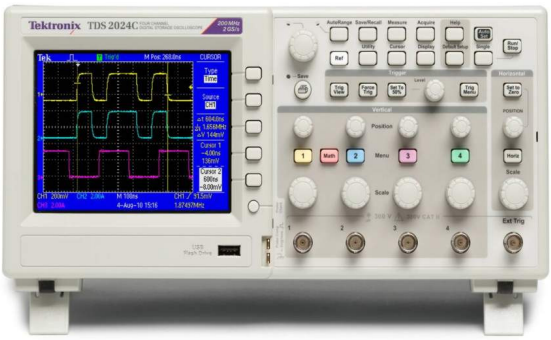


Figure 3.2: Oscilloscope

Figure 3.3: Spectrum analyser

Nonetheless, even basic oscilloscopes and spectrum analysers are costly devices for students or hobbyists. Apparently, therefore, the experiences here presented cannot be performed "at home", where lab instruments are usually not available.

Indeed, this is not true. Recently, low cost, multi-purpose instruments have been conceived for low frequency applications. On this regard the *Digilent Analog Discovery* device, shown in Fig.3.4 is worth special attention.

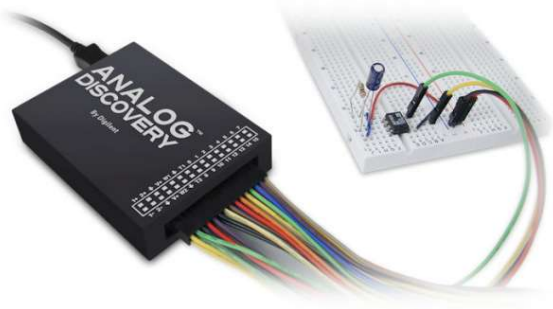


Figure 3.4: Digilent Analog Discovery

When connected to the PC through the USB port, this device is able to generate and acquire signals. With a moderate price, in the order of 270$, this tool can operate as signal generator, oscilloscope, spectrum analyser, network analyser, logic state analyser, digital signal generator

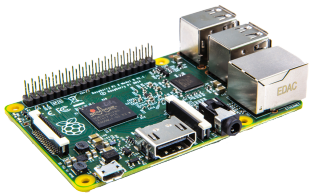
and power supply. The user interface of each single instrument, displayed on the PC's monitor, shows the same knobs, sliders and buttons of the "full hardware" instrument, allowing the user to perform the measurement activity as he was in a lab.

Of course, the upper limit of the bandwidth that can be handled by the *Digilent Analog Discovery*, in the order of some MHz, cannot be compared with that of more sophisticated and expensive instruments, however it is more than adequate for the didactic experiences described below. In this case, therefore, the signal generator, the oscilloscope and the spectrum analyser can be conveniently replaced by this multifunction tool.

3.3 The workstation

The experimental activities described in the following chapters require a PC hosting MATLAB/Simulink, a Raspberry Pi2 model B (Fig.3.5(a)), the instruments described in the previous section, and the following items:

- *Nr.1 USB-micro USB cable* (Fig.3.5(b))
- *Nr.1 micro SD memory card* (Fig.3.5(c))
- *Nr.1 Network cable* (Fig.3.5(d))
- *Nr.2 3.5mm-RCA jack cables* (Fig.3.5(e))
- *Nr.1 External sound card* (Fig.3.5(f))
- *Nr.1 USB-LAN adapter* (Fig.3.5(g))
- *Nr.2 BNC-RCA adapters* (Fig.3.5(h))



(a) Raspberry Pi2

(b) USB-micro USB cable

(c) Micro SD memory card

(d) LAN cable

(e) Stereo 3.5mm-RCA jack cable

(f) External sound card

(g) USB to LAN adapter

(h) RCA-BNC adapter

Figure 3.5: Equipment

- **USB-micro USB cable**. It is used to connect the Raspberry Pi2 to the PC's USB port, with the only aim to provide the power supply.
- **Micro SD memory card**. It contains the Raspberry Pi2 operating system, whose installation procedure is described in Chapter 1.

- **Network cable**. It is used to connect the Raspberry Pi2 with the USB to LAN adapter (Fig.3.5(g)), as described in Chapter 1.
- **3.5mm-RCA jack cables**. They are used to connect the sound card, equipped with 3.5 mm female jacks (Fig.3.5(f)), to the instruments. RCA-BNC adapter (Fig.3.5(h)) are also needed.
- **External sound card**. It is used to provide the Raspberry Pi2 with an analog input (microphone input), otherwise absent, and with a second analog output (headphones output). See Chapter 2 for further details.
- **USB to LAN adapter**. It can be used to connect the Raspberry Pi2 with the PC, providing an interface between the Raspberry Pi2 ethernet port and the PC's USB port. See Chapter 1 for further details.
- **RCA-BNC adapter**. It is used to connect the 3.5mm-RCA jack cable to the instruments, usually equipped with BNC connectors (Fig.3.5(e)).

Fig.3.6 shows an example of workstation setup.

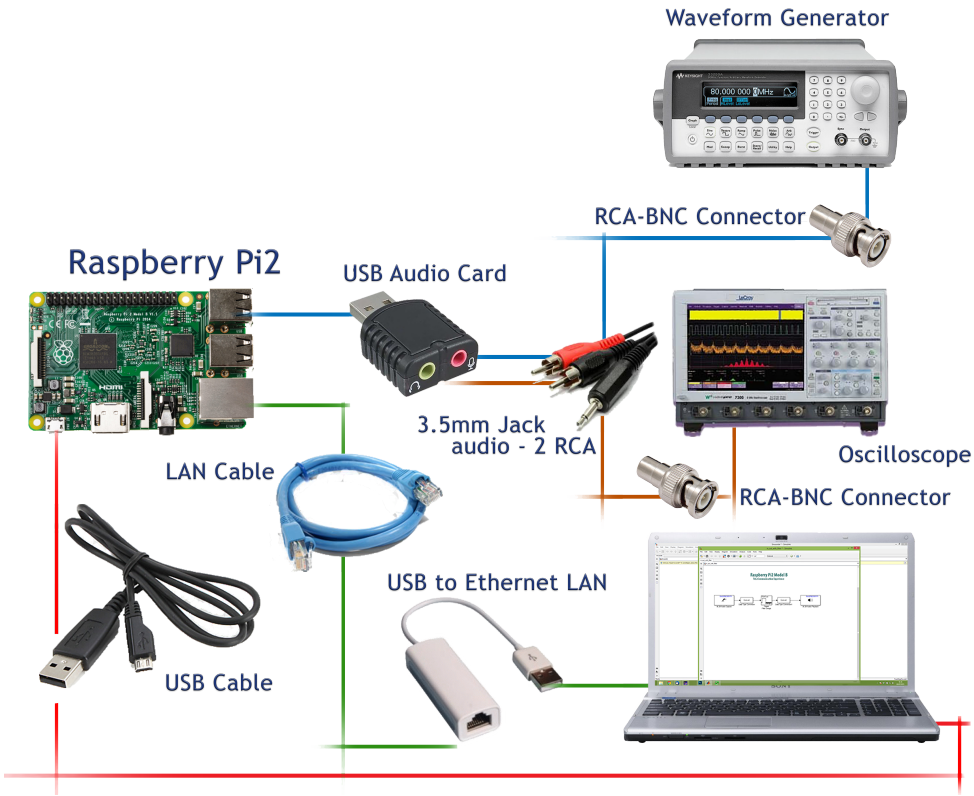
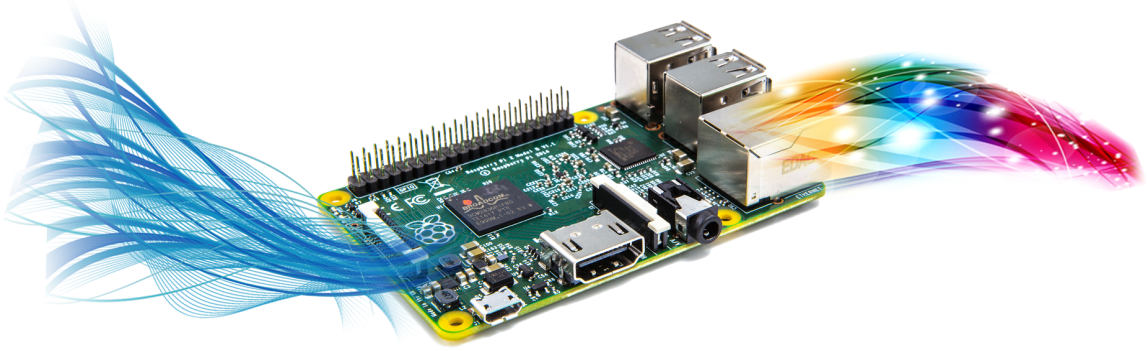


Figure 3.6: Example of workstation

Chapter 4

Raspberry Pi2 as a signal generator



In this chapter we present the first example of Simulink project conceived to be implemented on a Raspberry Pi2 board. The project, a sine wave generator, is intentionally simple, in order to focus the attention on the steps for its compilation and hardware implementation. Indeed, this project is basically a pretext to show how to generate the executable of a Simulink project and launch its execution on Raspberry Pi2 boards.

Once these procedures become familiar, the complexity of the projects will gradually increase, until getting to the implementation of the OFDM transmitter described in the last chapter.

4.1 Equipment required for this experience

The experimental activity described in this chapter requires an oscilloscope and the following equipment:

- *Nr.1 Raspberry Pi2* (Fig.4.1(a))
- *Nr.1 Micro SD memory card* (Fig.4.1(c))
- *Nr.1 Network cable* (Fig.4.1(e))
- *Nr.1 3.5mm-RCA jack cable* (Fig.4.1(g))
- *Nr.1 USB-micro USB cable* (Fig.4.1(b))
- *Nr.1 USB-LAN adapter* (Fig.4.1(d))
- *Nr.1 External sound card* (Fig.4.1(f))
- *Nr.1 BNC-RCA adapter* (Fig.4.1(h))

(a) Raspberry Pi2 (b) USB-micro USB cable (c) Micro SD memory card (d) USB to LAN adapter

(e) LAN cable (f) External sound card (g) 3.5mm-RCA stereo jack cable (h) RCA-BNC adapter

Figure 4.1: Equipment

4.2 Raspberry Pi2 as a sine wave signal generator

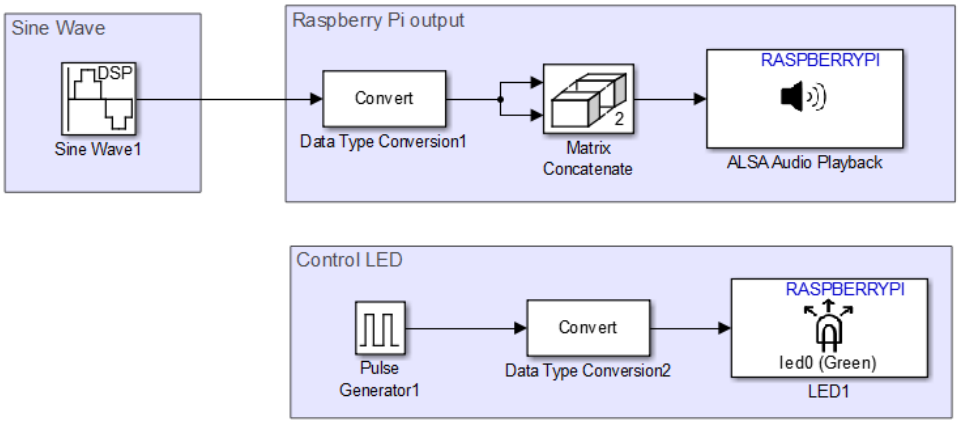


Figure 4.2: Simulink project of the sine wave signal generator

The first experimental activity proposed is aimed at modelling and implementing on a Raspberry Pi2 board a sine wave signal generator with configurable amplitude, frequency and phase offset. The corresponding Simulink model is shown in Fig.4.2, whereas Fig.4.3 shows the interconnection of the different devices composing the workstation.

In order to be as clear as possible, the scheme in Fig.4.2 highlights three macroblocks, called ***Sine Wave***, ***Raspberry Pi output*** and ***Control LED***, corresponding to three different tasks, associated to the objective of the project (***Sine Wave*** macroblock), to the output management (***Raspberry Pi output*** macroblock) and to the execution monitoring (***Control LED***

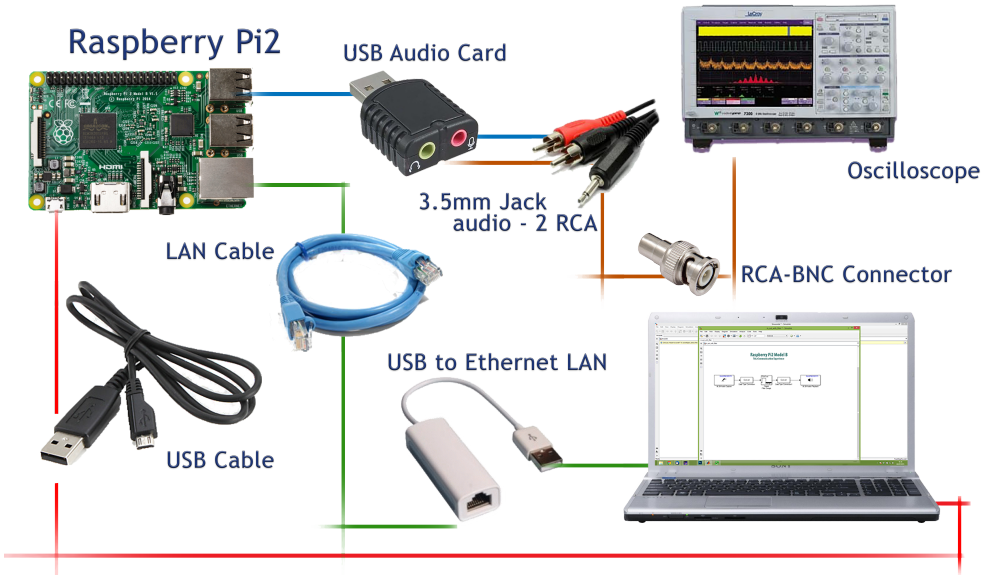


Figure 4.3: Connection scheme

macroblock), respectively.

The task of each macroblock is described below, whereas the detailed description of the elementary blocks that constitute each macroblock is reported in Section 4.3.

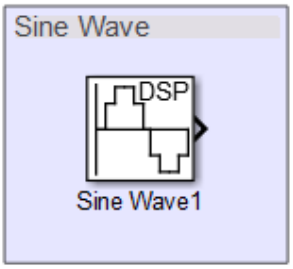


Figure 4.4: Sine Wave macroblock

4.2.1 *Sine Wave* macroblock

The ***Sine Wave*** macroblock, shown in Fig.4.4 and described in detail in Section 4.3.1, is constituted by a single elementary block, hence, strictly speaking, it should not be considered a macroblock. It is, however, the project's core, because it is in charge of generating the samples of a sine wave with configurable amplitude, frequency and phase offset. This block alone, therefore, accomplishes the project's objective, thus deserving the "macroblock" title.

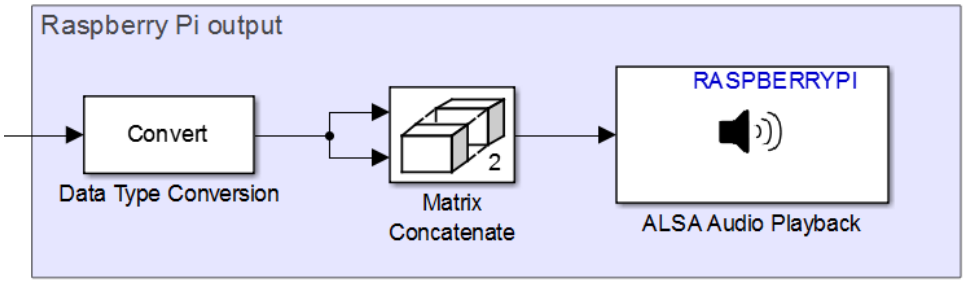


Figure 4.5: Raspberry Pi2 output system

4.2.2 *Raspberry Pi output* macroblock

The ***Raspberry Pi output*** macroblock, shown in Fig.4.5, represents the Raspberry Pi2's analog output. This macroblock has the function to adapt the signal at its input port to the format required by the Raspberry Pi2's output port, represented by the ***ALSA Audio Playback*** block. Such macroblock has, thus, a general nature and appears in all (or almost all) the projects described below.

The functioning and the configuration of the elementary blocks ***Data Type Conversion***, ***Matrix Concatenate*** and ***ALSA Audio Playback*** are detailed in Sections 4.3.2, 4.3.3, and 4.3.4.

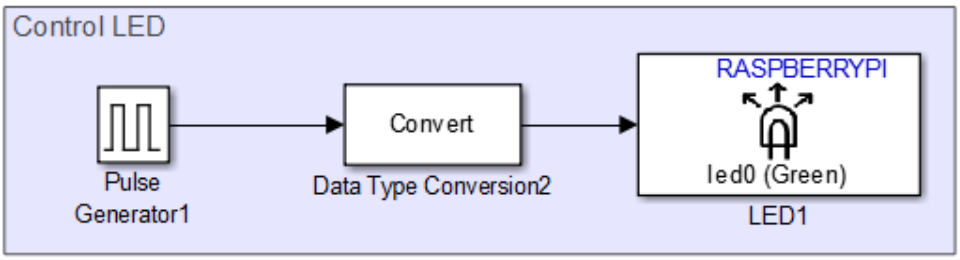


Figure 4.6: ***Control LED*** macroblock

4.2.3 *Control LED* macroblock

The only aim of the ***Control LED*** macroblock, shown in Fig.4.6, is to intermittently turn on and off the Raspberry Pi2's led during the project execution, visually confirming the that the project is running.

This macroblock is not strictly part of the actual sine wave generator (***Sine Wave***+***Raspberry Pi output***), to which, in fact, is not even connected. It is a general macroblock and will appear in all the projects described below.

The introduction of this macroblock, operating in parallel with the previously described ones, also shows that the Raspberry Pi2 can simultaneously accomplish different tasks and handle different outputs.

The functioning and the configuration of the ***Pulse Generator***, ***Data Type Conversion*** and ***LED*** elementary blocks are described in Sections 4.3.5, 4.3.2 and 4.3.6.

4.3 Elementary blocks used

The list of the elementary blocks used for realizing the project is provided hereafter, along with the reference to the sections in which their functioning is described.

- *Sine Wave* (Section 4.3.1)
- *Data Type Conversion* (Section 4.3.2)
- *Matrix Concatenate* (Section 4.3.3)
- *ALSA Audio Playback* (Section 4.3.4)
- *Pulse Generator* (Section 4.3.5)
- *LED* (Section 4.3.6)

4.3.1 Sine Wave

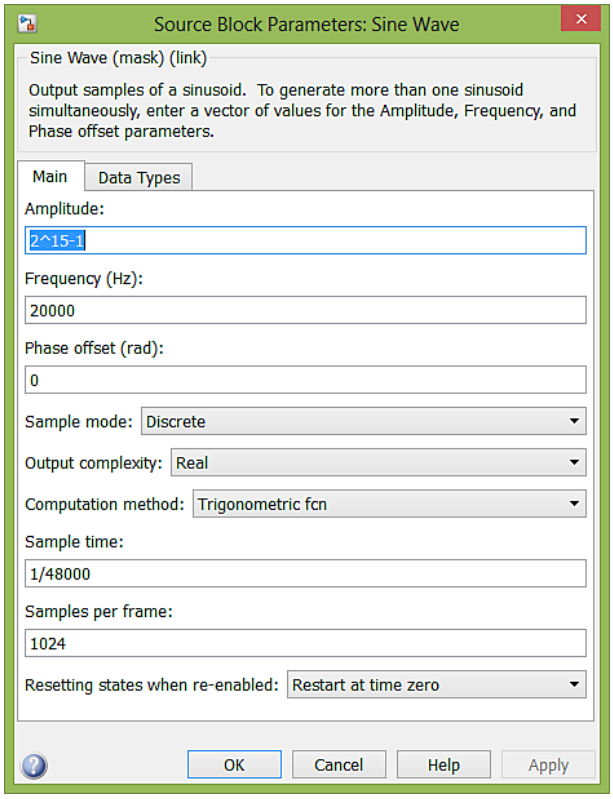


Figure 4.7: Configuration window of the ***Sine Wave*** block

The ***Sine Wave*** block generates at its output port a sampled sine wave, whose samples are taken at instants spaced out according to the quantity defined in the *Sample Time* field (see the block's configuration window shown in Fig.4.7).

Since the maximum sampling frequency supported by the external sound card adopted in our projects (see chapter 2) is 48000 samples/s, we chose *Sample Time*=1/48000 s. This obviously establishes a limit to the highest frequency of the sine wave that can be generated, that must not exceed the theoretic limit of 24 kHz nor the practical limit of 20 kHz.

The sine wave generated by the ***Sine Wave*** block can be properly set up by choosing the values of the corresponding parameters.

The first entry required by the block's configuration window, shown in Fig.4.7, is the amplitude, that must be entered in the *Amplitude* field. When choosing this value, it is necessary to bear in mind that the signal dynamic range is limited by the 16-bit representation established by the ***ALSA Audio Playback*** block (sect.4.3.4), that operates with data in the *int16* format. The highest value that can be assigned, then, will be equal to $2^{15} - 1$, corresponding to the highest integer value that can be represented with 15 bits + 1 sign bit.

The second and the third fields refer to the signal frequency, set as an example at 20000 Hz in the *Frequency* field, and the phase offset, set at 0 in the *Phase offset* field.

The *Sample Mode* field keeps the default value *Discrete*, that determines the generation of discrete-time samples.

The *Output Complexity* field, set at *Real*, produces a real output, as an alternative to a complex exponential output.

The *Computation Method* field, determining which method must be used to generate the sine wave, maintains the default *Trigonometric fnc* setting.

The *Sample per frame* field defines the *frame* length, that is the number of samples grouped to be transmitted at each step to the following block. In order to increase the efficiency, in fact, the Raspberry Pi2 board (and in general all *embedded* systems) operates on blocks of data (the *frames*) rather than on a single datum at a time. For this particular project the field *frame* is set at 1024.

The last parameter that is possible to set is called *Resetting States when re-enable*. By setting it at *Restart at time zero*, the block starts again with the initial settings in the event of a reset (this choice is only significant if the block is provided with an "enable" input, not present in this project).

4.3.2 Data Type Conversion

The ***Data Type Conversion*** block converts an input signal of any Simulink data type to the data type specified in its configuration window.

For the project here considered, in particular, the ***ALSA Audio Playback*** block specifically requires input data in the *int16* format (16-bit signed integer). The data type conversion from *Real* data, supplied by the ***Sine Wave*** block, into *int16* data is performed by the ***Data Type Conversion1*** block (in the ***Raspberry Pi output*** macroblock) by setting the *Inherit: Inherit via back propagation* mode in the *Output data type* field, as shown in Fig.4.8. In this way the ***Data Type Conversion1*** block automatically adapts the output data type to the requirements of the downstream stages.

In the same way, the ***Data Type Conversion2*** block (in the ***Control LED*** macroblock), set as *Inherit: Inherit via back propagation*, converts the signal produced by the ***Pulse Generator*** block into the boolean format required by the ***LED*** block.

4.3.3 Matrix Cancatenate

The ***Matrix Concatenate*** block concatenates multiple signals in a single output signal. The output signal can be either a *vector* or a *multidimensional array*.

In the project here considered, this block is used in *multidimensional* mode, in order to produce a two-channel output signal (that is, a stereo signal), starting from the two mono signals

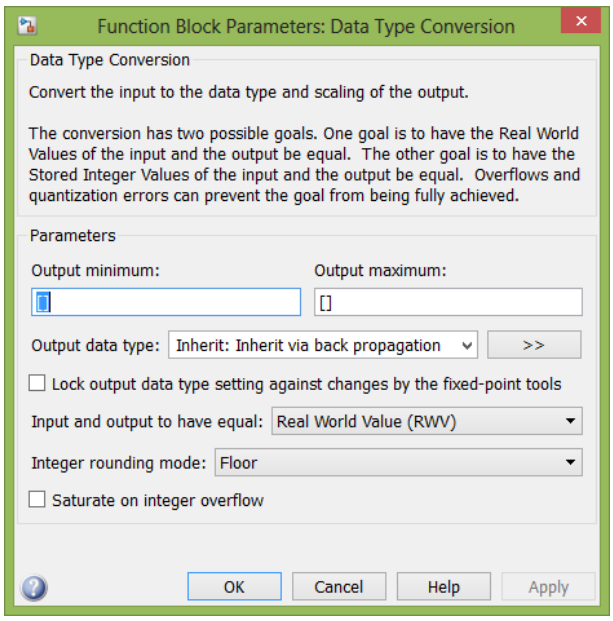


Figure 4.8: Configuration window of the ***Data type Conversion*** block

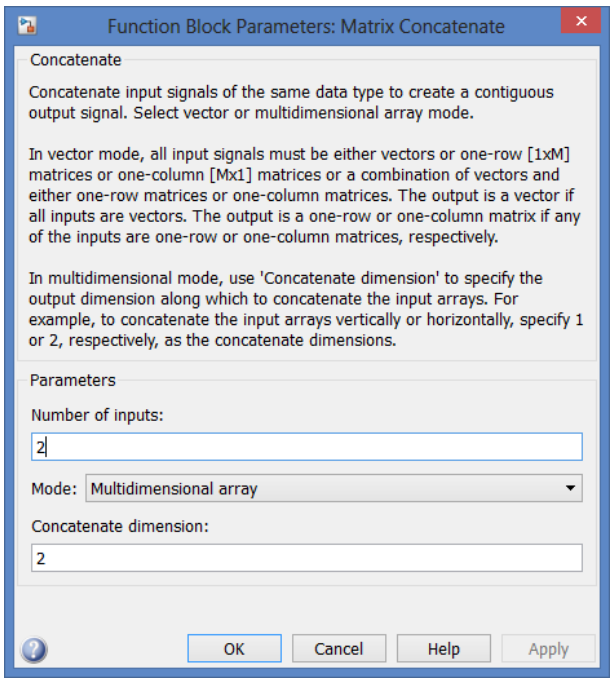


Figure 4.9: Configuration window of the ***Matrix Concatenate*** block

at its input. This operation is needed as the ***ALSA Audio Playback*** block, that follows the ***Matrix Concatenate*** block, requires a stereo input signal. For this reason, the *Number of inputs* field of the block's configuration window must be set at 2. The same for the *Concatenate Dimension* field (see Fig.4.9).

As a result, the two signals (the one identical to the other), at the input of the block, organized in *frames* of 1024 values, are concatenated and transferred to the output in the form of a single signal of dimension [1024×2], which is intended by the following ***ALSA Audio Playback*** block as a stereo signal (with two identical channels).

4.3.4 ALSA Audio Playback

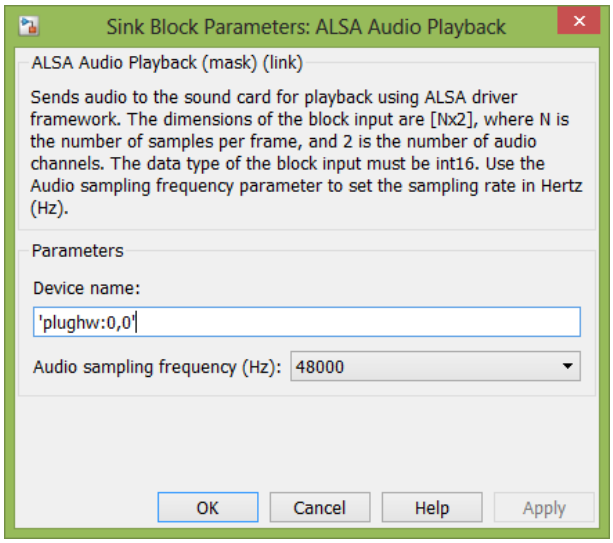


Figure 4.10: Configuration window of the ***ALSA Audio Playback*** block

The ***ALSA Audio Playback*** block represents the Raspberry Pi2's analog output. Its task is to perform the digital-to-analog conversion of the signal and send it to the sound card for playback. This block relies on the Raspberry Pi2's ALSA audio driver, that manages all audio devices connected to the Raspberry Pi2, including the USB sound card previously described.

The input signal must have a dimension of [Nx2], where N is the number of samples in each *frame* and 2 is the number of audio channels (in this specific case N=1024). Each sample must be represented in the *int16* format.

With reference to the configuration window shown in Fig.4.10, the sampling rate is selected in the *Audio sampling frequency* field. In this model, and in all the following ones, a sampling rate of 48000 samples/s was used, corresponding to the highest possible value.

The sound card's identifier must be entered in the *Device Name* field: The input is defined by the syntax *'plughw:card,device'*, where the two parameters *card* and *device* can be easily obtained by using the *aplay -l* command in a Linux Shell (sect.2.1). The result will be an on-screen list of all the devices connected to the Raspberry Pi2 and their corresponding parameters *card* and *device*.

If the configuration of the sound card described in chapter 2 is properly performed, the *Device Name* field will always have the form *'plughw:0,0'*.

4.3.5 Pulse Generator

The ***Pulse Generator*** block generates square wave pulses at regular intervals, that can be set through the configuration window shown in Fig.4.11.

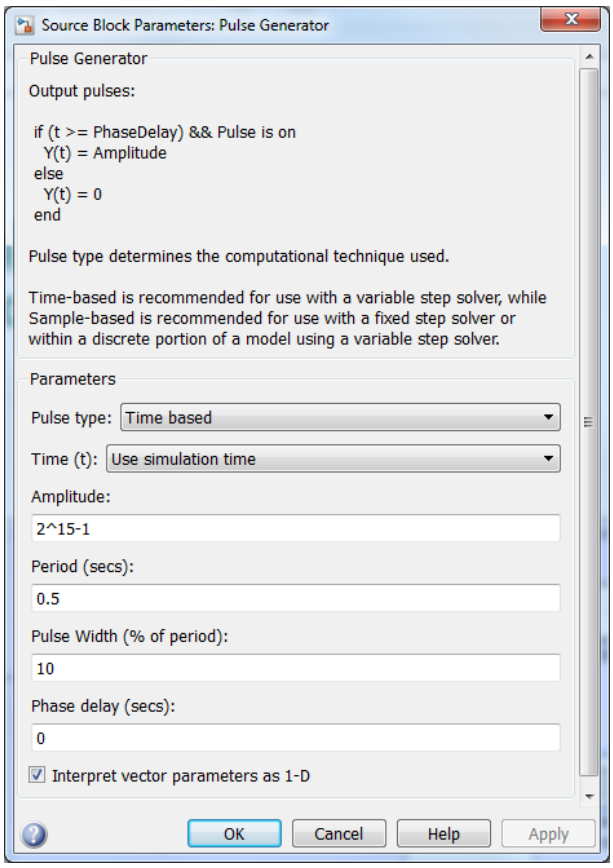


Figure 4.11: Configuration window for the ***Pulse Generator*** block

The only relevant parameters for this block are the *Amplitude*, set at its highest possible value $2^{15}-1$, the repetition *Period*, set at 0.5 s, and the *Pulse Width*, set at 10%. The remaining fields are completed with default settings.

This signal is used to turn on and off the Raspberry Pi2's LED when the project is running.

4.3.6 LED

The ***LED*** block is part of the Simulink library containing specific blocks for Raspberry Pi boards. Its task is to turn on or off the user-controllable LED provided by the Raspberry Pi2 hardware. The corresponding configuration window is shown in Fig.4.12.

The only fields of interest are those concerning the choice of the Raspberry Pi model used, defined in the *Board* field, and the LED to be controlled, through the *LED* field. The proper settings are shown in Fig.4.12.

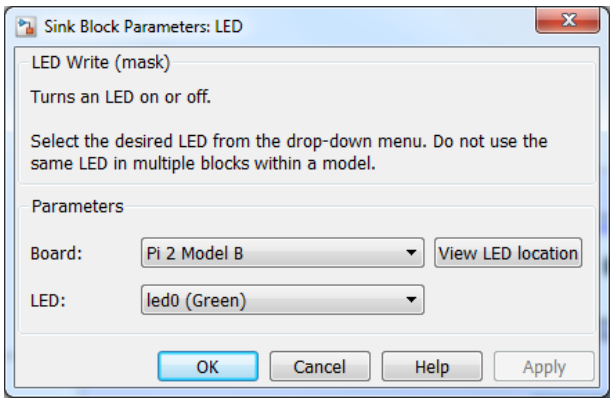


Figure 4.12: Configuration window of the ***LED*** block

4.4 Settings for the hardware execution of the project

Once the Simulink project has been realized and tested through simulations, the Raspberry Pi2 can step in. In order for the project to be implemented on the hardware, it is essential to provide Simulink with the necessary information to identify the Raspberry Pi2 in which to download the executable file.

Within the Simulink environment select the ***tools/run on target hardware/options*** menu, as shown in Fig.4.13, in order to select the target hardware. In this case the choice is, obviously, "Raspberry Pi" (Fig.4.14).

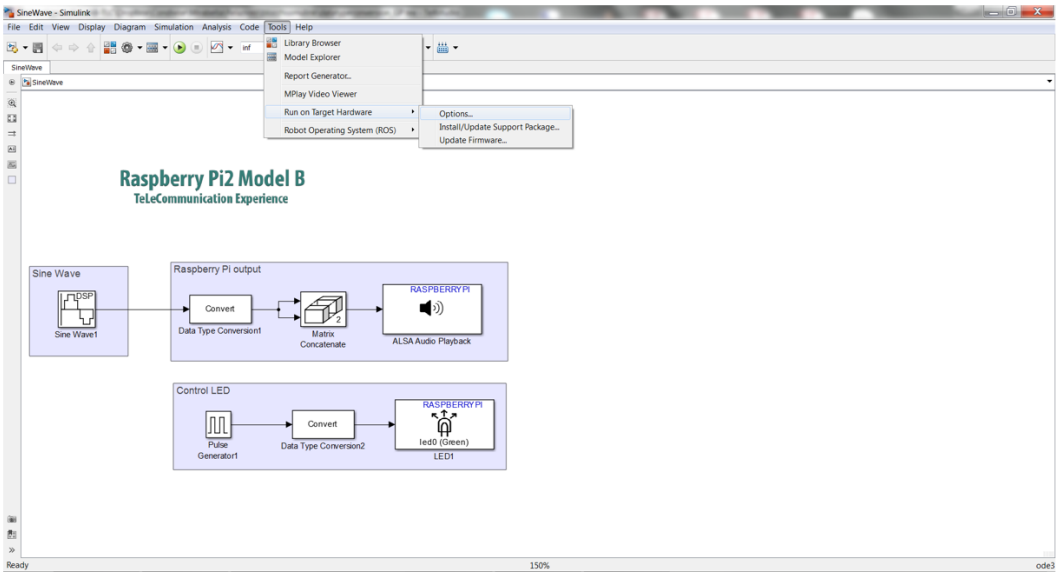


Figure 4.13: Prepare to run

In the following window (Fig.4.15) you are required to enter the parameters concerning *Host Name, User Name, Password, Build directory.*

If the default configuration is unchanged, the parameters will be those used during the OS installation phase, shown in Fig.1.27. Otherwise, the specific parameters chosen during the OS

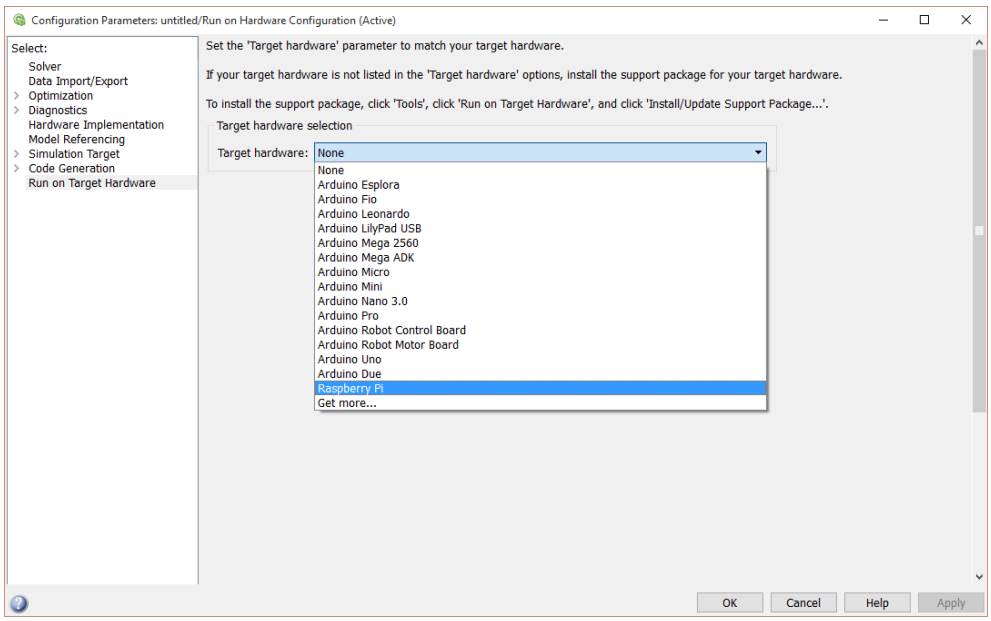


Figure 4.14: Target hardware setting

installation phase must be used.

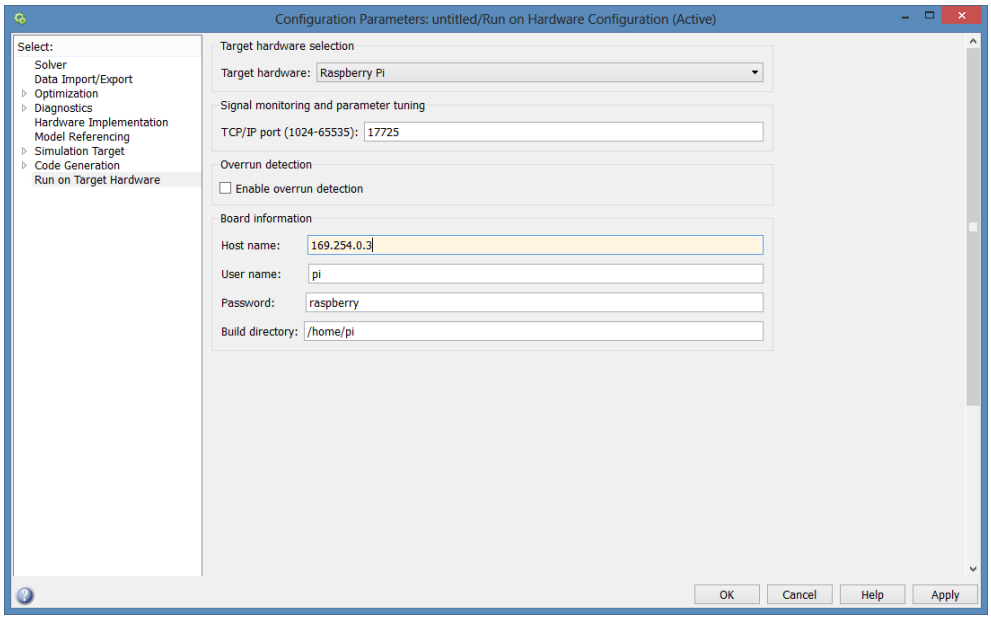


Figure 4.15: Configuration parameters

It is possible to set just one Raspberry Pi2 at a time, even in the event of more than one device connected to the PC through a network switch. Each Raspberry Pi2 will be unambiguously identified through its *Host Name*.

The *Enable overrun detection* is a functionality that can be activated at the user's discretion.

With its activation, it is possible to monitor the occurrence of possible situations in which the Raspberry Pi2 execution does not respect the timing required by the project.

Once Simulink has been properly configured to interact with the target Raspberry Pi2, it is possible to compile the project, generating the executable, and perform the ***Deploy to Hardware*** step (Fig.4.16). In order to successfully complete this operation, the user must have the writing rights on the Matlab *current folder*.

The project will be automatically executed as soon as the *deploy* procedure is completed.

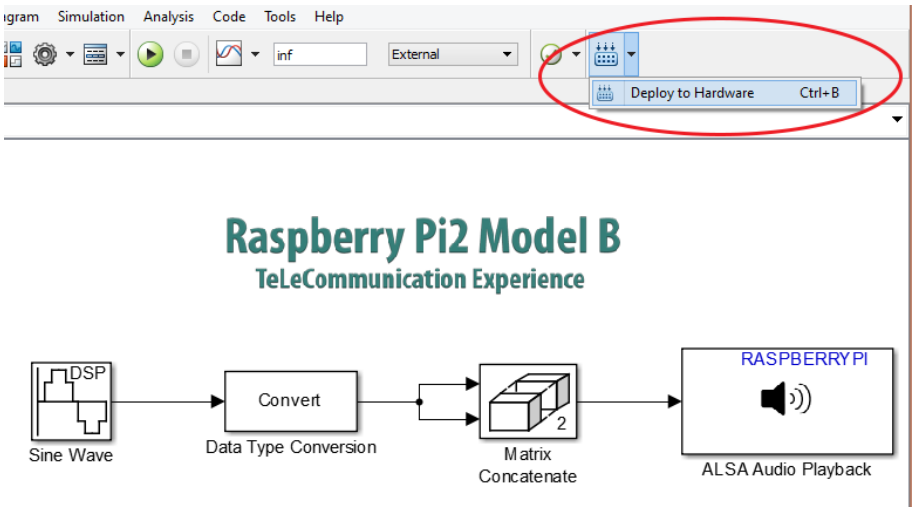


Figure 4.16: Deploy To hardware

It is now possible to observe the sine wave generated by the Raspberry Pi2 through an oscilloscope connected to the sound card output, as shown in Fig.4.3. The expected result is shown in Fig.4.17, representing an example of actual measurement performed in our lab.

Detailed information about the different ways to start and stop the execution of a Simulink project on a Raspberry Pi2 board is given in the following section.

4.5 Hardware execution of the project

Any Simulink project already loaded on the hardware, through the previously described "*Deploy to Hardware*" procedure, can be executed independently from Simulink. The executable file is saved in the *Build Directory* of the the micro SD memory card. Generally speaking, there are three different modes to launch the execution of a project on a Raspberry Pi2:

1. Launching the execution within Simulink.

2. Launching the execution with Matlab commands.

3. Launching the execution with Linux commands.

Each mode is described in the following sections.

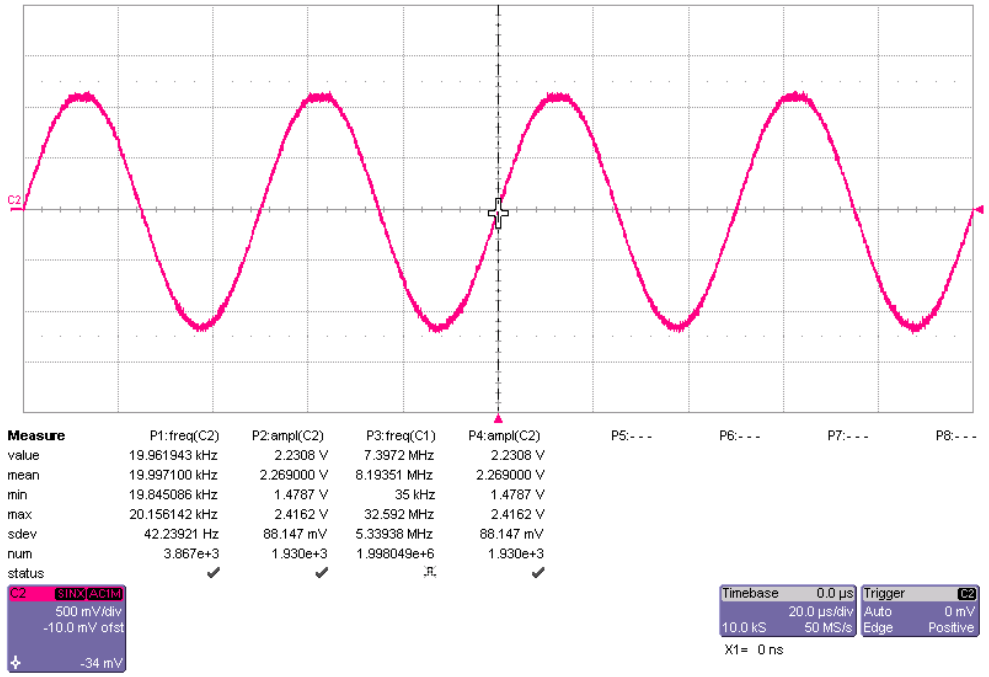


Figure 4.17: Signal displayed on the oscilloscope

```
top - 18:31:19 up 14 min,  1 user,  load average: 0.48, 0.18, 0.10
Tasks: 107 total,   1 running, 106 sleeping,   0 stopped,   0 zombie
%Cpu(s):  4.4 us,  0.2 sy,  0.0 ni, 95.2 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
KiB Mem:    762420 total,   141164 used,   621256 free,    17092 buffers
KiB Swap:   102396 total,        0 used,   102396 free,    74724 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 2499 root      rt   0  3668 2144 1968 S  20.1  0.3   0:11.38 SineWave
  175 root      20   0  2896 2224 1692 S   1.3  0.3   0:01.16 udevd
  682 root      20   0  2892 1920 1380 S   1.0  0.3   0:00.74 udevd
 2504 pi        20   0  4680 2388 2032 R   0.7  0.3   0:00.16 top
    7 root      20   0     0    0    0 S   0.3  0.0   0:00.53 rcu_preempt
   74 root      20   0     0    0    0 S   0.3  0.0   0:00.09 kworker/3:1
    1 root      20   0  2152 1328 1224 S   0.0  0.2   0:00.77 init
    2 root      20   0     0    0    0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S   0.0  0.0   0:00.04 ksoftirqd/0
    5 root       0 -20     0    0    0 S   0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0     0    0    0 S   0.0  0.0   0:00.05 kworker/u8:0
    8 root      20   0     0    0    0 S   0.0  0.0   0:00.00 rcu_sched
    9 root      20   0     0    0    0 S   0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0     0    0    0 S   0.0  0.0   0:00.01 migration/0
   11 root      rt   0     0    0    0 S   0.0  0.0   0:00.02 migration/1
   12 root      20   0     0    0    0 S   0.0  0.0   0:00.02 ksoftirqd/1
   14 root       0 -20     0    0    0 S   0.0  0.0   0:00.00 kworker/1:0H
```

Figure 4.18: Linux command "***top***"

4.5.1 Launching the execution within Simulink

The hardware execution of a project can be launched directly from Simulink, by means of the "*Deploy to Hardware*" (Fig.4.16) procedure or using the "*External Mode*" simulation setting.

- ***Deploy to Hardware***: with the previously described *Deploy to Hardware* procedure, the

executable is automatically launched as soon as the project compilation is completed.

Please note that the *Deploy to Hardware* procedure saves the executable and its auxiliary files on the Raspberry Pi2's memory card. This step is essential if you want to run the project later through Matlab or Linux commands, as they require the executable file to be already in the Raspberry Pi2's memory card.

It si important to underline, moreover, that the "*deploy to hardware*" mode deactivates the connection between the Raspberry Pi2 and the Simulink environment, making it impossible to stop the project within Simulink.

In order to stop the execution it is necessary to open a Linux Shell (Section 1.7), identify the running process to be stopped and force its termination, as described below.

First of all, a list of the processes running on the device must be obtained using the ***top*** command (Listing 4.1) within the Linux Shell[1], as shown in Fig.4.18:

```
0 top
```

Listing 4.1: List of running processes

Once the PID code associated to the process to be stopped, for example "SineWave", is found, the "***top***" window can be closed pressing the ***q*** key. The process can now be stopped using the ***kill*** command (Listing 4.2).

```
0 sudo kill PID
```

Listing 4.2: List of running programs

The second possibility offered by Simulink to run a project on a Raspberry Pi2 is the ***External mode***.

- ***External mode***: the execution is activated setting the *External* mode, as shown in Fig.4.19, and starting the Simulink simulation as usual, clicking *Run* (Fig.4.20).

  Contrary to the previous case, the "*External Mode*" keeps active the connection between Simulink and the Raspberry Pi2, allowing to stop the execution with the *Stop* button, as shown in Fig.4.21. In this case, however, the executable and the auxiliary files needed for a later execution outside Simulink will not be saved on the Raspberry Pi2's memory card.

  This mode is particularly appropriate during the design phase of the project. During the *External Mode* execution, in fact, it is possible to change the project's parameters (e.g., the sine's amplitude) and to observe their effects in real time.

  Please observe, however, that the execution in *External Mode* implies a greater computational burden for the Raspberry Pi2 compared to the execution activated by the *Deploy to Hardware* procedure. This is due to the continuous information exchange between the hardware and Simulink during the *External Mode* execution.

[1] Please note, by the way, that the ***top*** command provides also the computational burden *%CPU* of each process.

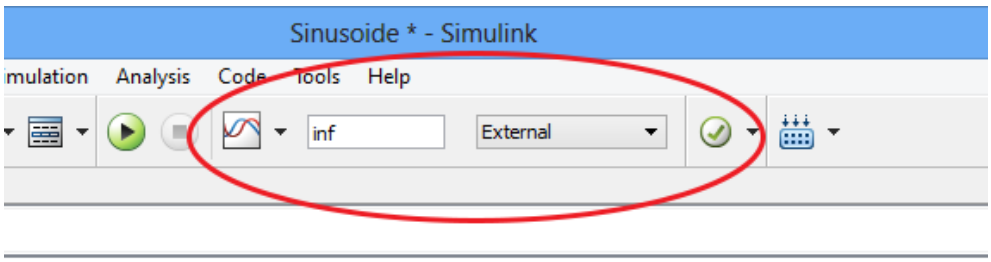


Figure 4.19: *External Mode* activation

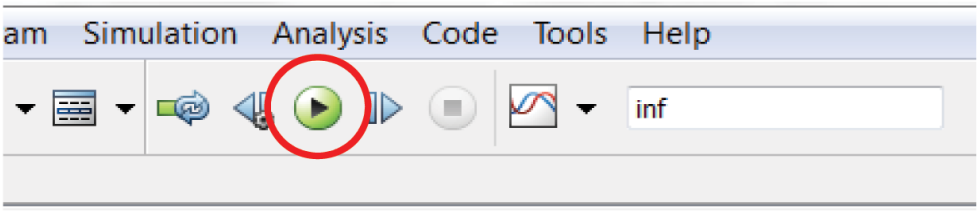


Figure 4.20: Project execution in *External Mode*

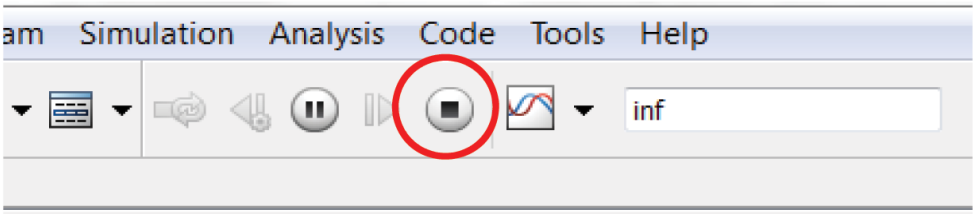


Figure 4.21: Project stop in *External Mode*

4.5.2 Launching the execution with Matlab commands

In order to launch the hardware execution of a project using Matlab commands, it is necessary to create the "***raspberrypi***" object "***h***" as shown[2] in Listing 4.3, and to invoke the command ***run***, entering as input parameter the name of the project to be launched (Listing 4.4).

```
h=raspberrypi('169.254.0.3');
```

Listing 4.3: Matlab command to create a ***raspberrypi*** object

```
run(h, 'project name')
```

Listing 4.4: Matlab command to execute a project on a Raspberry Pi2

[2]Here the ***raspberrypi*** command is used with *HostName* as the only parameter. Note that, as recalled in Section 1.7, Matlab assumes by default that *HostName* is equal to the IP address assigned to the device. The parameters *UserName*, *Password* and *BuildDir* are assumed at their default settings.

If the executable is not saved in the default ***BuildDir*** it is necessary to run the extended command "***h=raspberrypi(...)***" in Listing 4.5, specifying the new ***BuildDir***. It is now possible to launch the command ***run*** (Listing 4.4).

```
h=raspberrypi('HostName','UserName','Password','BuildDir') %new object for
    device raspberry pi
```

Listing 4.5: Matlab command to create a ***raspberrypi*** object

The project execution can be stopped at any time, using the command:

```
stop(h, 'project name')
```

Listing 4.6: Matlab command to stop the execution

4.5.3 Launching the execution with Linux commands

Any executable saved into the Raspberry Pi2's memory card cab be launched by means of the Linux Shell. The first step is to open the Linux Shell with the MATLAB commands:

```
h=raspberrypi("HostName");
h.openShell('ssh')
```

Listing 4.7: Opening a Linux Shell within MATLAB

Then the *BuildDir* of the project must be selected using the Linux command:

```
cd "BuildDir"
```

Listing 4.8: Opening the "BuildDir"

The project execution is finally launched with the Linux command

```
sudo ./"project name_rtt"/MW/"project name"
```

Listing 4.9: Project execution

For the experience here described, we chose a *BuildDir* different from the default one, as shown in Fig.4.22. In this case, in fact, the *BuildDir* is ***/home/pi/Sin***.

Coming back to the Linux Shell it is possible to access and display the *BuildDir*'s content launching the commands (see Fig.4.23)

```
cd Sin
dir
```

Listing 4.10: Entering the *BuildDir*

To run the project, it is enough to launch the command in Listing 4.11

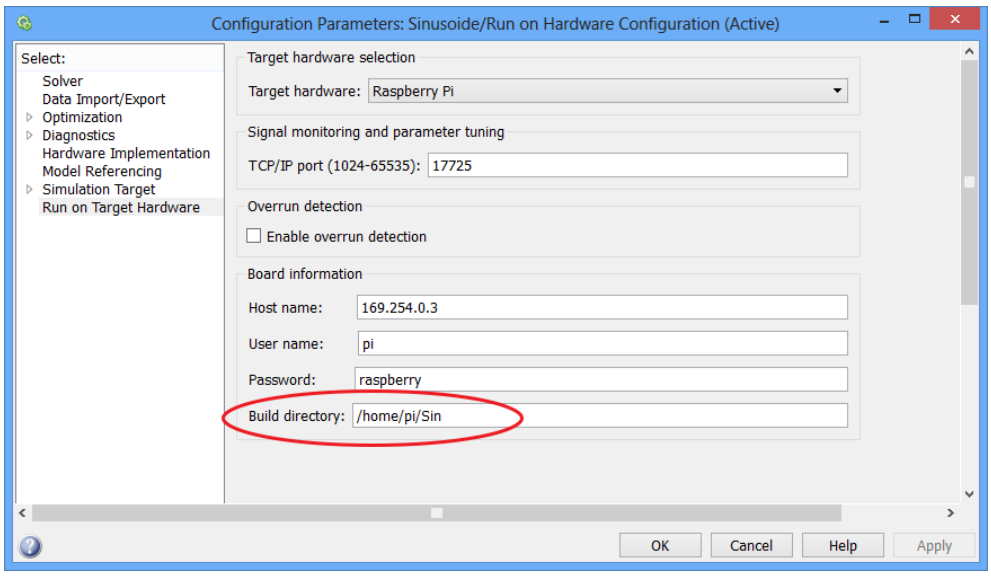


Figure 4.22: Project's Build Directory

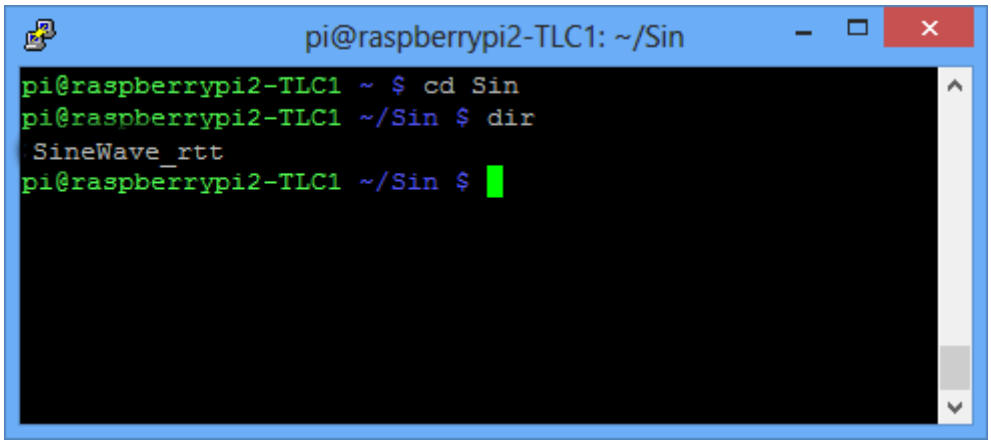


Figure 4.23: Project's Build Directory

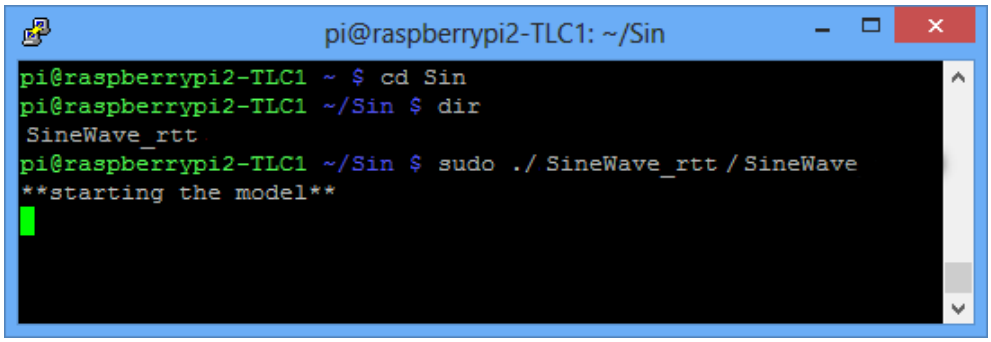


Figure 4.24: Project execution within the Linux environment

```
sudo ./SinWave_rtt/MW/SinWave
```

Listing 4.11: Linux directory access

In general, it is possible to run the project using the syntax reported in Listing 4.12, specifying each time the project name and, if necessary, the BuildDir (in case it is different from the default one). In particular, this command can be launched when the Shell is initialized, with no need to access the specific directory of the project, as done in Listing 4.8.

```
sudo ./home/pi/"BuildDir/"project name_rtt"/MW/"project name"
```

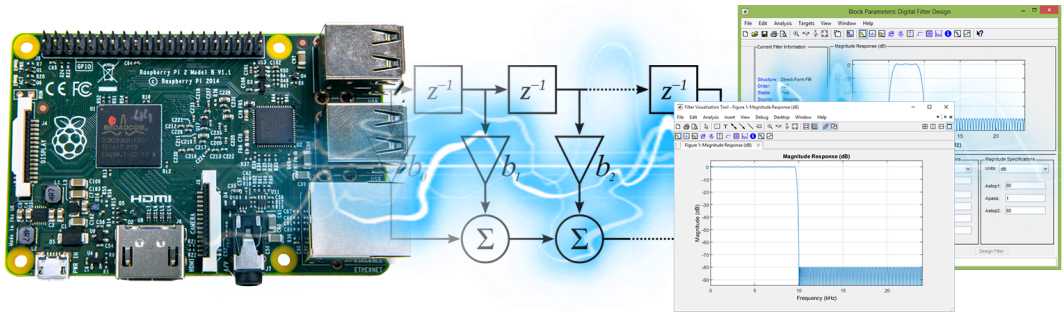
Listing 4.12: Project execution

In order to stop the execution, it is enough to press the ***Ctrl+C*** keys.

As an alternative, you can use the ***kill PID*** command, identifying in advance the PID code of the project that must be stopped with the ***top*** command (Listing 4.1) and stopping the project with the ***kill*** command (Listing 4.2).

Chapter 5

Raspberry Pi2 as a digital filter



Thanks to the external ADC/DAC introduced in Chapter 2, the Raspberry Pi2 is provided with an (otherwise absent) analog input (microphone input) and with a further analog output (headphone output). It is therefore possible to use Raspberry Pi2 boards for the digital processing of input signals.

The realization of a Finite Impulse Response (FIR) digital filter, described below, is an example of this kind of application.

5.1 Equipment required for this experience

To carry out the experimental activity explained below, a signal generator and an oscilloscope are needed, along with the following items:

- *Nr.1 Raspberry Pi2* (Fig.5.1(a))
- *Nr.1 micro SD memory card* (Fig.5.1(c))
- *Nr.1 Network cable* (Fig.5.1(e))
- *Nr.2 3.5mm-RCA jack cables* (Fig.5.1(g))
- *Nr.1 USB-micro USB cable* (Fig.5.1(b))
- *Nr.1 USB-LAN adapter* (Fig.5.1(d))
- *Nr.1 External audio card* (Fig.5.1(f))
- *Nr.2 BNC-RCA adapter* (Fig.5.1(h))

5.2 Raspberry Pi2 as digital filter

The Simulink model for implementing a FIR filter on a Raspberry Pi2 board is shown in Fig.5.2. Fig.5.3 shows, instead, the connections between the devices constituting the workstation.

(a) Raspberry Pi2 (b) USB-micro USB cable (c) Micro SD memory card (d) USB to LAN adapter

(e) LAN cable (f) External audio card (g) 3.5mm-RCA Stereo jack cable (h) RCA-BNC adapter

Figure 5.1: Equipment

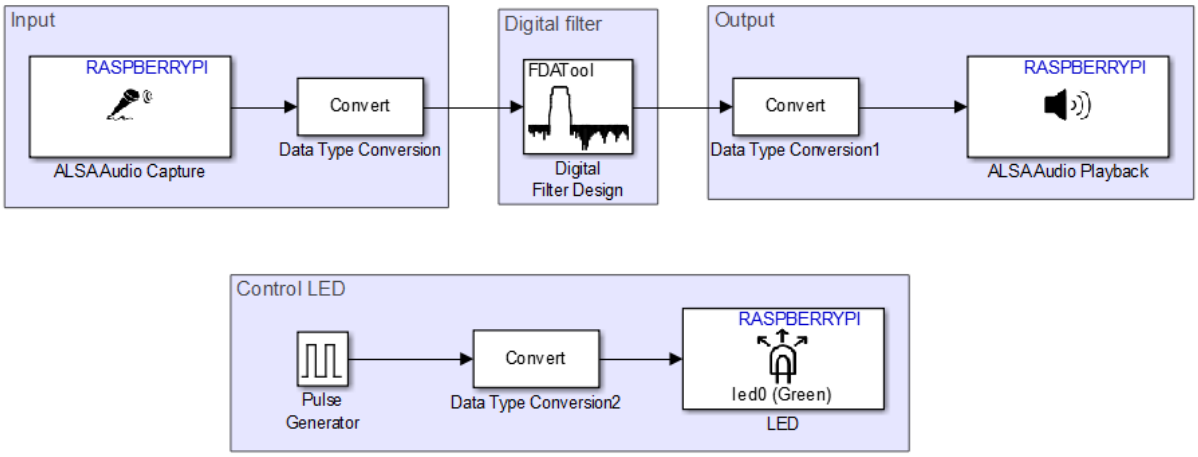


Figure 5.2: Fir filter Simulink model with Raspberry Pi2

In order to make the scheme shown in Fig.5.2 as clear as possible, four macroblocks have been highlighted: ***Input***, ***Digital filter***, ***Output*** and ***Control LED***.

The ***Input*** macroblock is the input stage of the system. The signal produced by an external signal generator is acquired by the analog-to-digital converter represented by the ***ALSA Audio Capture*** block and converted by the ***Data Type Conversion*** block from the *int16* format, adopted by ***ALSA Audio Capture*** block, into the *floating point* format required by the ***Digital Filter Design*** block.

The ***Digital filter*** macroblock represents the digital filter itself. The filter characteristics are selected by the user through the ***FDATool*** design tool, that opens when clicking on the

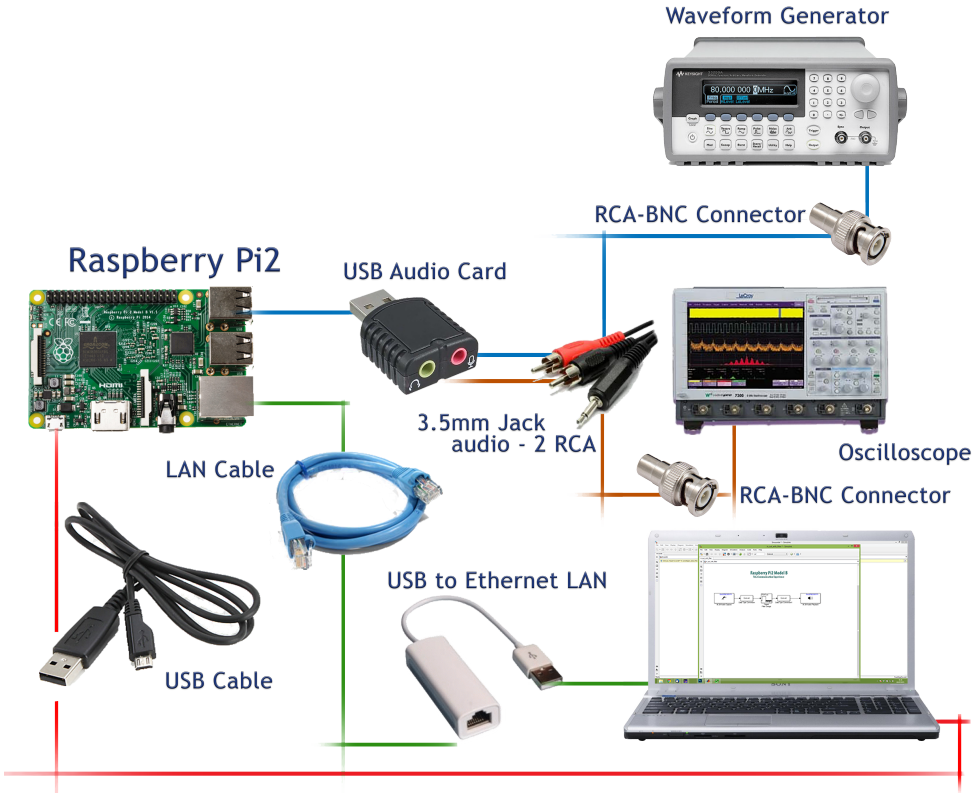


Figure 5.3: Raspberry Pi2 Input/Output connection scheme

block.

The ***Output*** macroblock is the output stage. The filtered signal, represented in the *floating point* format, is first converted by the ***Data Type Conversion1*** block into the *int16* format required by the following stage, and then it is sent to the ***ALSA Audio Playback*** block, representing the analog output port.

Comparing this macroblock with the similar ***Raspberry Pi output*** macroblock used in the previous project (Fig.4.2), you can notice the absence of the ***Matrix Concatenate*** block. In this case, in fact, it is not necessary to duplicate the signal to produce the "stereo" channel required by the ***ALSA Audio Playback*** block. This is possible because the signal produced by the ***ALSA Audio Playback*** block is already in the stereo format.

The ***Control LED*** macroblock, not strictly relevant for the aim of this project, will not be discussed, as it has already been described in Section 4.2.3.

5.3 Elementary blocks used

The list of the elementary blocks used for the project realization is provided hereafter, along with the reference to the sections in which their functioning is described.

- *ALSA Audio Capture* (Section 5.3.1)
- *Digital Filter Design* (Section 5.3.2)
- *Data Type Conversion* (Section 4.3.2)
- *ALSA Audio Playback* (Section 4.3.4)

5.3.1 ALSA Audio Capture

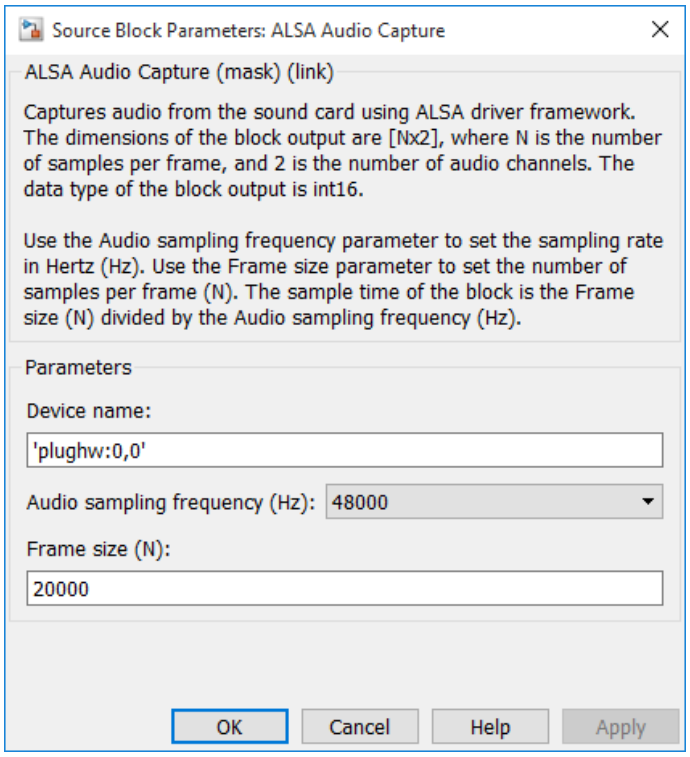


Figure 5.4: ***ALSA Audio Capture*** block

The ***ALSA Audio Capture*** block represents the sound card's ADC converter: It receives the analog signal provided by an external source and convert it into a digital signal. This block is, therefore, the analog input port of the system.

The ***ALSA Audio Capture*** block is controlled by the ALSA driver, that manages all the audio devices connected to the hardware, including the USB sound card described in chapter 4 that hosts the microphone input.

This block generates a signal with dimension [Nx2], where N is the number of samples grouped in each *frame* and 2 is the number of audio channels. Each sample is represented in the *int16* format, that is, as a 15 bits integer+1 sign bit.

The sampling frequency is selected in the *Audio sampling frequency* field of the configuration window shown in Fig.5.4. In this model, and in all the following ones, a sampling rate of 48000 samples/s was used, corresponding to the highest possible value.

The sound card's identifier must be entered in the *Device Name* field, according to the syntax *"plughw:card,device"*. The *card* and *device* parameters can be easily detected with the *arecord -l* command (section 2.1, Listing 2.6). As shown in section 2.1, a list of the input devices connected to the Raspberry Pi2 will be displayed. If the sound card configuration is properly carried out, the *Device Name* will always have the form *'plughw:0,0'*.

It is essential to activate the microphone input using the *alsamixer* command (section 2.1, Fig.2.7).

5.3.2 Digital Filter Design

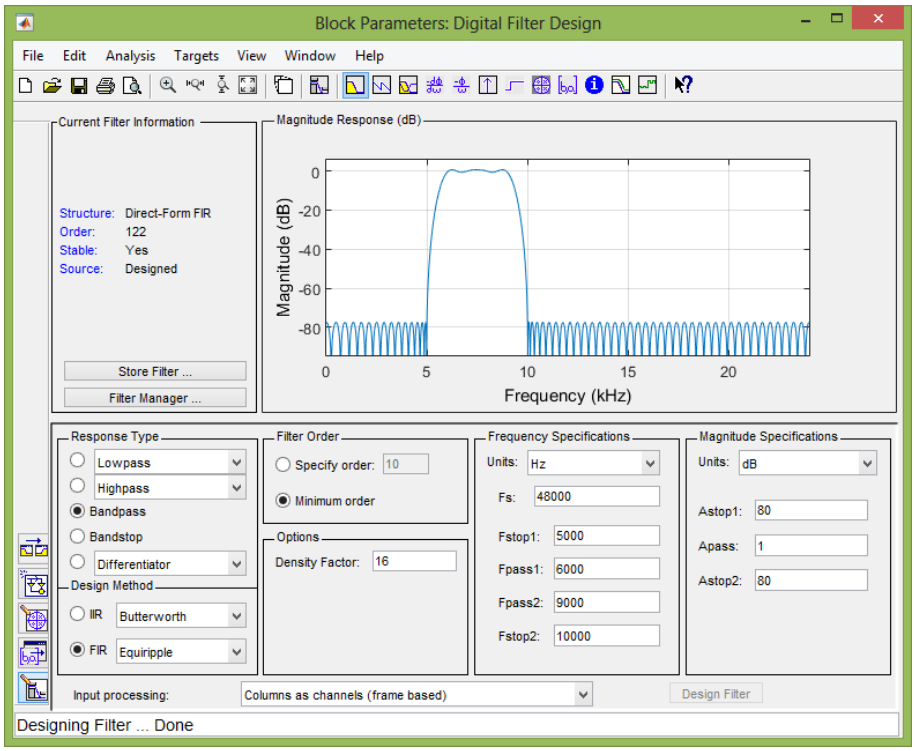


Figure 5.5: Configuration window of the ***Digital Filter Design.*** block

The specifications of the FIR filter are introduced by means of the ***Digital Filter Design - FDATool*** block, whose window (displayed after a double click on the block itself) is shown in Fig.5.5.

We won't describe here this powerful filter design tool and its functioning. We will focus our attention only on the sampling rate Fs, required by the tool as an input parameter.

For the correct functioning of the modelled system, the sampling rate must necessarily be the one used in the ***ALSA Audio Playback*** (par.4.3.4) and ***ALSA Audio Capture*** (par.5.3.1) blocks. In this specific case, then, we set Fs=48000 .

As for the input signal, generated by an external signal generator, it is necessary to remember that, in order to fulfil the conditions set by the sampling theorem with a good safety margin, its frequency range must be comprised in the [0 20 kHz] interval.

5.3.3 Data Type Conversion

The ***Data Type Conversion*** block is described in Section 4.3.2. In the project here considered this block adopts the same configuration shown in Fig.4.8.

5.3.4 ALSA Audio Playback

The ***ALSA Audio Playback*** block represents the DAC converter of system; his functioning is explained in Section 4.3.4. In the project here considered this block adopts the same configuration

shown in Fig.4.10.

5.3.5 Implementation and test of the digital filter

Once the Simulink project has been realized and tested through simulations, it is possible to carry out the *deploy to hardware* procedure, as described in Section 4.4. The project will be automatically executed as soon as the *deploy* procedure is completed.

In order to test the project, a pass-band filter with an attenuation of 80 dB in the intervals 0-5 kHz and 10-24 kHz has been designed. See Fig.5.5 for the corresponding configuration of the ***Digital Filter Design*** block.

Connecting the Raspberry Pi2 input port to the signal generator and the output port to the oscilloscope, as shown in Fig.5.3, it is possible to test the filter behaviour. In particular, changing the frequency of an input sine wave in the [0-20 kHz] interval, the filtering effect can be easily observed, as shown in Fig.5.6 and in Fig.5.7.

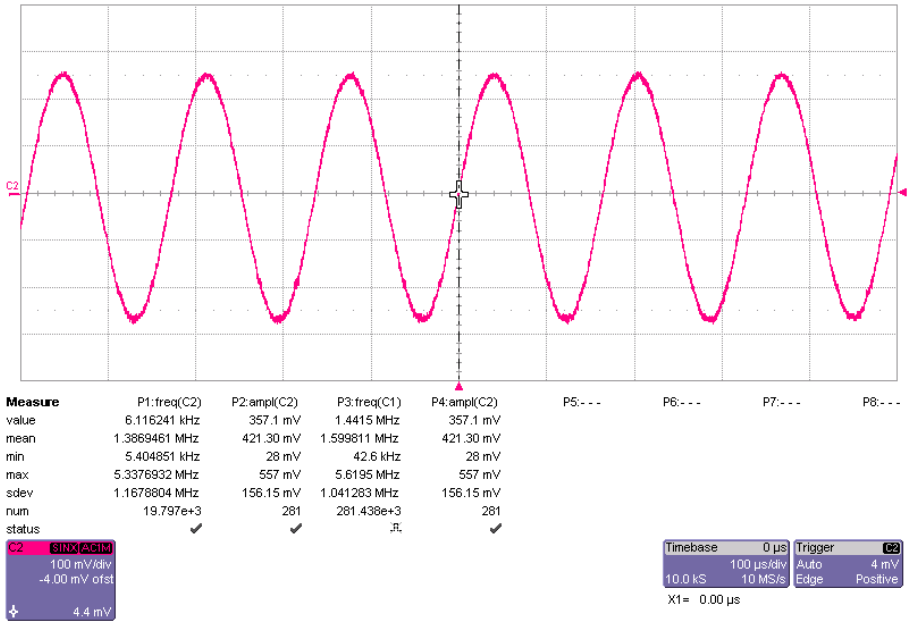


Figure 5.6: Measured filter's output: sine wave at 6.1KHz

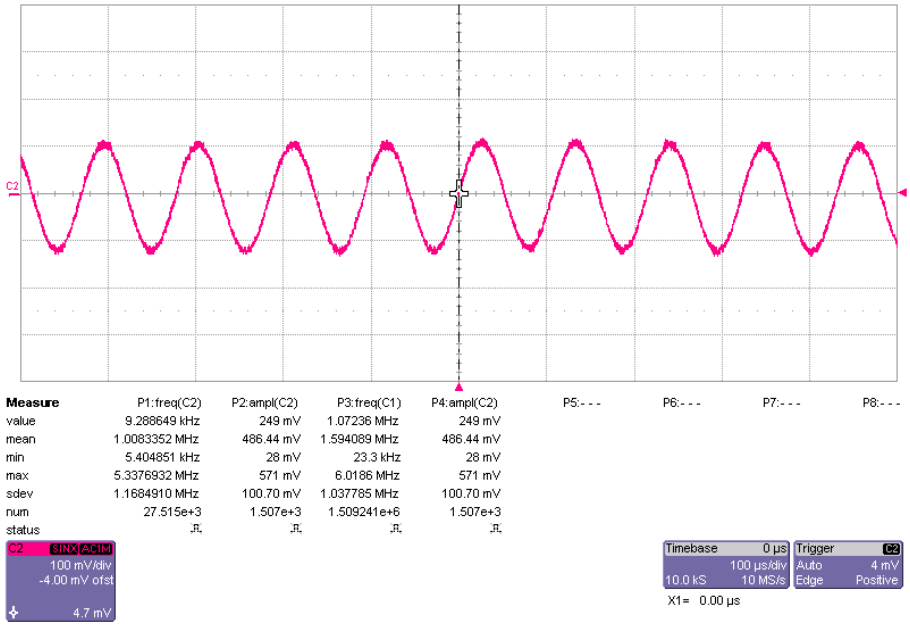


Figure 5.7: Measured filter's output: sine wave at 9.2 KHz

Chapter 6

Baseband modulations with Raspberry Pi2

In this chapter, we introduce two Simulink projects for the hardware implementation of baseband digital transmitters with Pulse Amplitude Modulation - PAM. As it is known, a general M-PAM modulator associates at each symbol a_i a suitable baseband waveform $g(t)$, generating a PAM signal represented by the following equation:

$$s(t) = \sum_{i=-\infty}^{\infty} a_i g(t - iT), \tag{6.1}$$

where

- a_i represents the generic symbol belonging to a M-ary alphabet;
- $g(t)$ is the waveform associated to each symbol;
- T is the time interval between one symbol and the following one.

In order to better understand the projects described below, it is useful to remind that each M-PAM symbol a_i represents a number of bits given by

$$b_{symbol} = \log_2 M. \tag{6.2}$$

The project of a 2-PAM transmitter will be firstly described in the following. In this case, the bits $\{0, 1\}$ to be transmitted are mapped into $M = 2$ possible symbols (for example $a_i \in \{-1, +1\}$).

Such model will evolve later in a 4-PAM transmitter, in which each couple of bits is mapped into $M = 4$ symbols (for example $a_i \in \{-1, -\frac{1}{3}, \frac{1}{3}, +1\}$).

6.1 Equipment required for this experience

To carry out the experimental activity explained below, an oscilloscope is needed, along with the following items:

- *Nr.1 Raspberry Pi2* (Fig.6.1(a))
- *Nr.1 Micro SD memory card* (Fig.6.1(c))
- *Nr.1 Network cable* (Fig.6.1(e))
- *Nr.1 3.5mm-RCA jack cable* (Fig.6.1(g))
- *Nr.1 USB-micro USB cable* (Fig.6.1(b))
- *Nr.1 adattatore USB-LAN* (Fig.6.1(d))
- *Nr.1 Scheda audio esterna* (Fig.6.1(f))
- *Nr.1 adattatore BNC-RCA* (Fig.6.1(h))

(a) Raspberry Pi2 (b) USB-micro USB cable (c) Micro SD memory card (d) USB to LAN adapter

(e) LAN cable (f) External audio card (g) 3.5mm-RCA stereo jack cable (h) RCA-BNC adapter

Figure 6.1: Equipment

6.2 Raspberry Pi2 as 2-PAM transmitter

The Simulink model for implementing a 2-PAM transmitter on a Raspberry Pi2 board is shown in Fig.6.2. Fig.6.3 shows, instead, the interconnections among the different devices constituting the workstation.

In order to make the scheme shown in Fig.6.2 as clear as possible, four macroblocks have been highlighted.

The first macroblock, called ***BaseBand Modulation***, contains all the Simulink blocks needed to implement the 2-PAM modulation. The signal at its output port goes through an

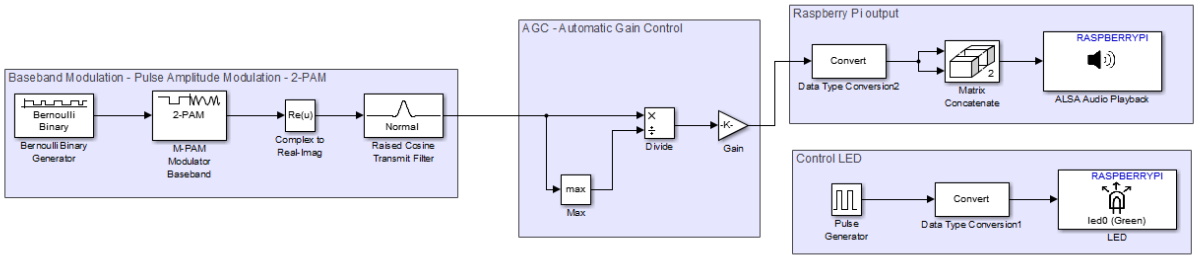


Figure 6.2: Simulink model 2-PAM modulation

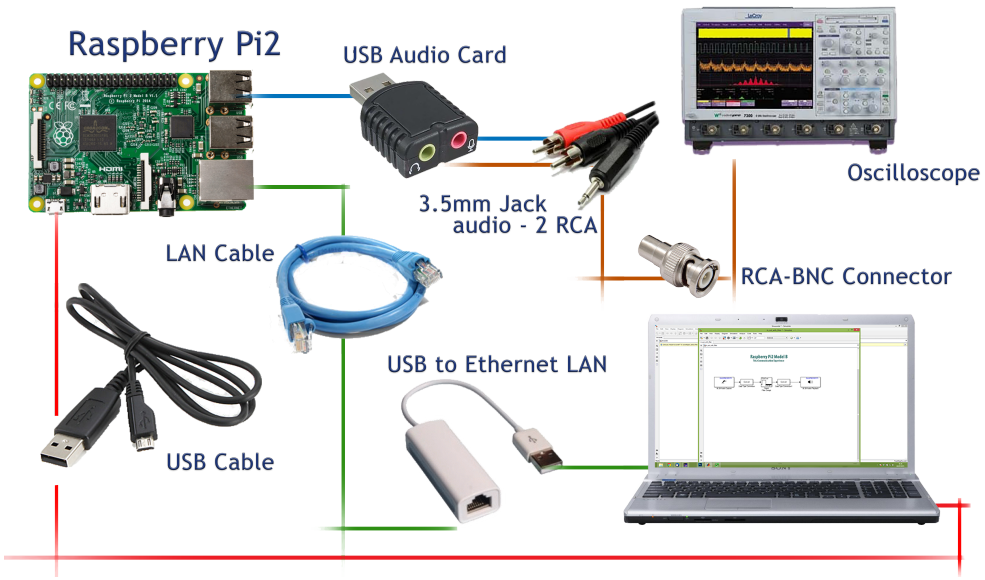


Figure 6.3: Connection scheme

Automatic Gain Control - AGC stage, that adapts the signal's dynamic range to the level required by the following macroblock, called ***Raspberry Pi output*** (see Section 4.2.2), representing the Raspberry Pi2's analog output.

The ***Control LED*** macroblock, discussed in Section 4.2.3, will be no more considered.

6.3 Elementary blocks used

The list of the elementary blocks used for the project realization is provided hereafter, along with the reference to the sections in which their functioning is described.

- *Bernoulli Binary Generator* (Section 6.3.1)
- *M-PAM Modulator Baseband* (Section 6.3.2)
- *Complex to Real-Imag* (Section 6.3.3)
- *Raised Cosine Transmit Filter* (Section 6.3.4)
- *Max-Divide-Gain* (Section 6.3.5)
- *Data Type Conversion* (Section 4.3.2)
- *Matrix Concatenate* (Section 4.3.3)
- *ALSA Audio Playback* (Section 4.3.4)

6.3.1 Bernoulli Binary Generator

The ***Bernoulli Binary Generator*** block is the binary information source. It generates a random sequence of independent bits with Bernoulli statistics: the bit ***0*** is generated with probability *Probability of a zero* and the bit ***1*** with probability *1-Probability of a zero*.

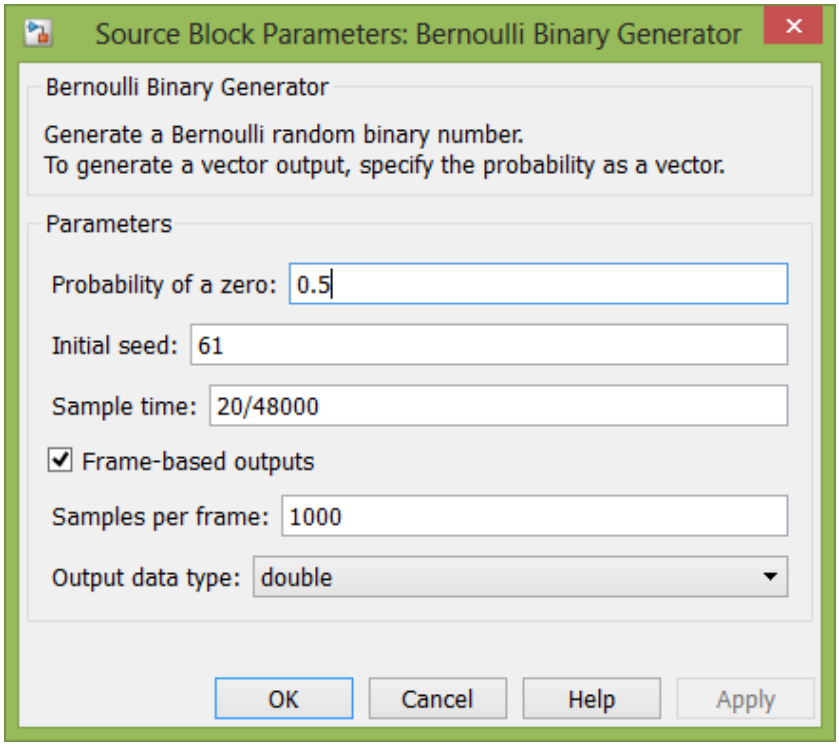


Figure 6.4: Configuration window of the ***Bernoulli Binary Generator*** block

The configuration window of this block (Fig.6.4) asks first of all to specify the *Probability of a zero*, that will be assigned the value 0.5, in order to have a sequence of equiprobable bits.

The *Initial Seed* field concerns the seed used by the random number generator. In our case, the value specified is 61, but the proper functioning of the model is independent of the chosen value.

The *Sample Time* parameter represents the time interval between the generation of a bit and the following one. The reciprocal of such parameter $B_r = \frac{1}{Sample\ Time}$ represents the *bit rate* $[\frac{bit}{s}]$ of the binary generator, that is, the number of bits generated per second.

The value to assign to *Sample Time* is strictly connected to

- the *Audio Sampling Frequency* defined in the ***ALSA Audio Playback*** block (sect.4.3.4),
- the upsampling factor *Output samples per symbol* defined in the ***Raised Cosine Transmit Filter*** (further details about this block will be given in section 6.3.4),
- the number of bits b_{symbol} associated to each modulation symbol (eq.(6.2)).

In particular it must be

$$\frac{B_r}{b_{symbol}} \cdot Output\ samples\ per\ symbol = Audio\ Sampling\ Frequency. \tag{6.3}$$

Recalling that $B_r = \frac{1}{Sample\ Time}$, it is, therefore

$$Sample\ Time = \frac{Output\ samples\ per\ symbol}{b_{symbol} \cdot Audio\ Sampling\ Frequency}. \tag{6.4}$$

Please observe that only specific values of *Sample Time* are allowed. In fact, the *Audio sampling frequency* is dictated by the ***ALSA Audio Playback*** block, b_{symbol} depends on the adopted modulation and *Output Samples per symbol* must be an integer[1].

In this specific case, as *Audio Sampling Frequency*=48000, *Output samples per symbol*=20 and $b_{symbol} = \log_2 M = 1$, it results $Sample\ Time = \frac{20}{48000}$.

The *Sample per frame* field specifies the number of bits grouped[2] in a single *frame*, established in this specific case as 1000. Such choice is, however, not binding.

Finally, the *Output data type* is maintained at the default setting *double*.

6.3.2 M-PAM Modulator Baseband

The ***M-PAM Modulator Baseband*** block converts the input bits sequence into a symbols sequence. In the 2-PAM case here considered, an output symbol a is generated for each input bit b (as dictated by eq.(6.2)), according to the law defined in table 6.1

Table 6.1: 2-PAM modulator. Symbol encoding

b	a
0	-S
1	S

where the value of S depends on the choice made in the block's configuration window for the *Normalization Method* field, that will be discussed later.

The first input parameter to set in the block's configuration window, shown in Fig.6.5, is the *M-ary number* of possible output symbols. In the 2-PAM case it is *M-ary number=2*.

The *Input Type* field relates to the type of input data. The ***Bernoulli Binary Generator*** block, preceding the examined block, generates bits, and so this field must be set accordingly (*bit*).

The *Constellation ordering* field defines the law regulating the correspondence between bits and symbols. In the 2-PAM case such correspondence is the one shown in table 6.1, independently of the choice made (*Gray* o *Binary*).

For the *Normalization Method* field the *Peak Power* option is chosen. The sequence of symbols is thus normalized in terms of peak power, whose value, referenced to 1 Ohm, is defined by the field *Peak power, referenced to 1 Ohm (watts)*. In the project here considered the *Peak power* is 1.

The resulting constellation is shown in Fig.6.6.

According to the previous choices, the output of the ***M-PAM Modulator Baseband*** block is a sequence of +1 e -1. Note that, despite the fact that symbols are purely real quantities, the ***M-PAM Modulator Baseband*** block generates each of them in the complex format, associating to each symbol an imaginary component with null value (e.g., $[..., 1+i0, -1+i0, ...]$).

[1] The *Output Samples per symbol* parameter will be discussed in section 6.3.4.

[2] As observed in section 4.3.1, Raspberry Pi2 boards operate on blocks of data (*frames*) for efficiency reasons.

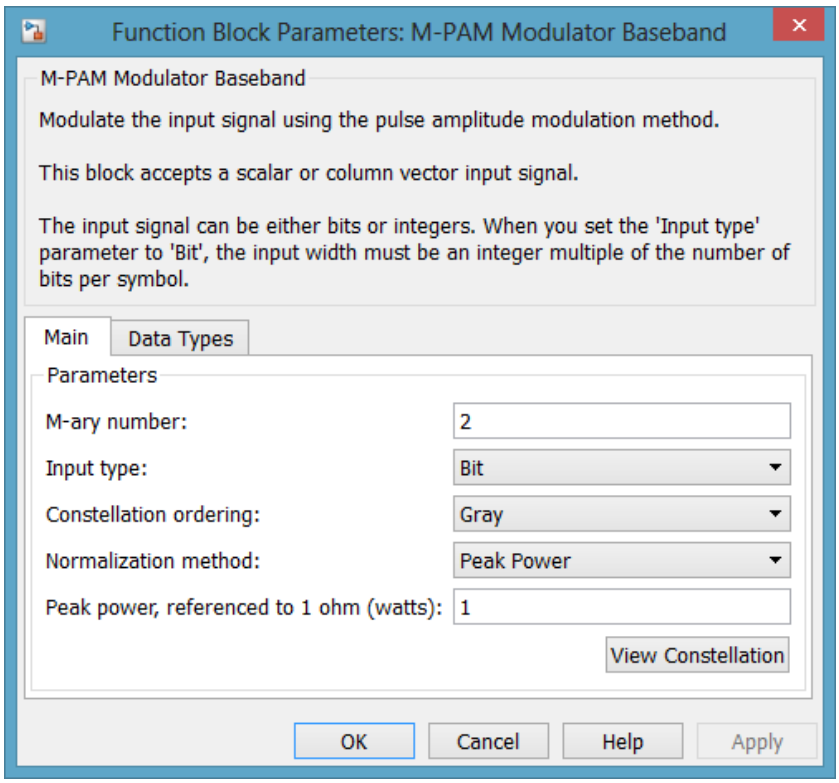


Figure 6.5: Configuration window of the ***M-PAM Modulator Baseband*** block

6.3.3 Complex to Real-Imag

The ***M-PAM Modulator Baseband*** block is followed by the ***Complex to Real-Imag*** block, with the purpose of converting the symbols from the complex into the real format, removing the imaginary (null) components (Fig.6.7). It is a necessary conversion as the Raspberry Pi2's output system requires real data.

6.3.4 Raised Cosine Transmit Filter

The ***Raised cosine transmit filter*** generates the PAM signal, according to eq.(6.1), starting from the symbols at its input. The corresponding configuration window is shown in Fig.6.8.

In particular, this block associates to each symbol generated by the ***M-PAM Modulator Baseband*** block (one symbol each *Sample Time* in the 2-PAM case here considered) the sampled values of the waveform $g(t)$, taken with a sampling interval T_g.

The ratio $\frac{Sample\ Time}{T_g}$ defines the *Output Samples per symbol* parameter, set at 20 in the present project. In general, the value to assign to this parameter depends on:

- the *Audio sampling frequency* and b_{symbol}. In fact, once the *Audio sampling frequency* is defined (dictated by the ***ALSA Audio Playback*** block) and b_{symbol} is known (given the adopted modulation) the *Output Samples per symbol* must be an integer such that eq.(6.4) is satisfied. As observed in section 6.3.1, this poses a condition to the values of *Sample Time*.

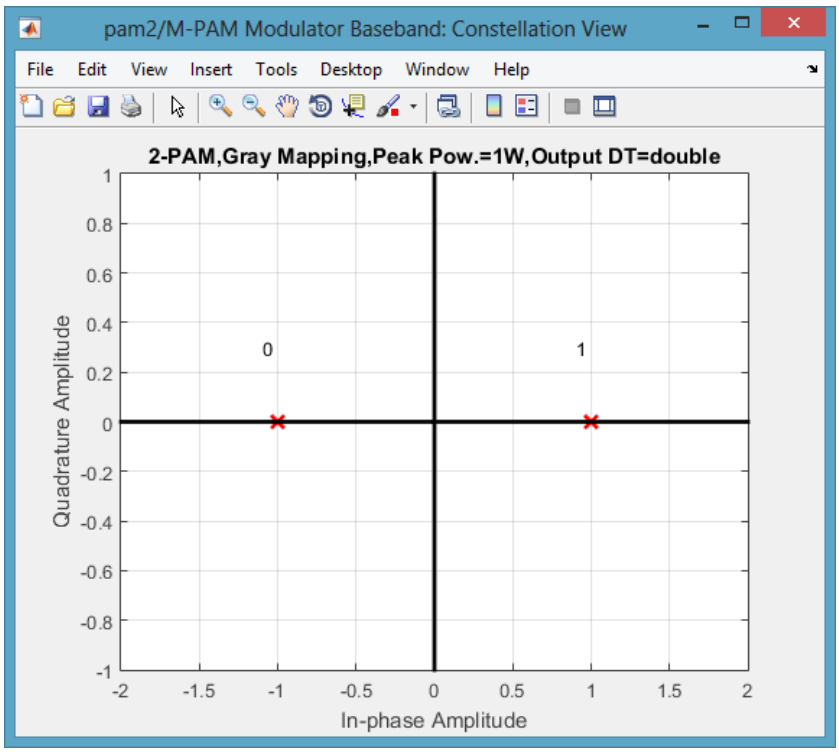


Figure 6.6: 2-PAM Constellation

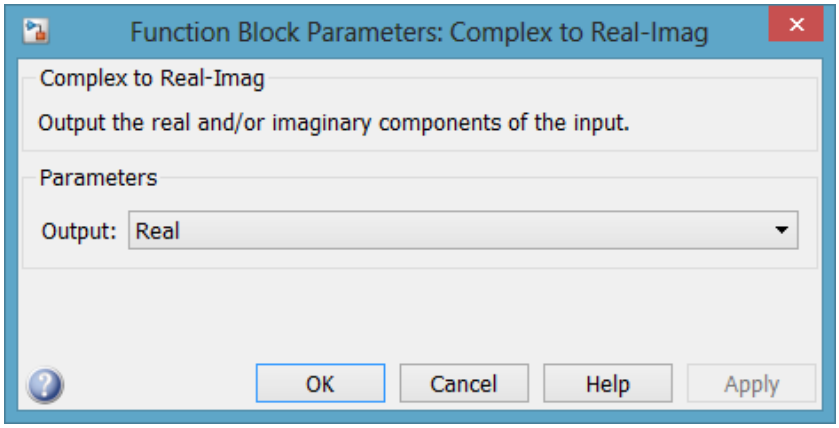


Figure 6.7: Configuration window of the ***Complex to Real-Imag*** block

- the bandwidth B of the PAM signal, which is function[3] of $G(f) = \mathcal{F}\{g(t)\}$: Given B, the minimum value of T_g that fulfils the sampling theorem is such that $\frac{1}{T_g} \geq 2B$. The parameter *Output Samples per symbol*$=\frac{Sample\ Time}{T_g}$ must be, therefore, an integer value such that the condition $\frac{1}{T_g} \geq 2B$ is fulfilled.

Both the above conditions must be fulfilled by the value chosen for the *Output Samples per symbol*

[3]The symbol $\mathcal{F}\{\cdot\}$ represents the continuous-time Fourier transform.

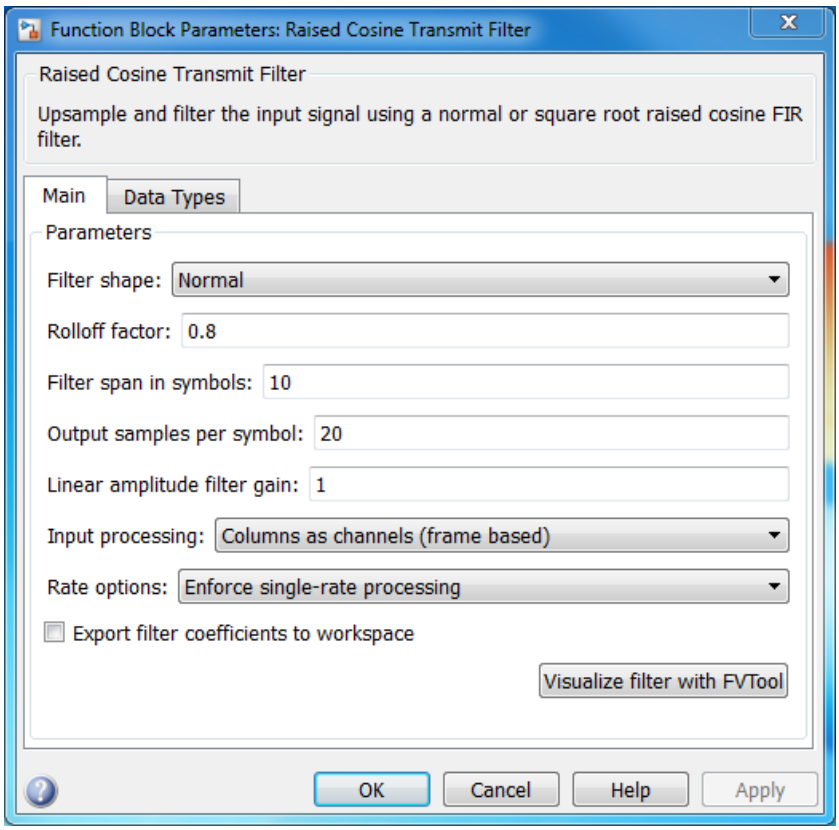


Figure 6.8: Configuration window of the ***Raised cosine transmit filter*** block

parameter.

The duration of the waveform $g(t)$ is defined by the *Filter span in symbols* parameter, set at 10. This value is meant normalized with respect to *Sample Time*.

As a consequence of the previous choices, the number of samples of $g(t)$ that are generated by the ***Raised cosine transmit filter*** for each input symbol is

$$\text{Output Samples per symbol} \cdot \text{Filter span in symbols} = 20 \cdot 10 = 200.$$

This sequence represents the impulse response of the raised-cosine filter and therefore, as it is a FIR filter, it is the sequence of its coefficients.

The particular waveform $g(t)$ is chosen setting the *Filter Shape* field. The possible choices are *Normal*, corresponding to the raised cosine waveform, and *Square root*, referring to the square root raised cosine waveform. In this project the *Filter Shape* field is set at *Normal*.

The *Rolloff factor* parameter represents the filter's roll-off factor. Its value, that affects the signal bandwidth B, can be freely chosen within the interval $[0, 1]$.

The *Linear amplitude filter Gain* parameter represents the filter gain, and is set at its default value 1.

Consistently with the previous blocks, also the ***Raised Cosine Transmit Filter*** works in *frame based* mode and the *Input processing* field has been consequently set.

The *Rate options* field, set at *Enforce Single Rate Processing*, forces the block to adapt the

output frame length so that the input and output rates are the same, independently of the upsampling operation performed by the filter itself.

6.3.5 Max-Divide-Gain (Automatic Gain Control)

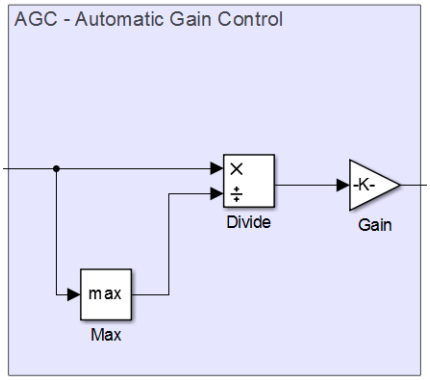


Figure 6.9: Automatic Gain Control

The ***Automatic Gain Control*** macroblock adapts the signal's dynamic range to the requirements of the Raspberry Pi2™'s output port. The macroblock structure, shown in Fig.6.9, is based on the ***Max, Divide*** and ***Gain*** elementary blocks.

For each input frame, in particular, the ***Automatic Gain Control*** macroblock performs, first of all, a normalization, dividing all the samples of each frame by their maximum value, then carries out a multiplication by $K = 2^{14} - 1$. The configuration windows of the three blocks are shown in Fig.6.10(a), Fig.6.10(b) and Fig.6.11, respectively.

As a consequence of this operation, the highest value in each frame is equal to $2^{14} - 1$, consistent with the highest value required by the Raspberry Pi2™'s output port, equal to $2^{15} - 1$.

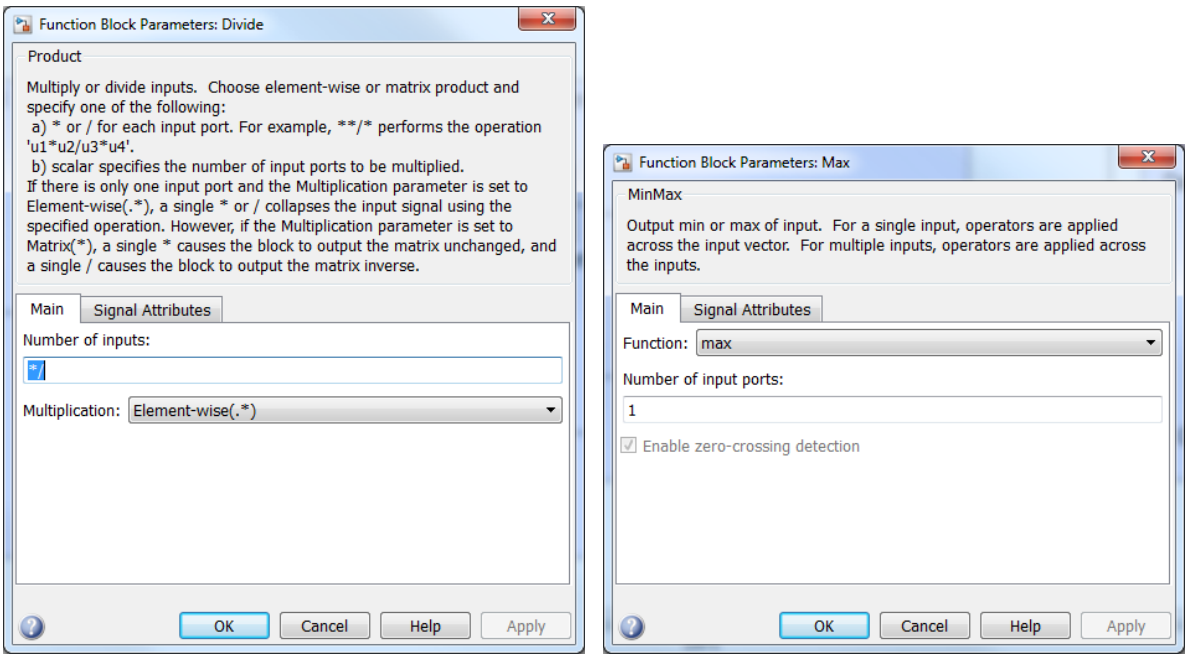
In principle, the gain could be set at $2^{15} - 1$, but some malfunctioning was observed with such configuration. Such problems were solved setting the dynamic range to $2^{14} - 1$.

6.3.6 Implementation and test of the 2-PAM transmitter

Once the Simulink project has been realized and tested through simulations, it is possible to carry out the *Deploy to Hardware* procedure, as described in Section 4.4. The project will be automatically executed as soon as the *deploy* procedure is completed.

Connecting the Raspberry Pi2's output to the oscilloscope, following the scheme in Fig.6.3, it is possible to observe the typical shape of a 2-PAM signal with raised cosine filtering, as shown in Fig.6.12.

The eye diagram shown in Fig.6.13 shows, however, an unexpected degradation, that deserves specific investigations. Observing a larger time interval (Fig.6.14) unexpected fluctuation are detected, most likely caused by the low quality of the digital-to-analog conversion.



(a) Divide (b) Max

Figure 6.10: ***Max*** and ***Divide*** blocks

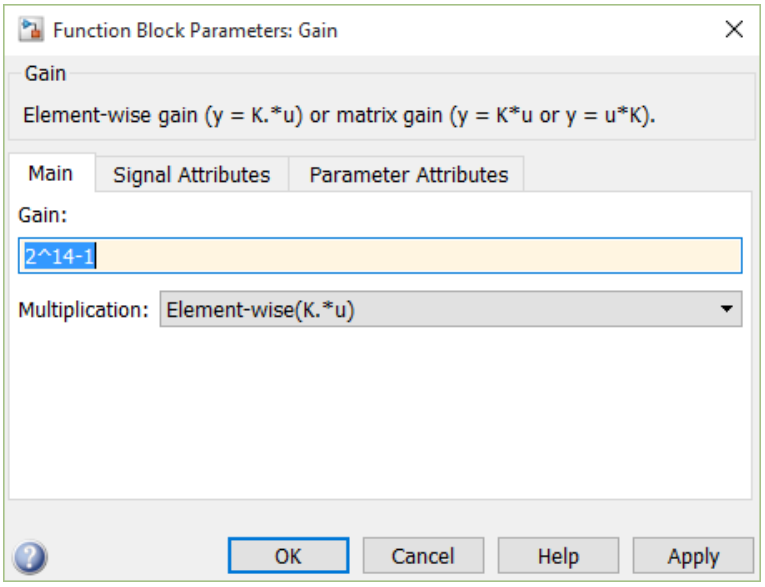


Figure 6.11: ***Gain*** block

6.4 Raspberry Pi2 as a 4-PAM transmitter

In the following section we will describe the Simulink model for the hardware implementation of a 4-PAM transmitter. In this case a modulation symbol a_i is generated for each couple of input bits (for example $a_i \in \{-1, -\frac{1}{3}, \frac{1}{3}, +1\}$).

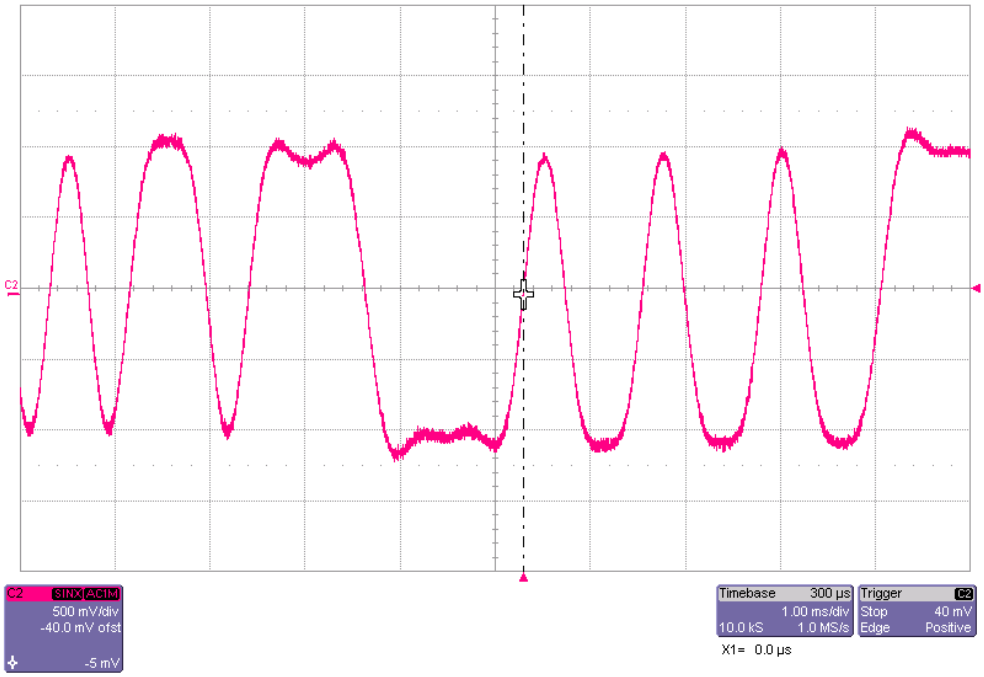


Figure 6.12: Measured 2PAM signal

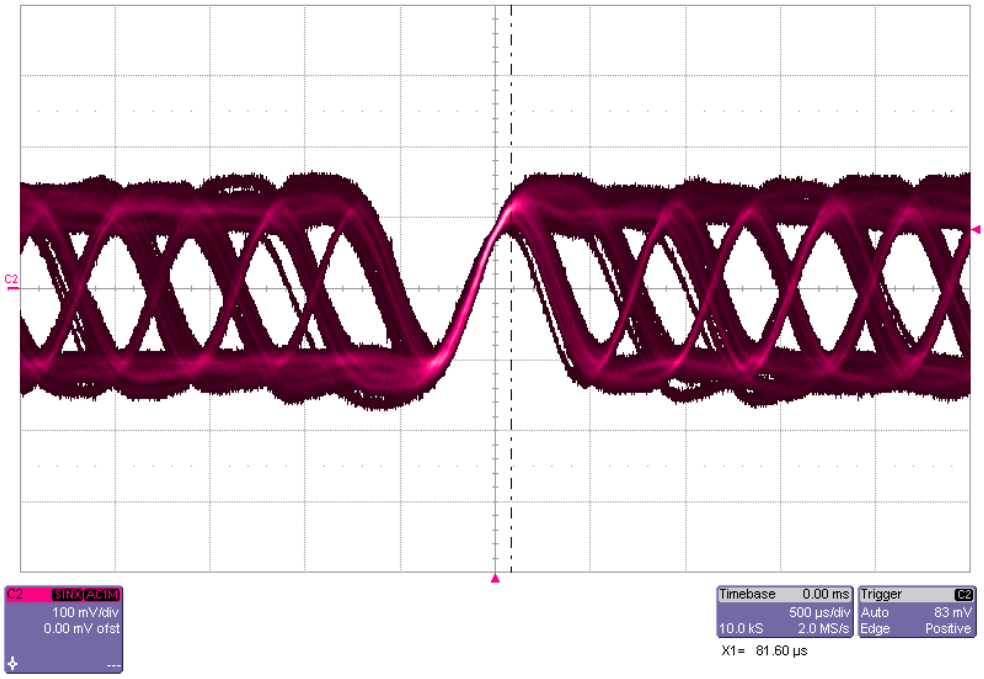


Figure 6.13: Measured 2-PAM eye diagram

The model of the 4-PAM transmitter, shown in Fig.6.15, shows minimal variations compared to that of the 2-PAM transmitter (Fig.6.2). The changes concern only the ***Bernoulli Binary***

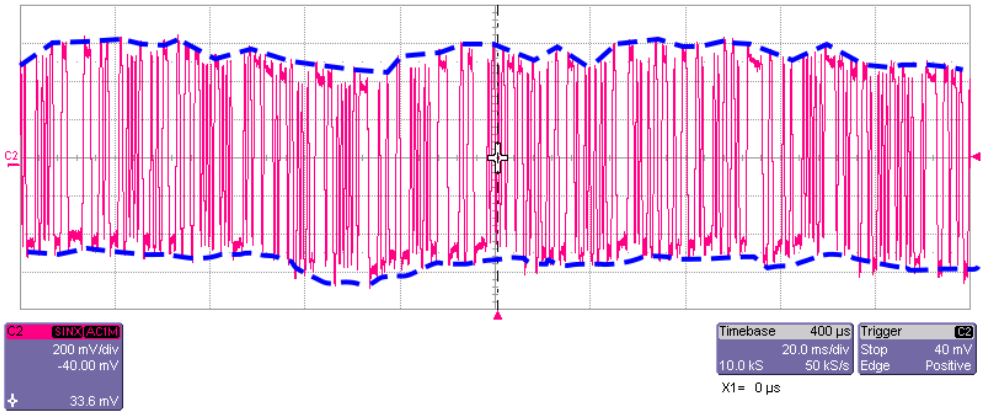


Figure 6.14: ADC/DAC tension levels fluctuations

Generator and ***M-PAM Modulator*** blocks. For this reason, in the following section we will discuss only the changes concerning such blocks, whereas the configurations of the remaining blocks are unchanged with respect to the 2-PAM transmitter.

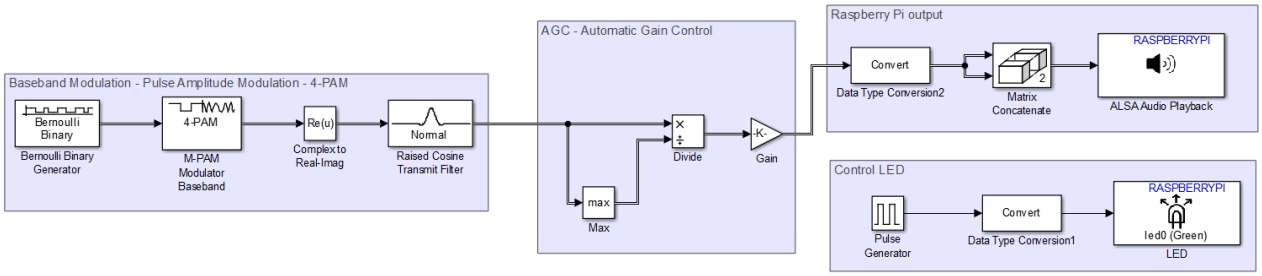


Figure 6.15: Simulink model 4-PAM modulator

6.5 Elementary blocks used

The list of the elementary blocks used for the project realization is provided hereafter, along with the reference to the sections in which their functioning is described.

- *Bernoulli Binary Generator* (Section 6.5.1)
- *M-PAM Modulator Baseband* (Section 6.5.2)
- *Complex to Real-Imag* (Section 6.3.3)
- *Raised Cosine Transmit Filter* (Section 6.3.4)
- *Max-Divide-Gain* (Section 6.3.5)
- *Data Type Conversion* (Section 4.3.2)
- *Matrix Concatenate* (Section 4.3.3)
- *ALSA Audio Playback* (Section 4.3.4)

6.5.1 Bernoulli Binary Generator

As it is known, the ***Bernoulli Binary Generator*** represents the binary information source. It generates a random sequence of independent bits with Bernoulli statistics. Comparing its configuration window, shown in Fig.6.16, with that of the same block for the 2-PAM modulator,

shown in Fig.6.4, the only difference is the *Sample Time* parameter, that represents the time interval between the generation of a bit and the following one.

As anticipated in section 6.3.1, the value to assign to this parameter is strictly related to the *Audio Sampling Frequency*, defined in the ***ALSA Audio Playback*** block (sect.4.3.4), to the number of *Output samples per symbol*, defined in the ***Raised Cosine Transmit Filter*** (sect.6.3.4), and to the number of bits b_{symbol} associated to each symbol of the chosen modulation (eq.(6.2)). We already pointed out, in particular, that for a generic M-PAM modulation it is:

$$Sample\ Time = \frac{Output\ samples\ per\ symbol}{b_{symbol} \cdot Audio\ Sampling\ Frequency}.$$

In this specific case, as *Audio Sampling Frequency*=48000, *Output samples per symbol*=20 and $b_{symbol} = \log_2 M = 2$, it is $Sample\ Time = \frac{20}{2 \cdot 48000}$, as shown in Fig.6.16.

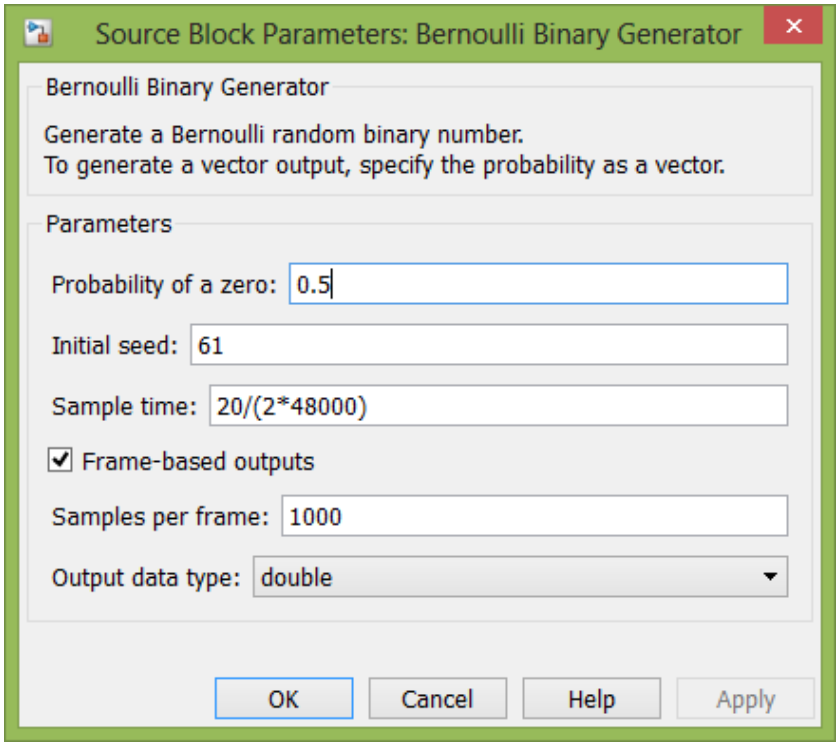


Figure 6.16: Configuration window of the ***Bernoulli Binary Generator*** block

All the other parameters are the same adopted for the 2-PAM transmitter (sect.6.3.1).

6.5.2 M-PAM Modulator Baseband

As explained in section 6.3.2, the ***M-PAM Modulator Baseband*** block converts the sequence of input bits into a sequence of symbols. In the 4-PAM case, in particular, each pair *bb* of input bits corresponds a symbol *a* (as it results from equation (6.2)), according to a correspondence law which is, in principle, arbitrary. The correspondence adopted in this project is shown in table 6.2, where the values of S_1 and S_2 depend on the choice made in the block's configuration window for the *Normalization Method* field.

Table 6.2: 4-PAM modulator.Symbol encoding

bb	*a*
00	-S_2
01	-S_1
11	S_1
10	S_2

The only difference between the configuration chosen for the 2-PAM modulator, shown in Fig.6.5, and the 4-PAM modulator relates to the *M-ary number* parameter, that must be set at 4, consistently with the 4-PAM modulation here considered. The remaining parameters are unchanged.

The configuration window resulting in the present case is shown in Fig.6.17.

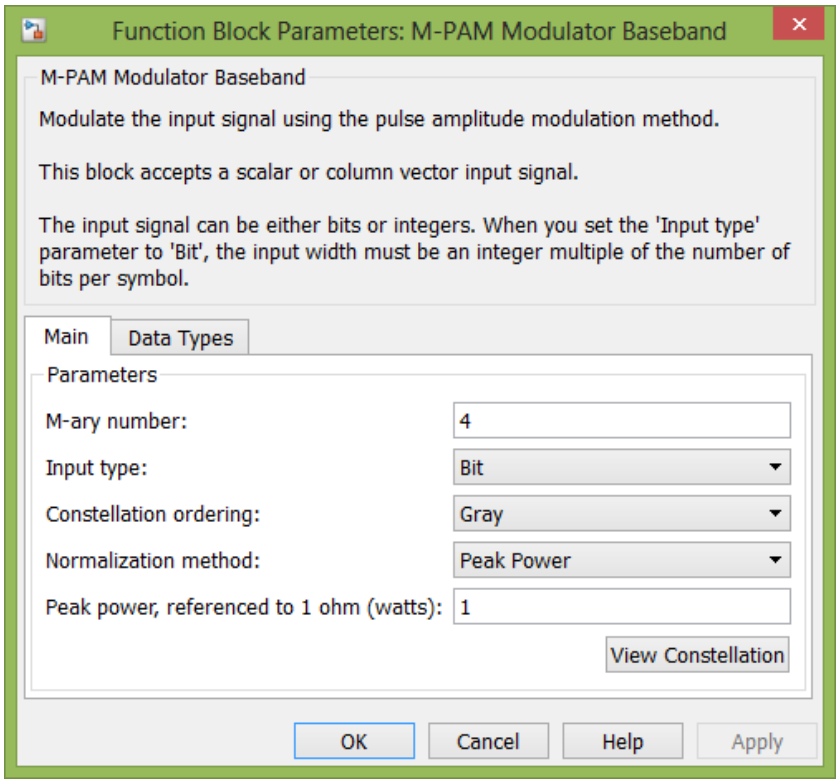


Figure 6.17: Configuration window of the ***M-PAM Modulator Baseband*** block

With such choices, the output of the ***M-PAM Modulator Baseband*** block is a sequence of symbols $-S_2 = -1$, $-S_1 = -\frac{1}{3}$, $S_1 = +\frac{1}{3}$, $S_2 = +1$. The corresponding constellation is shown in Fig.6.18.

6.5.3 Complex to Real-Imag

The ***Complex to Real-Imag*** block is described in Section 6.3.3. In the project here considered this block adopts the same configuration shown in Fig.6.7.

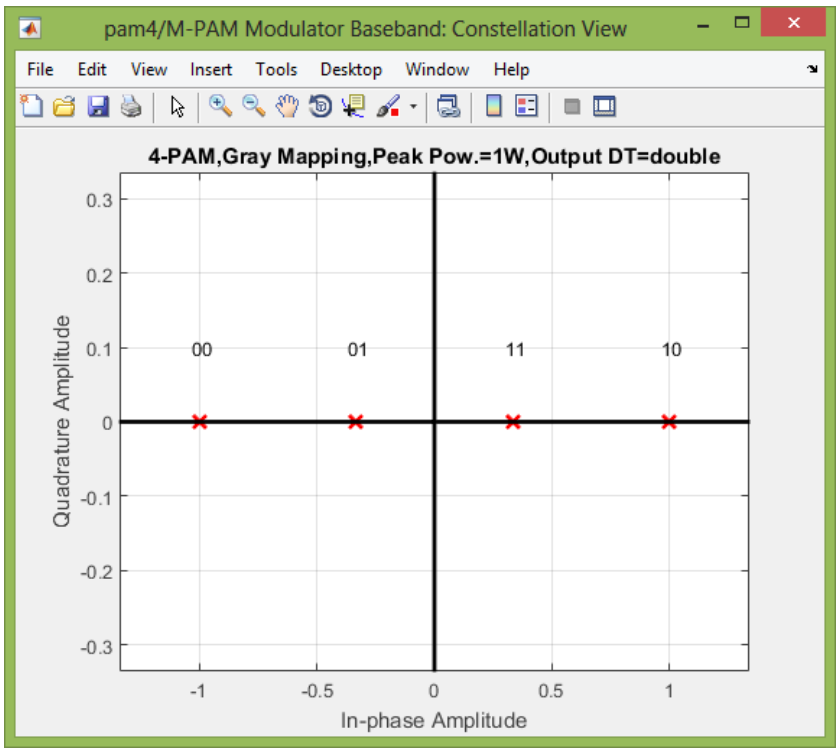


Figure 6.18: 4-PAM constellation

6.5.4 Raised Cosine Transmit Filter

The ***Raised Cosine Transmit Filter*** is described in Section 6.3.4. In the project here considered this block adopts the same configuration shown in Fig.6.8.

6.5.5 Max-Divide-Gain (Automatic Gain Control)

The ***Max-Divide-Gain*** blocks, constituting the automatic gain control macroblock, are described in Section 6.3.5. In the project here considered these blocks adopt the same configuration shown in Fig.6.10 and in Fig.6.11.

6.5.6 Data Type Conversion

The ***Data Type Conversion*** block is described in Section 4.3.2. In the project here considered this block adopts the same configuration shown in Fig.4.8.

6.5.7 Matrix Concatenate

The ***Matrix Concatenate*** block is described in Section 4.3.3. In the project here considered this block adopts the same configuration shown in Fig.4.9.

6.5.8 ALSA Audio Playback

The ***ALSA Audio Playback*** block is described in Section 4.3.4. In the project here considered this block adopts the same configuration shown in Fig.4.10.

6.5.9 Implementation and test of the 4-PAM transmitter

Once the Simulink project has been realized and tested through simulations, it is possible to carry out the *Deploy to Hardware* procedure, as described in Section 4.4. The project will be automatically executed as soon as the *deploy* procedure is completed.

Connecting the Raspberry Pi2's output to the oscilloscope, following the scheme in Fig.6.3, it is possible to observe the typical shape of a 4-PAM signal with raised cosine pulses, as shown in Fig.6.19

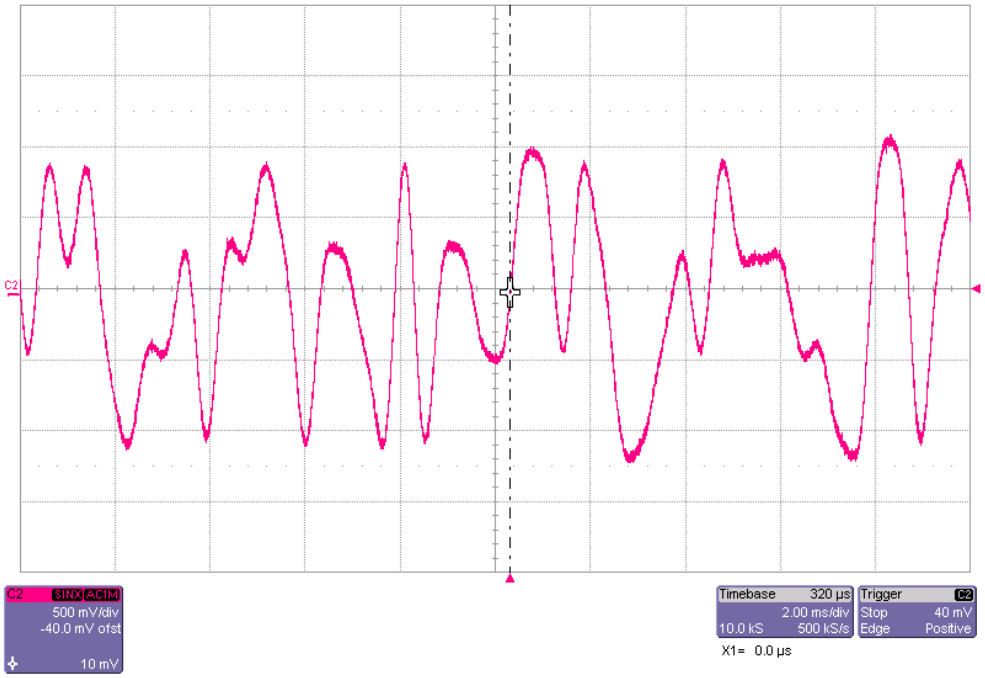


Figure 6.19: 4-PAM signal

6.6 PAM modulations with square pulses

The 2-PAM and 4-PAM transmitters previously described adopt a raised cosine shaping filter, realized through the ***Raised Cosine Transmit Filter***.

For didactic purposes it could be useful to consider also 2-PAM and 4-PAM transmitters with square pulses. The correspondent models are shown in Fig.6.20 and in Fig.6.21.

Comparing the new models with the previous ones, shown in Fig.6.2 and 6.15, it is immediately evident that the only difference is in the ***BaseBand Modulation*** macroblock, described in the following section.

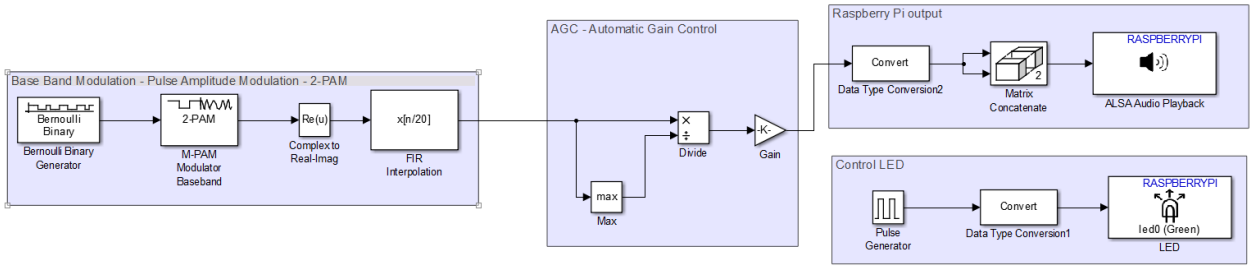


Figure 6.20: 2-PAM transmitter with square pulses

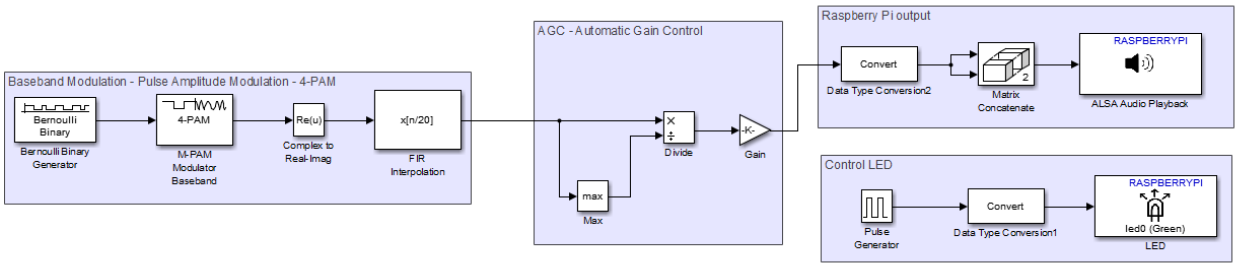


Figure 6.21: 4-PAM transmitter with square pulses

6.6.1 BaseBand Modulation

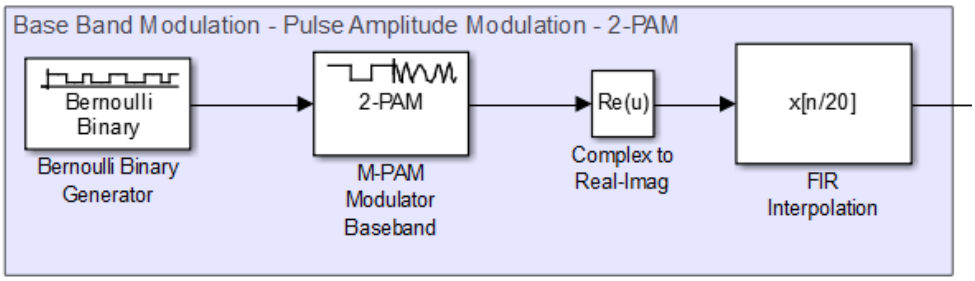
In order to realize PAM transmitters with square pulses, starting from the previously discussed PAM transmitters with raised cosine pulses (see Fig.6.2 and Fig.6.15), it is enough to replace the ***Raised Cosine Transmit Filter*** with a ***FIR Interpolation*** block that, properly configured, works as a square pulse shaping filter.

The new ***BaseBand Modulation*** macroblocks for the 2-PAM and 4-PAM transmitters with square pulses are shown in Fig.6.22.

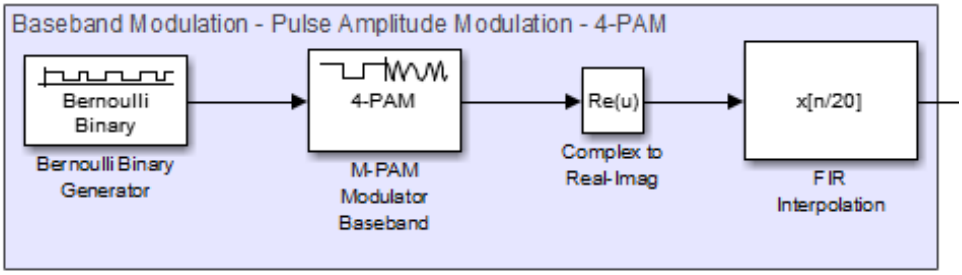
The configuration window of the ***FIR Interpolation*** block is shown in Fig.6.23. As it plays the role of shaping filter, this block must necessarily perform an interpolation (for each input symbol, it must generate the sampled sequence of the corresponding waveform $g(t)$). Also in this case an upsamplig factor equal to 20 is adopted, hence we set *Interpolation factor*=20. This parameter is the correspondent, for the ***FIR Interpolation*** block, of the *Output samples per symbol* parameter requested by the ***Raised Cosine Transmit Filter*** (see Fig.6.8).

The *FIR filter coefficients* field requires to enter the filter's coefficients, that corresponds to the samples of the desired waveform $g(t)$. It follows that, in order to generate a square pulse with duration equal to the symbol time, it is enough to enter a sequence of 1s as long as the interpolation factor (equal to 20, in our case). For this purpose, the command *ones (1,20)* can be used.

The remaining parameters of the configuration window keep the default settings.



(a) 2-PAM transmitter with square pulse shaping filter.



(b) 4-PAM transmitter with square pulse shaping filter.

Figure 6.22: Square pulse PAM modulations

6.6.2 2-PAM and 4-PAM transmitters with square pulses implementation and check

Once the Simulink project has been realized and tested through simulations, it is possible to carry out the *Deploy to Hardware* procedure, as described in Section 4.4. The project will be automatically executed as soon as the *deploy* procedure is completed.

Connecting the Raspberry Pi2's output to the oscilloscope, following the scheme in Fig.6.3, the signals can be finally observed.

Fig.6.24 and 6.25 show, in particular, the 2-PAM and 4-PAM signals with square pulses. The overshoots that can be observed are due to the filtering carried out by the sound card, that removes relevant spectral components of the signal spectrum.

The corresponding eye diagrams, shown in Fig.6.26 and 6.27 reflect such overshoots and the undesired fluctuations already observed in Section 6.3.6.

In Fig.6.28 an example of spectrum measurement is also reported.

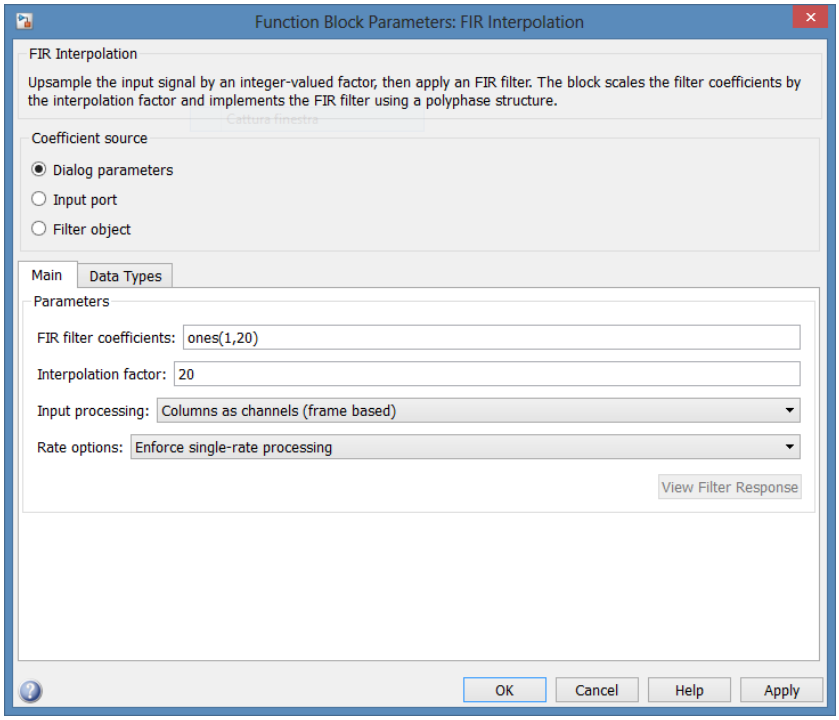


Figure 6.23: Configuration window of the ***FIR Interpolation*** block

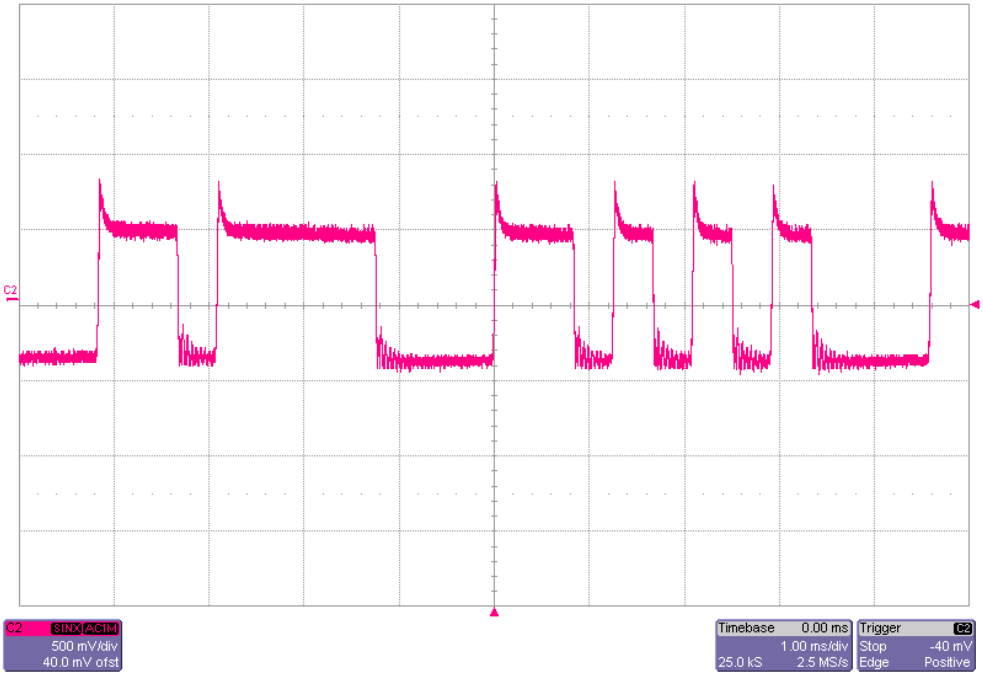


Figure 6.24: 2-PAM modulation with square pulses

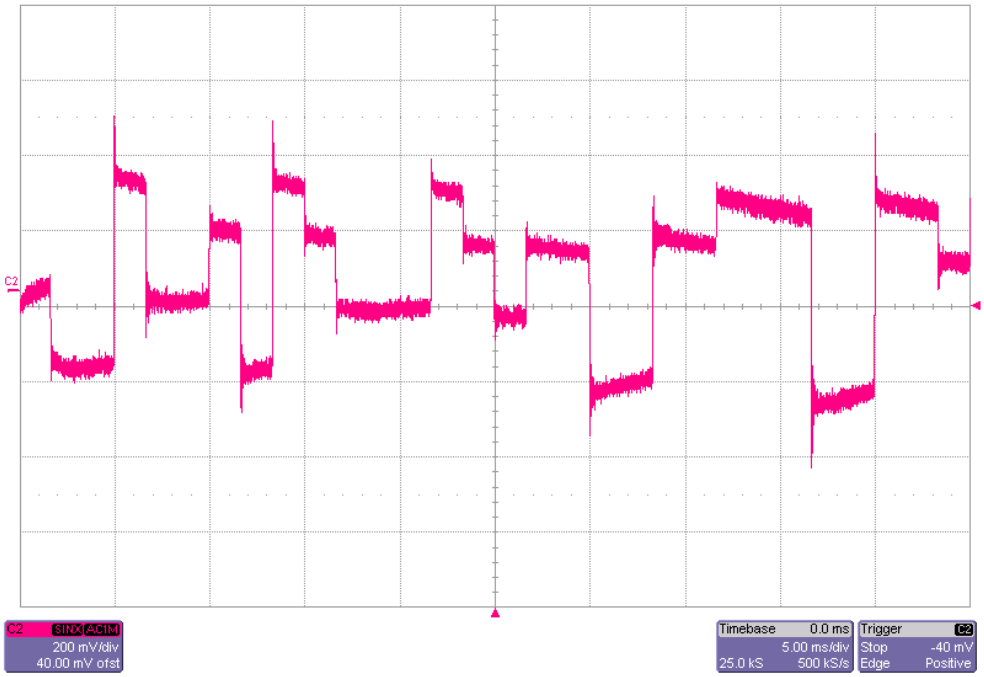


Figure 6.25: 4-PAM modulation with square pulses

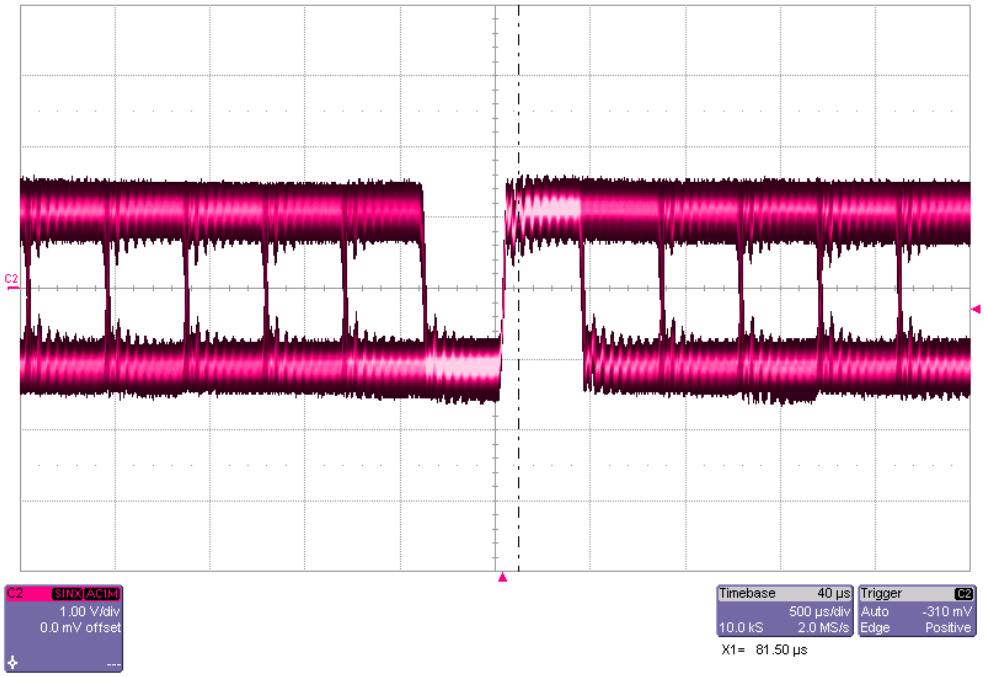


Figure 6.26: 2-PAM modulation. Eye diagram

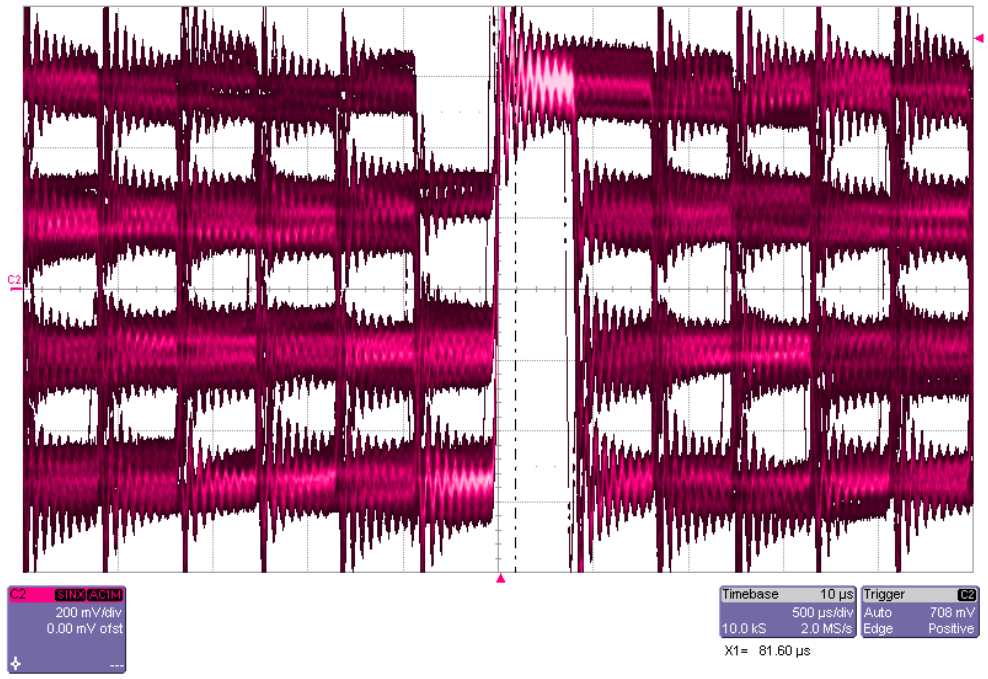


Figure 6.27: 4-PAM modulation. Eye diagram

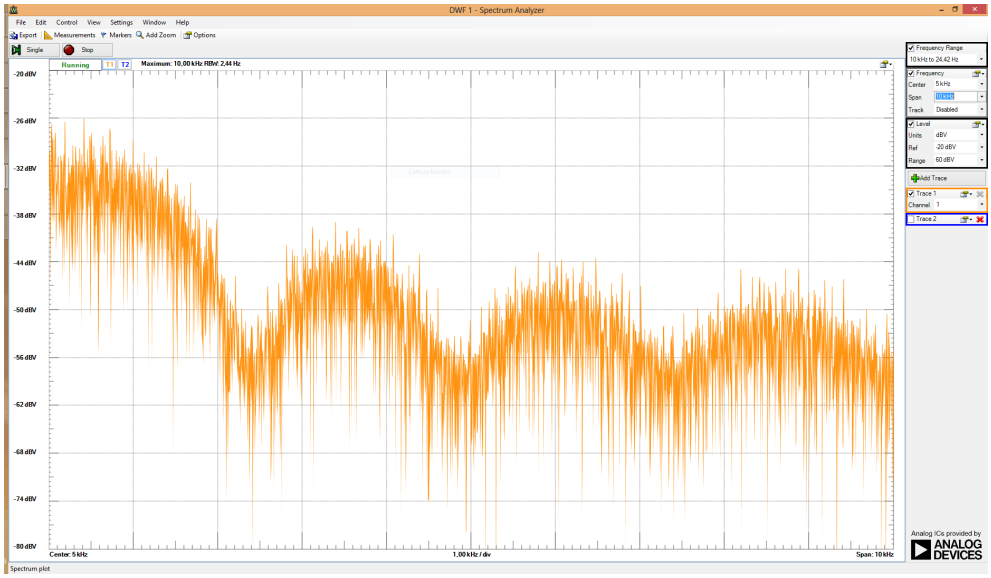
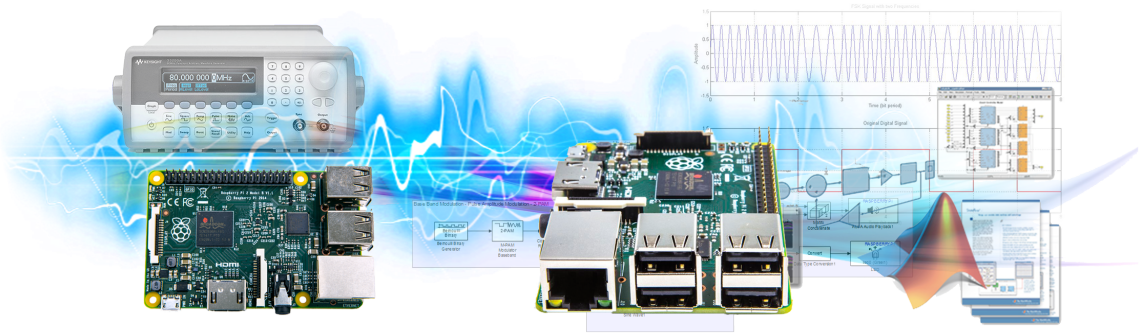


Figure 6.28: Spectrum of the 4-PAM signal with square pulses

Chapter 7

2-ASK and 4-ASK modulations with Raspberry Pi2



The ASK (Amplitide Shift Keying) modulation represents digital data as variations in the amplitude of a sine wave. L-ASK signals, with L denoting the possible amplitude values, can be generated by product modulation, that is, by multiplying the L-PAM signal carrying the data with a sine wave, usually denoted as *carrier*. The general expression of an L-ASK modulated signal is given by:

$$s(t) = V_0 \sum_{i=-\infty}^{\infty} a_i g(t - iT) \cos(2\pi f_0 t), \tag{7.1}$$

where

- V_0 represents the carrier amplitude;
- a_i represents the generic symbol;
- $g(t)$ is the waveform associated to each symbol;
- T is the time interval between a symbol and the following one;
- f_0 is the carrier frequency.

In this chapter we describe the Simulink models for the hardware implementation of 2-ASK and 4-ASK transmitters.

In particular, we will firstly describe the project of the 2-ASK transmitter, in which each input bit is mapped into a modulation symbol (for example $a_i \in \{-1, +1\}$), then we will introduce the 4-ASK transmitter, in which each couple of input bits is mapped over $L = 4$ symbols (for example $a_i \in \{-1, -\frac{1}{3}, \frac{1}{3}, +1\}$).

Each project requires the carrier to be generated externally, through a signal generator. The constraints imposed by the limited band ([0 20 kHz]) of the adopted DAC (see Chapter 2) obviously influence the choice of the carrier frequency and the signal bandwidth: the first is of the order of 15 kHz, the second of few kHz.

7.1 Equipment required for this experience

The experimental activity described in this chapter requires a signal generator, an oscilloscope and the following equipment:

- *Nr.1 Raspberry Pi2* (Fig.7.1(a))
- *Nr.1 Micro SD memory card* (Fig.7.1(c))
- *Nr.1 Network cable* (Fig.7.1(e))
- *Nr.1 3.5mm-RCA jack cable* (Fig.7.1(g))
- *Nr.1 USB-micro USB cable* (Fig.7.1(b))
- *Nr.1 USB-LAN adapter* (Fig.7.1(d))
- *Nr.1 External audio card* (Fig.7.1(f))
- *Nr.2 BNC-RCA adapter* (Fig.7.1(h))

(a) Raspberry Pi2 (b) USB-micro USB cable (c) Micro SD memory card (d) USB to LAN adapter

(e) LAN cable (f) External audio card (g) 3.5mm-RCA stereo jack cable (h) RCA-BNC adapter

Figure 7.1: Equipment

7.2 2-ASK transmitter

The Simulink model for implementing a 2-ASK transmitter on a Raspberry Pi2 board is shown in Fig.7.2. Fig.7.3 shows, instead, the interconnections among the different devices constituting the workstation.

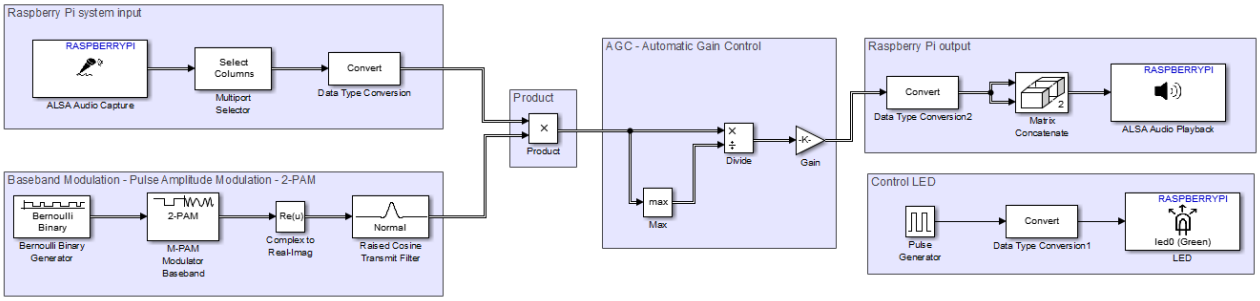


Figure 7.2: 2-ASK transmitter: Simulink model

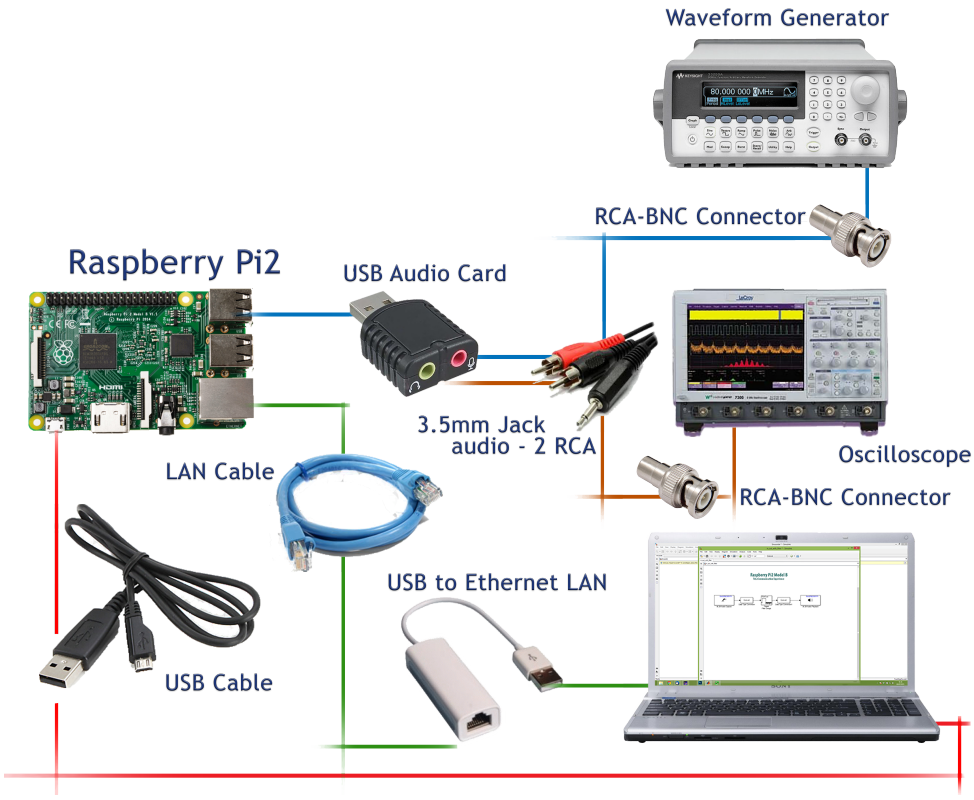


Figure 7.3: Raspberry Pi2's Input/Output connection scheme

In order to make the scheme shown in Fig.7.2 as clear as possible, six macroblocks have been highlighted: ***Baseband Modulation***, ***Raspberry Pi system input***, ***Product***, ***AGC-***

Automatic Gain Control, ***Raspberry Pi output*** and ***Control LED***[1].

7.2.1 Baseband Modulation

The ***Baseband Modulation*** macroblock is the 2-PAM modulator described in Section 6.2. Its elementary blocks ***Bernoulli Binary Generator***, ***M-PAM Modulator Baseband***, ***Complex to Real-Imag*** and ***Raised Cosine Transmit Filter*** have been described in Sections 6.3.1, 6.3.2, 6.3.3 and 6.3.4 together with the corresponding configurations, that are unchanged in this project.

The 2-PAM signal generated by this macroblock modulates the carrier, which is the output of the ***Raspberry Pi system input*** macroblock.

7.2.2 Raspberry Pi system input

The ***Raspberry Pi system input*** macroblock receives a sinusoid (the carrier), generated by an external signal generator, and adapts it to the format required by the subsequent block. This macroblock, in particular, converts the stereo signal, at the ***ALSA Audio Capture***'s output, into a mono signal, through the ***Multiport Selector*** block, and the *int16* format into a *double* format, by means of the ***Data Type Conversion*** block.

7.2.3 Product

The ***Product*** block simply performs the multiplication between the carrier, outputting from the ***Raspberry Pi system input*** block, and the modulating signal, outputting from the ***Baseband Modulation*** block. The ***Product*** block acts, in other words, as a *mixer*, performing a product modulation.

7.2.4 Max-Divide-Gain (Automatic Gain Control)

The ***Max-Divide-Gain*** blocks, performing the gain automatic gain control, are described in Section 6.3.5. The project here considered adopts the same configuration shown in Fig.6.10 and in Fig.6.11.

7.2.5 Raspberry Pi output

The **Raspberry Pi output** macroblock represents the Raspberry Pi's output port. Its elementary blocks ***Data Type Conversion***, ***Matrix Concatenate*** and ***ALSA Audio Playback*** have been described in Sections 4.3.2, 4.3.3 and 4.3.4.

7.2.6 Control LED

The only aim of the ***Control LED*** macroblock is to intermittently turn on and off the Raspberry Pi2's led during the execution of the project, visually confirming the that the project is running. It has been discussed in Section 4.2.3.

[1]The ***Control LED*** macroblock has been discussed in Section 4.2.3. In the following, therefore, the details of its functioning will not be provided any more.

7.3 Elementary blocks used

The list of the elementary blocks used for the project realization is provided hereafter, along with the reference to the sections in which their functioning is described.

- *Bernoulli Binary Generator* (Section 6.3.1)
- *M-PAM Modulator Baseband* (Section 6.3.2)
- *Complex to Real-Imag* (Section 6.3.3)
- *Raised Cosine Transmit Filter* (Section 6.3.4)
- *ALSA Audio Capture* (Section 5.3.1)
- *Multiport Selector* (Section 7.3.6)
- *Data Type Conversion* (Section 4.3.2)
- *Product* (Section 7.3.8)
- *Max - Divide - Gain* (Section 6.3.5)
- *Matrix Concatenate* (Section 4.3.3)
- *ALSA Audio Playback* (Section 4.3.4)

7.3.1 Bernoully Binary Generator

The ***Bernoully Binary Generator*** block is described in Section 6.3.1. It represents the binary information source. Its task is to produce an output random sequence of independent bits with Bernoulli statistics. For the current project this block adopts the same configuration shown in Fig.6.4, that entails a bit rate $B_r = \frac{48000}{20} = 2400\ \frac{bit}{s}$.

7.3.2 M-PAM Modulator Baseband

The ***M-PAM Modulator Baseband*** block is described in Section 6.3.2. It converts the input bits $\in \{0, 1\}$ into 2-PAM symbols $\in$ {-1, 1}. For the current project this block adopts the same configuration shown in Fig.6.5.

As recalled in Section 6.3.2, despite the fact that 2-PAM symbols are purely real quantities, the ***M-PAM Modulator Baseband*** block generates each of them in the complex format, associating to each symbol an imaginary component with null value (e.g., $[..., 1+i0, -1+i0, ...]$). A ***Complex to Real-Imag*** block is thus needed in order to remove the imaginary components.

7.3.3 Complex to Real-Imag

The ***Complex to Real-Imag*** block is described in Section 6.3.3. Its task is to remove the symbols' imaginary component. For the current project this block adopts the same configuration shown in Fig.6.7.

7.3.4 Raised Cosine Transmit Filter

The ***Raised Cosine Transmit Filter*** block is described in Section 6.3.4. For the current project this block adopts the same configuration shown in Fig.6.8.

7.3.5 ALSA Audio Capture

The ***ALSA Audio Capture*** block is described in Section 5.3.1. For the current project this block adopts the same configuration shown in Fig.5.4.

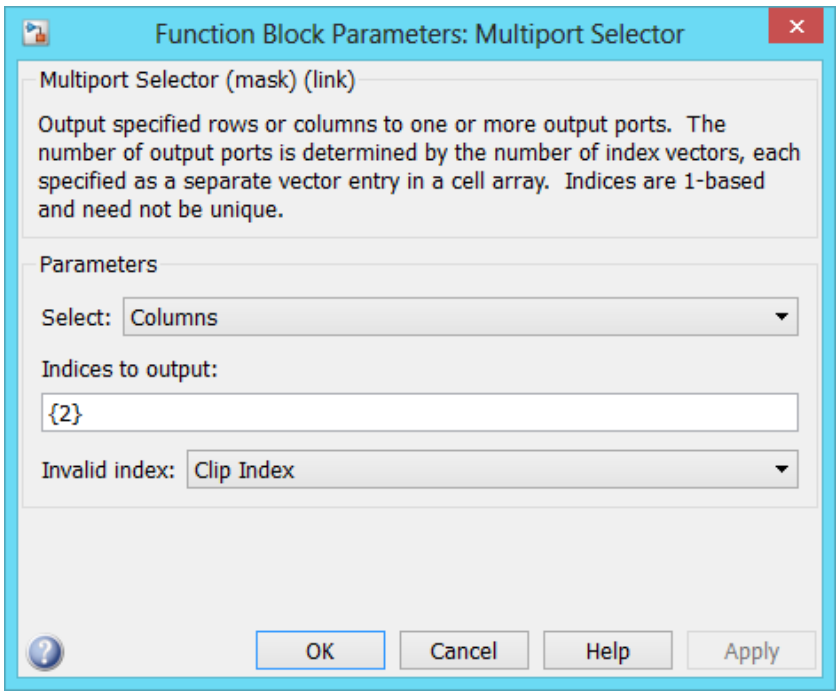


Figure 7.4: Configuration window of the ***Multiport selector*** block

7.3.6 Multiport selector

The ***Multiport selector*** block receives an input stereo signal, that is, a two-channel signal produced by the ***ALSA Audio Capture*** block, and selects only one of the channels, producing an output mono signal.

In the configuration window shown in Fig.7.4, the only relevant parameter is *Indices to output*, that is set at 2 in order to select the second channel only.

7.3.7 Data Type Conversion

The ***Data Type Conversion*** block is described in Section 4.3.2. It converts an input signal of any Simulink data type to the data type specified in its configuration window.

For the project here considered, in particular, the ***Data Type Conversion*** block within the ***Raspberry Pi system input*** macroblock performs the conversion from the *int16* data, supplied by the ***ALSA Audio Capture*** block, into the *double* data required by the ***Product*** block.

The ***Data Type Conversion2*** block, within the ***Raspberry Pi output*** macroblock, performs, instead, the opposite conversion, in order to adapt the signal to the *int16* format required by the ***ALSA Audio Playback*** block.

In both cases, the adopted configuration is shown in Fig.4.8.

7.3.8 Product

The ***Product*** block performs the product of its inputs, whose number is defined by the *Number of inputs* parameter of the configuration window. This parameter is set at 2, as shown in Fig.7.5.

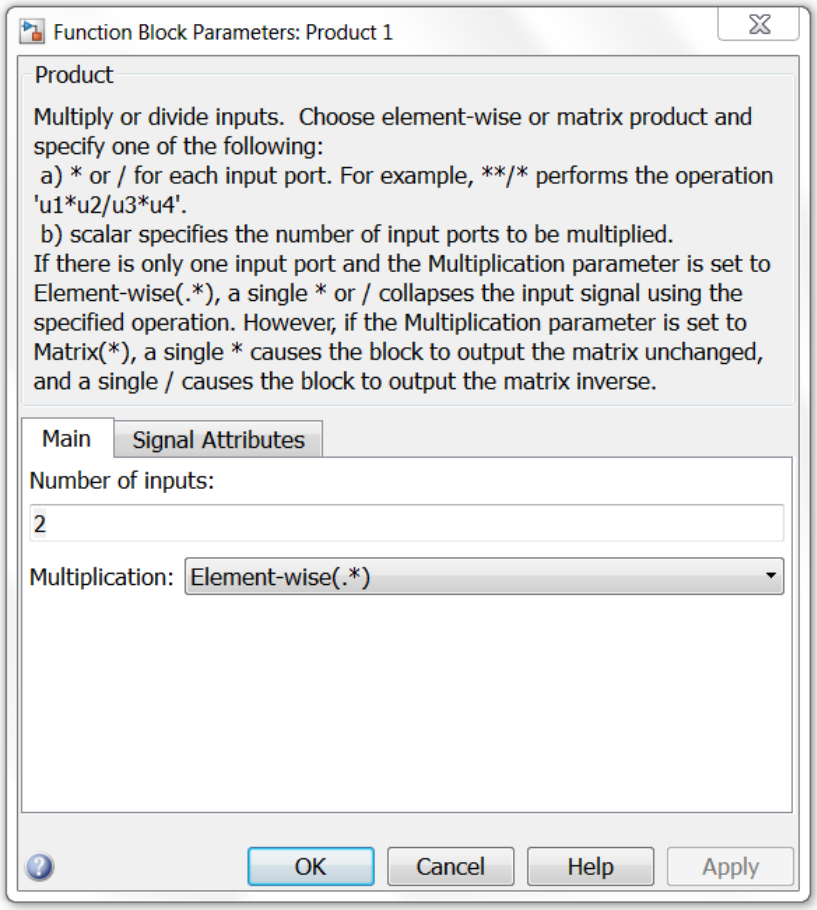


Figure 7.5: ***Product*** configuration window

7.3.9 Max-Divide-Gain (Automatic Gain Control)

The ***Max-Divide-Gain*** blocks, performing the automatic gain control, are described in Section 6.3.5.

For the current project this block adopts the same configuration shown in Fig.6.10 and in Fig.6.11.

7.3.10 Matrix Concatenate

The ***Matrix Concatenate*** block is described in Section 4.3.3. It is used to produce a two-channel output signal (that is, a stereo signal), starting from the two mono signals at its input. This operation is needed as the ***ALSA Audio Playback*** block, that follows the ***Matrix Concatenate*** block, requires a stereo input signal.

For the current project this block adopts the same configuration shown in Fig.4.9.

7.3.11 ALSA Audio Playback

The ***ALSA Audio Playback*** block is described in Section 4.3.4. It represents the Raspberry Pi2's analog output. Its task is to perform the digital-to-analog conversion of the signal and send it to the sound card for playback. For the current project this block adopts the same configuration shown in Fig.4.10.

7.3.12 Implementation and test of the 2-ASK transmitter

Once the Simulink project has been realized and tested through simulations, it is possible to carry out the *Deploy to Hardware* procedure, as described in Section 4.4. The project will be automatically executed as soon as the *deploy* procedure is completed. For the different execution modes see Section 4.5.

Connecting the Raspberry Pi2's input port (microphone port) to the signal generator, that generates the carrier with a frequency around 10-15 kHz, and the output port to the oscilloscope, according to the scheme in Fig.7.3, the 2-ASK signal can be finally observed, as shown in Fig.7.6.

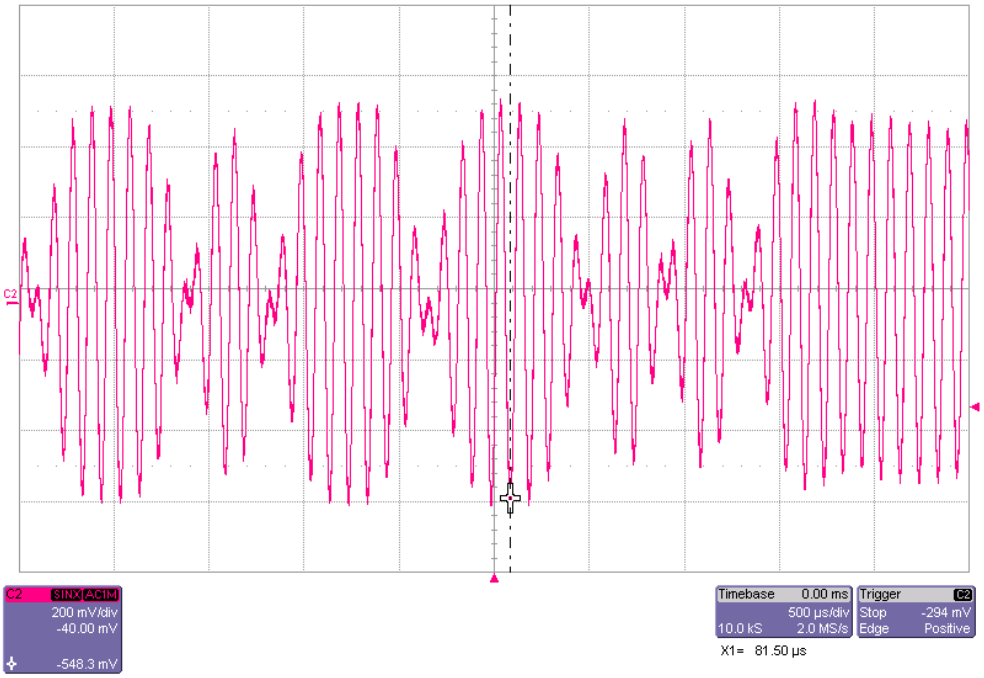


Figure 7.6: 2-ASK signal with 10 kHz carrier

7.4 4-ASK transmitter

The 4-ASK transmitter, shown in Fig.7.7, shows minimal variations compared to the previously described 2-ASK transmitter (Fig.7.2). The changes concern only the ***Baseband Modulation*** macroblock, in charge of generating the modulating signal. In the 4-ASK case, in fact, the ***Baseband Modulation*** macroblock must generate a 4-PAM signal.

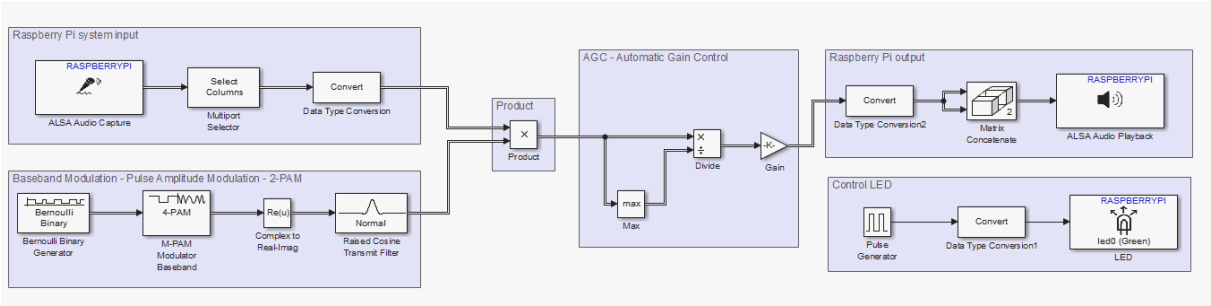


Figure 7.7: Simulink sheme 4-ASK modulator

The 4-ASK transmitter is obtained, therefore, from the 2-ASK transmitter by simply replacing the 2-ASK ***Baseband Modulation*** macroblock with the corresponding macroblock for the 4-ASK transmitter, described in Section 6.4.

7.5 Elementary blocks used

The list of the elementary blocks used for the realization of the project is provided hereafter, along with the reference to the sections in which their functioning is described.

- *Bernoulli Binary Generator* (Section 6.3.1)
- *M-PAM Modulator Baseband* (Section 6.5.2)
- *Complex to Real-Imag* (Section 6.3.3)
- *Raised Cosine Transmit Filter* (Section 6.3.4)
- *ALSA Audio Capture* (Section 5.3.1)
- *Multiport Selector* (Section 7.3.6)
- *Data Type Conversion* (Section 4.3.2)
- *Product* (Section 7.2.3)
- *Max - Divide - Gain* (Section 6.3.5)
- *Matrix Concatenate* (Section 4.3.3)
- *ALSA Audio Playback* (Section 4.3.4)

You can easily see that all the blocks were previously described.

7.5.1 Implementation and test of the 4-ASK transmitter

Once the Simulink project has been realized and tested through simulations, it is possible to carry out the *Deploy to Hardware* procedure, as described in Section 4.4. The project will be automatically executed as soon as the *deploy* procedure is completed. For the different execution modes see Section 4.5.

Connecting Raspberry Pi2's input port (microphone port) to the signal generator, that generates the carrier with a frequency around 10-15 kHz, and the output port to the oscilloscope, according to the scheme in Fig.7.3, the 4-ASK signal can be finally observed, as shown in Fig.7.6.

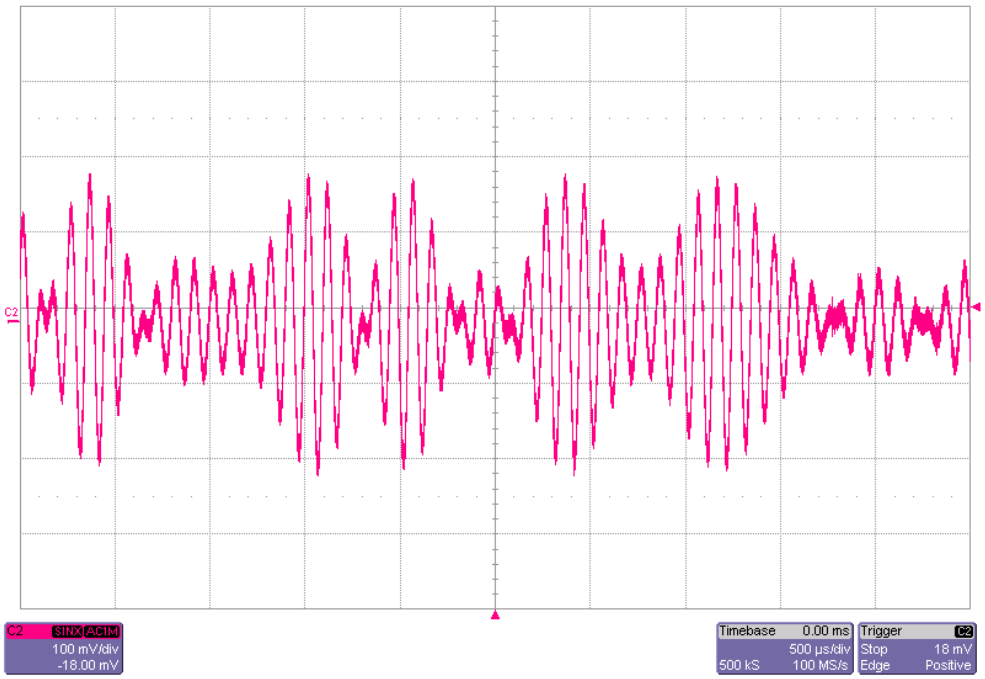
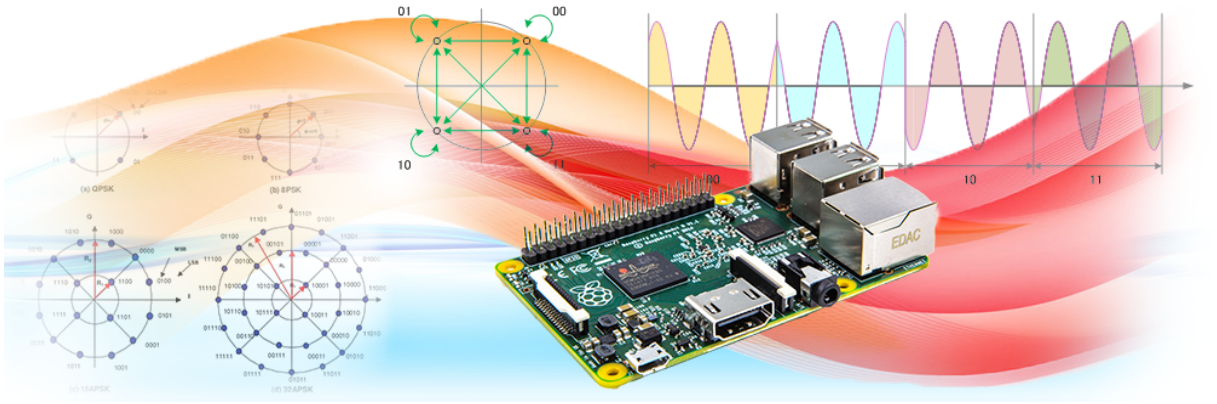


Figure 7.8: 4-ASK signal with 10 kHz carrier

Chapter 8

QPSK modulation with Raspberry Pi2



Quadrature phase shift keying (QPSK) is a digital modulation scheme in which two bits modulate the amplitudes of two carrier waves, using the 2-ASK digital modulation scheme. The two carrier waves differ in their phase by $\frac{\pi}{2}$ and are thus called quadrature carriers, hence the name of the scheme. The general expression of a QPSK signal is:

$$s(t) = \frac{V_o}{\sqrt{2}} \sum_{i=-\infty}^{\infty} a_{\mathrm{p}i} g(t - iT) \cos(2\pi f_o t) - \frac{V_o}{\sqrt{2}} \sum_{i=-\infty}^{\infty} a_{\mathrm{q}i} g(t - iT) \sin(2\pi f_o t) \tag{8.1}$$

where

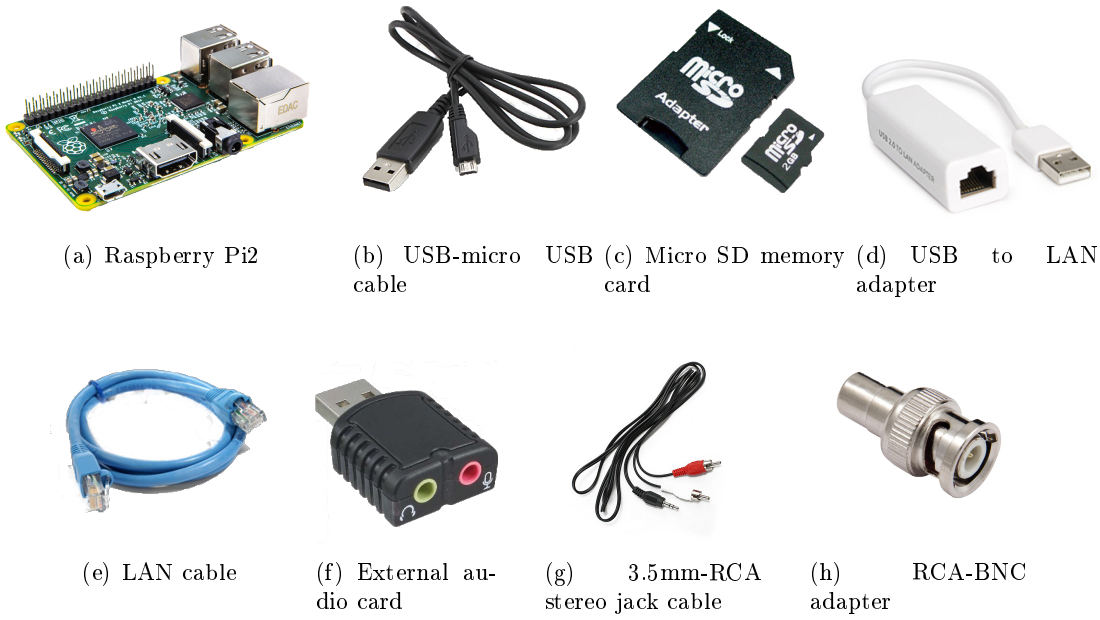
- V_0 is the carriers' amplitude, being $V_0 \cos(2\pi f_o t)$ and $V_0 \sin(2\pi f_o t)$ the two quadrature carriers;
- $a_{\mathrm{p}i} \in \{-1, 1\}$ and $a_{\mathrm{q}i} \in \{-1, 1\}$ are the in-phase and quadrature symbols;
- $g(t)$ is the waveform associated to each symbol;
- f_0 is the carrier frequency;
- T is the time interval between one symbol and the following one.

The QPSK signal expressed in 8.1 can be be generalized to produce an M-quadrature amplitude modulation (M-QAM) using two L-ASK signals to modulate the carriers, with $M = L^2$.

8.1 Equipment required for this experience

The experimental activity described in this chapter requires an oscilloscope and the following equipment:

- *Nr.1 Raspberry Pi2* (Fig.8.1(a))
- *Nr.1 Micro SD memory card* (Fig.8.1(c))
- *Nr.1 Network cable* (Fig.8.1(e))
- *Nr.1 3.5mm-RCA jack cable* (Fig.8.1(g))
- *Nr.1 USB-micro USB cable* (Fig.8.1(b))
- *Nr.1 USB-LAN adapter* (Fig.8.1(d))
- *Nr.1 External audio card* (Fig.8.1(f))
- *Nr.2 BNC-RCA adapter* (Fig.8.1(h))



(a) Raspberry Pi2 (b) USB-micro USB cable (c) Micro SD memory card (d) USB to LAN adapter

(e) LAN cable (f) External audio card (g) 3.5mm-RCA stereo jack cable (h) RCA-BNC adapter

Figure 8.1: Equipment

8.2 QPSK transmitter

The Simulink model used to implement a QPSK transmitter on a Raspberry Pi2 board is shown in Fig.8.2. Fig.8.3 shows, instead, the interconnections among the different devices constituting the workstation.

In order to make the scheme shown in Fig.8.2 as clear as possible, four macroblocks have been highlighted[1].

[1]The ***Control LED*** macroblock has been discussed in Section 4.2.3. In the following, therefore, the details of its functioning will not be provided any more.

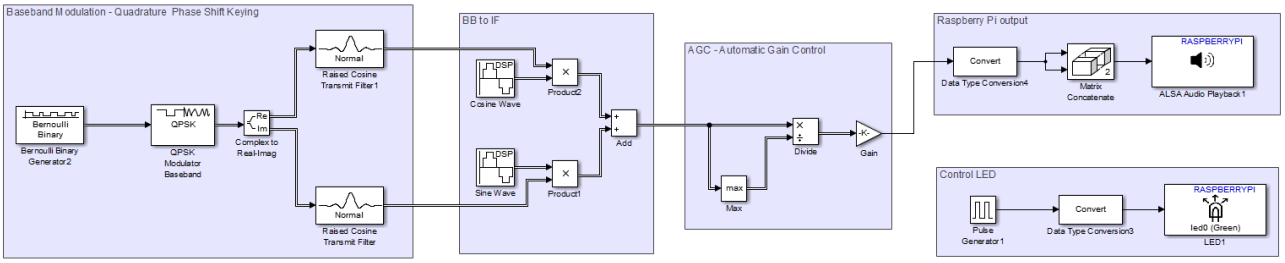


Figure 8.2: Simulink block scheme of the QPSK transmitter

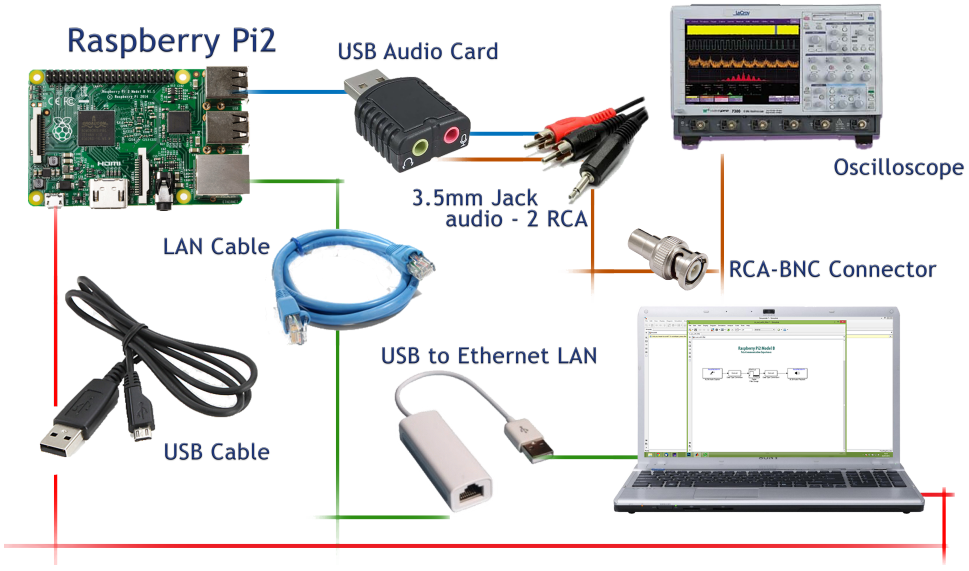


Figure 8.3: Raspberry Pi's Input/Output connection scheme

The first macroblock, called ***BaseBand Modulation - Quadrature Phase Shift Keying***, contains all the Simulink blocks which contribute to the generation of the QPSK modulated signal starting from the bits. The data, shaped with the appropriate filters, are then processed by the macroblock ***BB to IF***. At this point the signal passes through the ***Automatic Gain Control AGC*** stage, which adapts the signal dynamic to the level required by subsequent macroblock, called ***Rasberry Pi output***.

8.2.1 BaseBand Modulation - Quadrature Phase Shift Keying

The macroblock in charge of producing the QPSK baseband signal starts from the bits generated by the ***Bernoully Binary Generator*** and creates the correct phase variations that will be added later to the carrier. The phase variations are generated by the simulink block ***QPSK - Modulator Baseband***, described in Section 8.3.2. Another operation made by the macroblock is to separate the Real and Imaginary parts through the use of the ***Complex to Real-Imag***

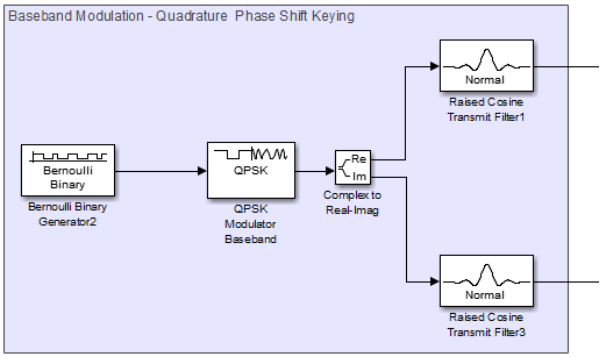


Figure 8.4: Baseband modulation - QPSK

block. Finally, the two sequences of symbols are filtered through the raised cosine shaping filters implemented in the ***Raised Cosine Transmit Filter*** block.

8.2.2 BB to IF

The ***BBtoIF*** macroblock converts the signal from the baseband to the bandwidth centered in the intermediate frequency f_{IF}. This operation is done through the use of a modulator the uses two carriers in quadrature at a frequency $f_{IF} = 15$ kHz.

The scheme of the ***BBtoIF*** macroblock indeed corresponds to the classic quadrature modulator, with a real and an imaginary part product modulated with carriers that in quadrature to each other. Conventionally, $cos(2\pi f_{IF} t)$ is used as the carrier of the real part and $-sin(2\pi f_{IF} t)$ as the carrier of the imaginary part. The quadrature carriers are thus generated by the ***Cosine Wave*** and ***Sine Wave*** blocks. The two signals are then summed into a single output signal.

In Fig.8.5, the settings of the blocks used to generate the carriers are shown. In particular, a phase shift of π is added to generate $-sin(2\pi ft)$, whereas a phase shift of $\pi/2$ is added to generate $cos(2\pi ft)$.

8.2.3 Max-Divide-Gain (Automatic Gain Control)

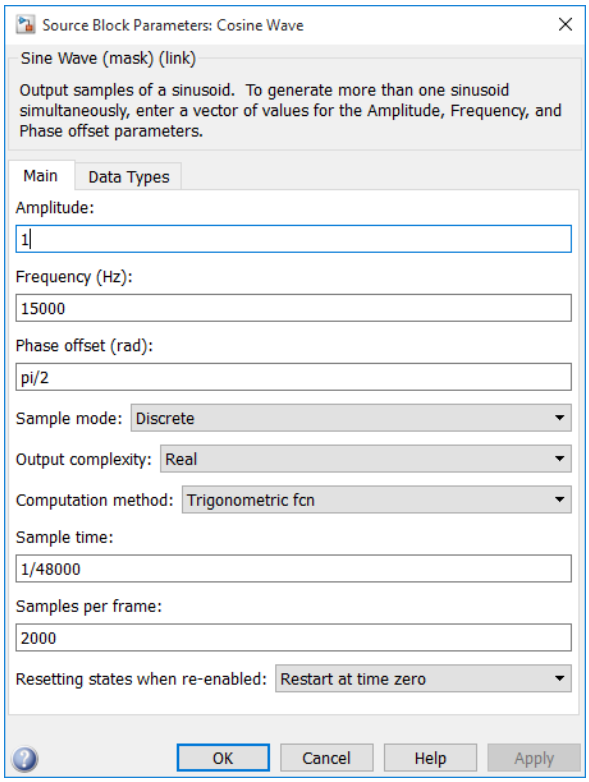
The blocks ***Max-Divide-Gain*** that realize the automatic control gain macroblock are detailed in Section 6.3.5. The adopted model uses the same settings as shown in Fig.6.10 and Fig.6.11.

8.2.4 Raspberry Pi output

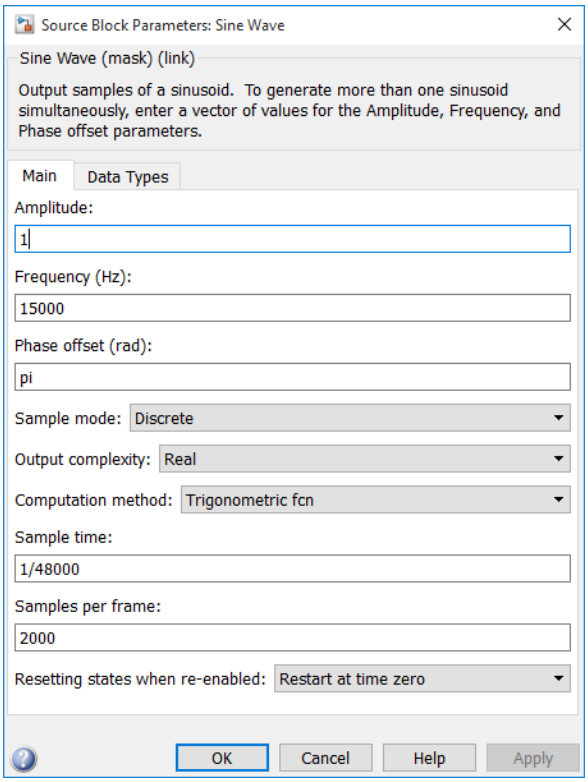
The **Raspberry Pi output** macroblock, described in Section 4.2.2, represents the signal output port. The ***Data Type Conversion2***, ***Matrix Concatenate***, and ***ALSA Audio Playback*** blocks have been described in Sections 4.3.2, 4.3.3, and 4.3.4, respectively.

8.2.5 Control LED

The ***Control LED*** macroblock, that is not really part of the QPSK modulator and just used to check that the scheme is running on the board, have been already described in Section 4.2.3.



(a) Settings of the phase carrier (cosine)



(b) Settins of the quadrature carrier (sine)

Figure 8.5: Settings of the ***Cosine Wave*** and ***Sine Wave*** blocks

8.3 List of the adopted Simulink blocks

- *Bernoulli Binary Generator* (Section 8.3.1)
- *QPSK Modulator Baseband* (Section 8.3.2)
- *Complex to Real-Imag* (Section 8.3.3)
- *Raised Cosine Transmit Filter* (Section 6.3.4)
- *Sine Wave/Cosine Wave*(Section 8.3.5)
- *Max-Divide-Gain* (Section 6.3.5)
- *Data Type Conversion* (Section 4.3.2)
- *Matrix Concatenate* (Section 4.3.3)
- *ALSA Audio Playback* (Section 4.3.4)

8.3.1 Bernoully Binary Generator

The ***Bernoully Binary Generator*** block has been described in Section 6.3.1. It represents the binary source of information, with the objective to generate an independent and random sequence of bits with a Bernoulli distribution. The present model uses the settings shown in Fig.8.6, with a bit rate $B_r = \frac{48000}{10} = 4800\ [\frac{bit}{s}]$.

8.3.2 QPSK Modulator Baseband

The ***QPSK Modulator Baseband*** block generates four different phase shifts, depending on the value of the incoming pairs of bits. A Gray coding is used to have adjacent symbols differing for a single bit, thus minimizing the bit error probability. The configuration of the ***QPSK***

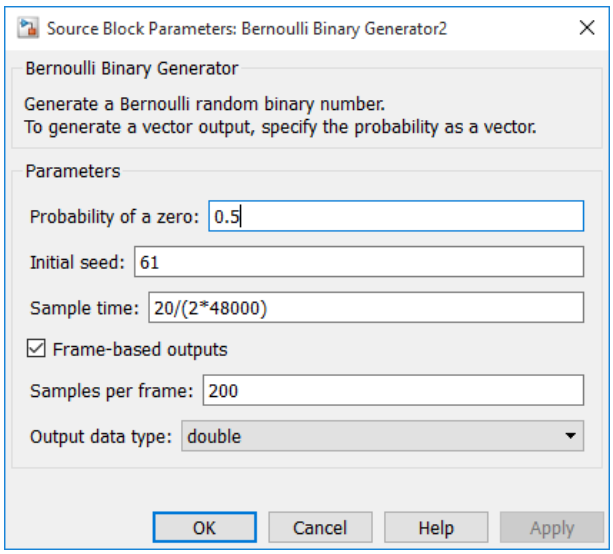


Figure 8.6: Settings of the Bernoulli Binary Generator block for the QPSK model

Modulator Baseband block is shown in Fig.8.7 and the constellation and phase values are are presented in Fig.8.8.

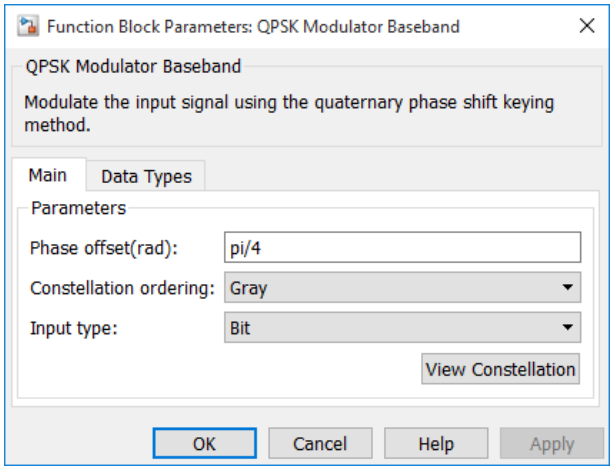
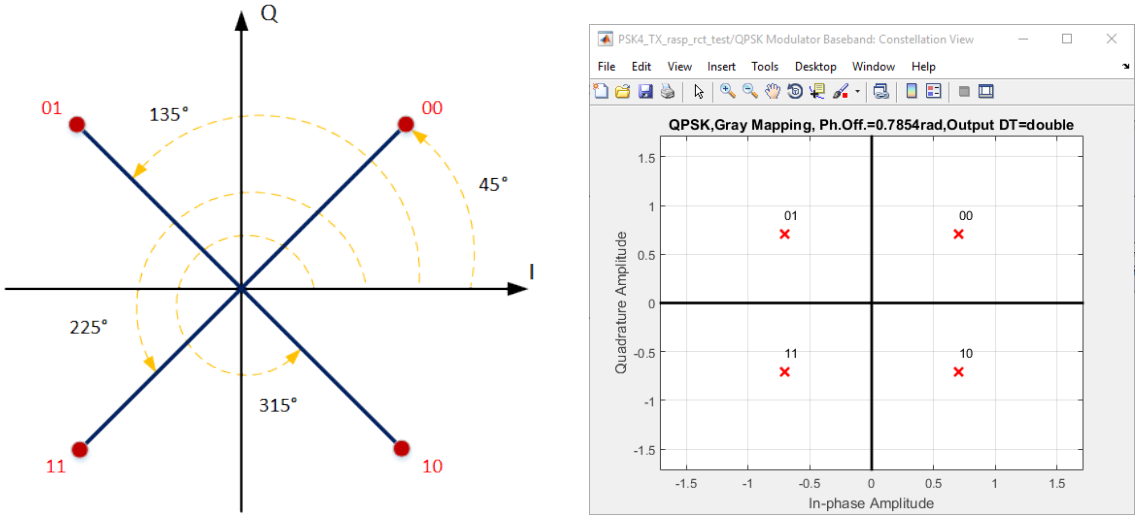


Figure 8.7: Settings of the QPSK Modulator Baseband block

8.3.3 Complex to Real-Imag

The ***Complex to Real-Imag*** block separates the real and imaginary parts of the complex signals provided at its input. This allows the processing of the in-phase and quadrature components. The settings of this block are shown in Fig.8.9.



(a) Phase values (b) QPSK constellation

Figure 8.8: Phase values and constellation of the QPSK modulation

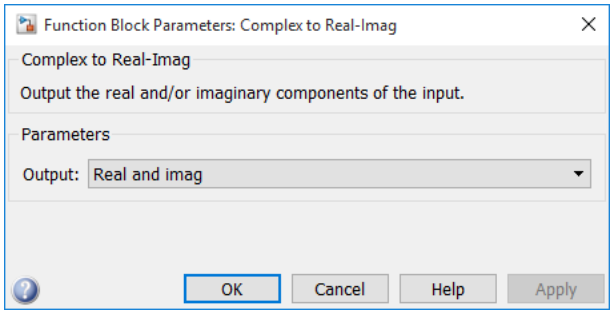


Figure 8.9: Settings of the Complex to Real-Imag block

8.3.4 Raised Cosine Transmit Filter

The ***Raised Cosine Transmit Filter*** block has been detailed in Section 6.3.4. The present model adopts the same settings shown in Fig.6.8.

8.3.5 Sine Wave/Cosine Wave

The ***Sine Wave*** and ***Cosine Wave*** blocks generate the two carriers, in quadrature to each other, that are required to bring the signal at the intermediate frequency. The settings of these blocks, similar to those discussed in Section 4.3.1, are shown in Fig.8.5.

8.3.6 Data Type Conversion

The ***Data Type Conversion*** block has been shown in Section 4.3.2. It converts the input data to the required type. This block autonomously inherits the correct input and output types as a function of the connected blocks.

8.3.7 Max-Divide-Gain (Automatic Gain Control)

The ***Max-Divide-Gain*** blocks, that realize the automatic gain control macroblock, have been discussed in Section 6.3.5. The present model adopts the same settings as those shown in Fig.6.10 and Fig.6.11.

8.3.8 Matrix Concatenate

The ***Matrix Concatenate*** block has been detailed in Section 4.3.3. It is used to duplicate the same signal over two channels, an operation that is necessary to provide a stereo-signal to the ***ALSA Audio Playback*** block. The present model adopts the same settings as those that are shown in Fig.4.9.

8.3.9 ALSA Audio Playback

The ***ALSA Audio Playback*** block has been discussed in Section 4.3.4. It represents the DAC converter of the Raspberry Pi2, thus generating the analog output signal to be provided to the headphone connector. The present model adoptts the same settings as those shown in Fig.4.10.

8.3.10 Implementation and test of the QPSK transmitter

Once the model has been created in Simulink and its correct operations have been tested through simulations, the *deploy to hardware* process is performed, as detailed in Section 4.4. With the connections shown in Fig.8.3, the output signal can be checked on the oscilloscope, as shown in Fig.8.10, and on the spectrum analyzer, as shown in Fig.8.11

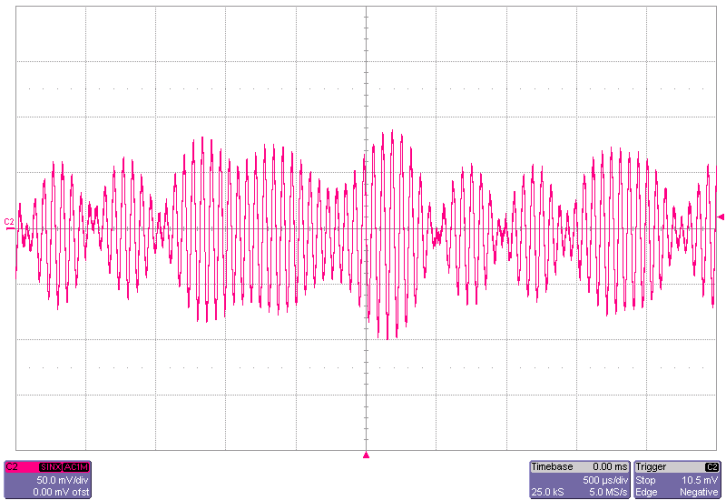


Figure 8.10: QPSK signal

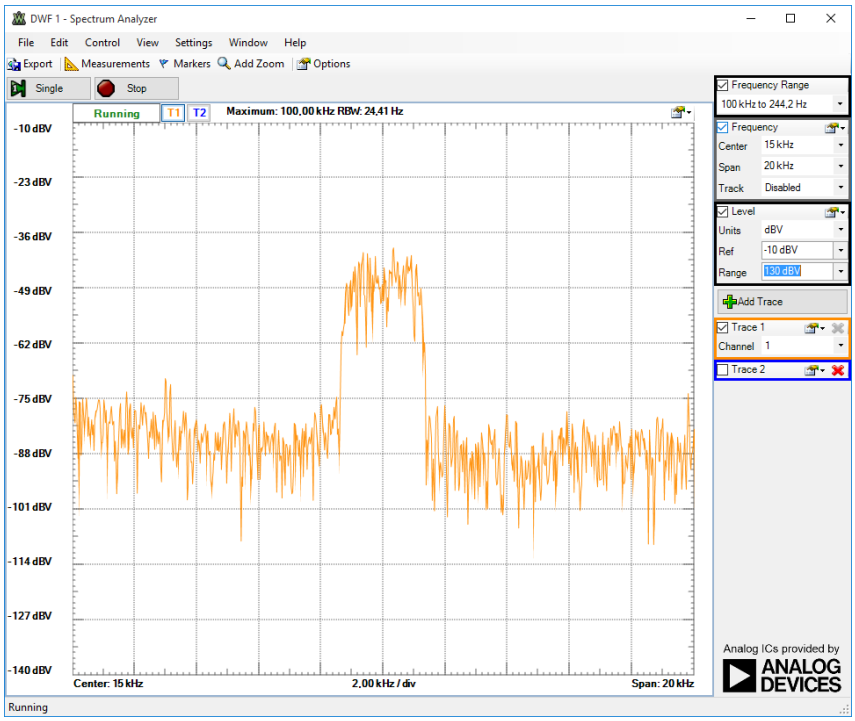
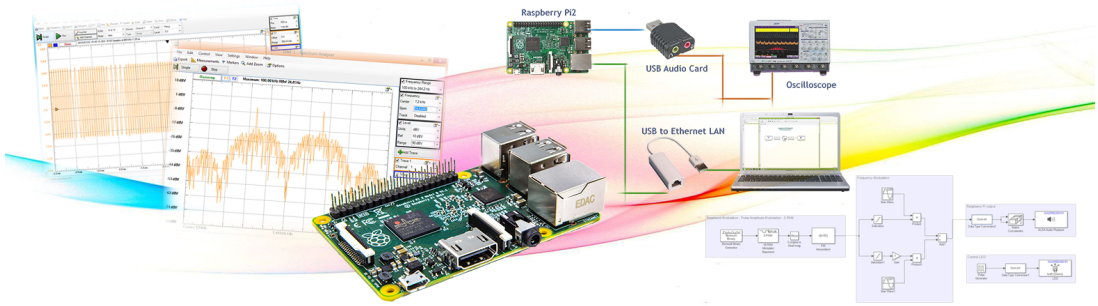


Figure 8.11: QPSK spectrum

Chapter 9

2-FSK modulation with Raspberry Pi2



Binary FSK (Frequency Shift Keying), usually referred to simply as 2-FSK, is a frequency modulation scheme in which the digital information is transmitted by shifting the frequency of a continuous carrier in a binary manner, so that one frequency or the other is used depending on the symbol being transmitted.

In the general case of FSK modulation with L levels (L-FSK), each symbol a_i, carrying the information of $\log_2 L$ bit, is associated to the corresponding frequency shift, according to the expression:

$$s(t) = V_0 cos\left[2\pi\left(f_0 + \Delta f \sum_{i=-\infty}^{\infty} a_i rect(\frac{t - iT}{T})\right)t\right], \tag{9.1}$$

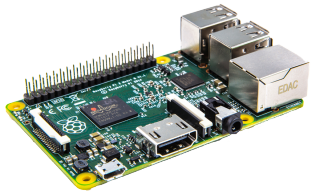
where

- V_0 represents the carrier amplitude;
- f_0 is the carrier frequency;
- Δf is the elementary frequency shift with respect to the carrier;
- a_i represents the generic symbol (in the case $L = 4$, for example, $a_i \in \{-3, -1, 1, 3\}$);
- T is the time interval between one symbol and the following one;
- $rect(t/T)$ is the square pulse equal to 1 in the interval $[-\frac{T}{2}, \frac{T}{2}]$ and 0 elsewhere.

9.1 Equipment required for this experience

The experimental activity described in this chapter requires an oscilloscope and the following equipment:

- *Nr.1 Raspberry Pi2* (Fig.9.1(a))
- *Nr.1 Micro SD memory card* (Fig.9.1(c))
- *Nr.1 Network cable* (Fig.9.1(e))
- *Nr.2 3.5mm-RCA jack cable* (Fig.9.1(g))
- *Nr.1 cavo USB-micro USB* (Fig.9.1(b))
- *Nr.1 USB-LAN adapter* (Fig.9.1(d))
- *Nr.1 External audio card* (Fig.9.1(f))
- *Nr.2 BNC-RCA adapters* (Fig.9.1(h))



(a) Raspberry Pi2

(b) USB-micro USB cable

(c) Micro SD memory card

(d) USB to LAN adapter

(e) LAN cable

(f) External audio card

(g) 3.5mm-RCA stereo jack cable

(h) RCA-BNC adapter

Figure 9.1: Equipment

9.2 2-FSK transmitter

In this chapter the Simulink model for the hardware implementation of a 2-FSK transmitter is introduced. With reference to eq.(9.1), in particular, the following settings are adopted: $V_0 = 2^{15} - 1$, $a_i \in \{-1, 1\}$, $\Delta f = 2.4$ kHz, $f_0 = 7.2$ kHz and $T = \frac{20}{48000}$ s.

As for the bit-frequency correspondence, a sine wave with frequency $f_1 = 4.8$ kHz is associated to the bit 1, whereas a sine wave with frequency $f_2 = 9.6$ kHz is associated to the bit 0.

In order to limit the signal bandwidth, it is advisable that the transitions between the two sine waves occur with phase continuity: The choice of f_1 and f_2, together with the choice of the bit rate $B_r = \frac{48000}{20} = 2400$ $[\frac{bit}{s}]$, guarantee this condition to be fulfilled. The duration $T = \frac{1}{Br}$ of a bit is, in fact, a multiple of the periods of the two sine waves used by the modulation.

In this project the two sine waves are generated within the model, with no need for external signal generators.

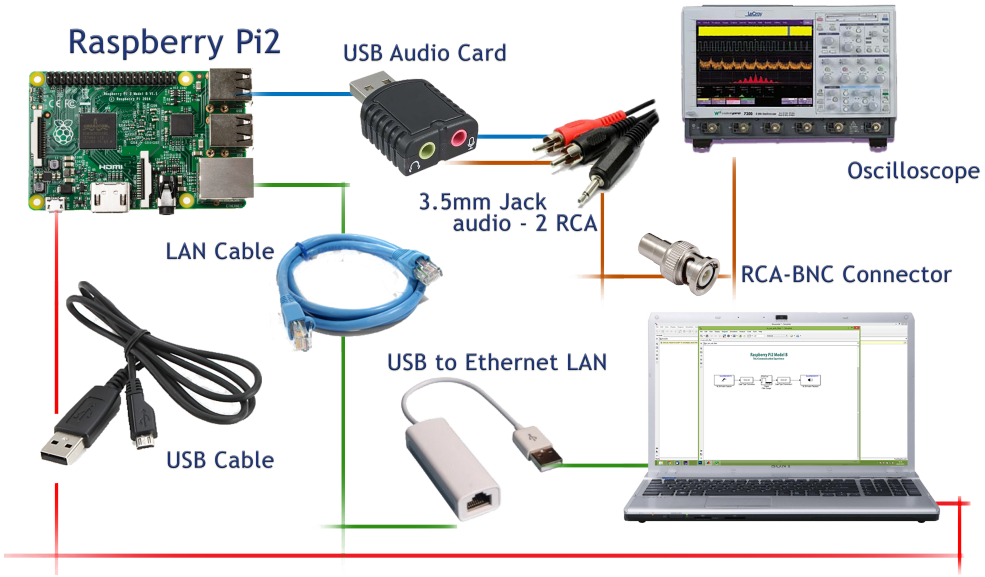


Figure 9.2: Connection scheme

The Simulink model of the 2-FSK transmitter is shown Fig.9.3, whereas Fig.9.2 shows the interconnections among the different devices constituting the workstation.

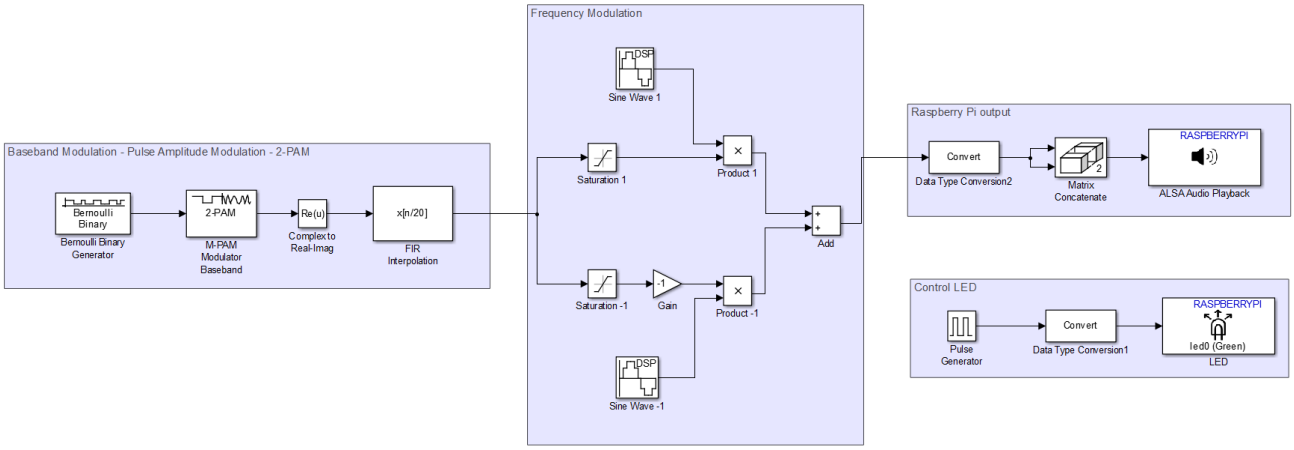


Figure 9.3: 2-FSK Simulink transmitter scheme

In order to make the scheme in Fig.9.3 as clear as possible, four macroblocks have been highlighted: ***Baseband Modulation***, ***Frequency Modulation***, ***Raspberry Pi output*** and ***Control LED***[1].

[1]The ***Control LED*** macroblock has been discussed in Section 4.2.3. In the following, therefore, the details of its functioning will not be provided any more.

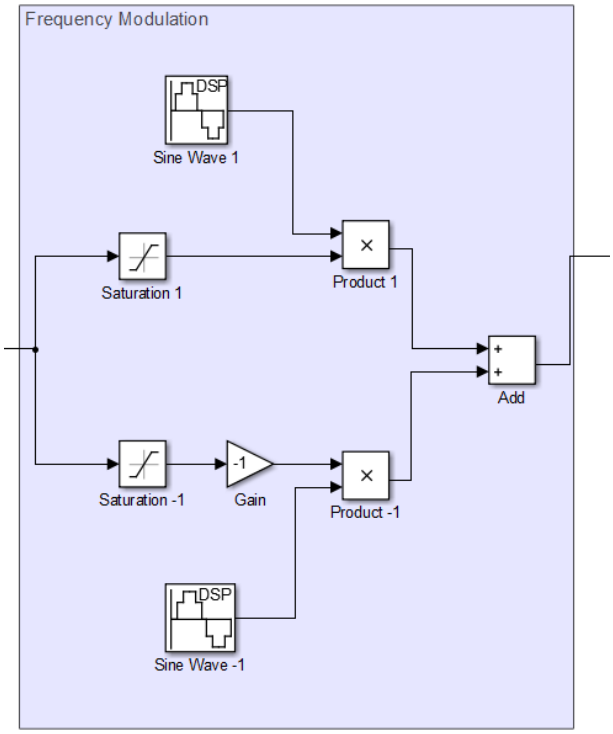


Figure 9.4: ***Frequency Modulation*** macroblock

9.2.1 Baseband Modulation

The ***Baseband Modulation*** macroblock is the 2-PAM modulator described in Section 6.6.1, whose task is to generate a 2-PAM signal with square pulses. Its elementary blocks ***Bernoulli Binary Generator***, ***M-PAM Modulator Baseband***, ***Complex to Real-Imag*** and ***FIR interpolation*** are described in Sections 6.3.1, 6.3.2, 6.3.3 and 6.6.1, along with the corresponding configurations, that are unchanged for the current project.

The ***Baseband Modulation*** output is, therefore, a 2-PAM signal with square pulses, like the one shown in Fig.6.24.

9.2.2 Frequency Modulation

The ***Frequency Modulation*** macroblock implements the 2-FSK modulator, which is the core of the whole project.

Depending on the current input symbol $a_i \in \{-1, 1\}$, the ***Frequency Modulation*** macroblock (Fig.9.4) outputs either the sine wave with frequency $f_1 = 4.8$ kHz, generated by the ***Sine Wave 1*** block, or the sine wave with frequency $f_2 = 9.6$ kHz, generated by the ***Sine Wave -1*** block.

The two ***Saturation*** blocks work as "switches", controlled by the symbols a_i, enabling only one of the two sine waves to reach the macroblock's output: When the input symbol is 1, the ***Saturation 1*** block outputs the value 1; with input -1, on the other hand, the output value is 0. This block controls the output of the ***Product 1*** block, producing the 4.8 kHz sine wave

generated by ***Sin Wave 1*** in case of symbol 1, or a null signal, in case of symbol -1.

The blocks sequence ***Saturation -1***, ***Gain***, ***Sin Wave -1*** and ***Product -1*** works in a perfectly specular way: In case of input symbol -1, the 9.6 kHz sine wave generated by ***Sin Wave 2*** is enabled at the ***Product -1*** output; in case of symbol 1, on the other hand, the output of the ***Product -1*** block is null.

It follows that, depending on the symbol to transmit, one of the ***Add*** block's inputs is null and the other one carries the sine wave associated to the symbol. The ***Add*** block's output, then, reproduces the sine wave to transmit.

Table 9.1 shows the outputs of the different blocks composing the ***Frequency Modulation*** macroblock for the possible values of the input symbol.

a_i **input**	***Saturation 1*** **output**	***Saturation -1*** **output**	***Product 1*** **output**	***Product -1*** **output**	***Add*** **output**
1	1	0	sine at 4.8 kHz	0	sine at 4.8 kHz
-1	0	-1	0	sine at 9.6 kHz	sine at 9.6 kHz

Table 9.1: ***Frequency Modulation*** macroblock

9.2.3 Raspberry Pi output

The ***Raspberry Pi output*** macroblock is described in Section 4.2.2. It represents the Raspberry Pi2's analog output. This macroblock adapts the signal at its input port to the format required by the Raspberry Pi2's DAC, represented by the ***ALSA Audio Playback*** block. For the current project its elementary blocks adopt the same configuration described in Section 4.3.2, 4.3.3 and 4.3.4.

9.2.4 Control LED

The only aim of the ***Control LED*** macroblock is to intermittently turn on and off the Raspberry Pi2's led during the execution of the project, visually confirming the that the project is running. This macroblock is discussed in Section 4.2.3.

9.3 Elementary blocks used

The list of the elementary blocks used for the project realization is provided hereafter, along with the reference to the sections in which their functioning is described.

- *Bernoulli Binary Generator* (Section 6.3.1)
- *M-PAM Modulator Baseband* (Section 6.3.2)
- *Complex to Real-Imag* (Section 6.3.3)
- *FIR Interpolation* (Section 6.6)
- *Sin Wave* (Section 9.3.5)
- *Saturation* (Section 9.3.6)
- *Gain* (sect.9.3.7)
- *Product* (Section 9.3.8)
- *Add* (Section 9.3.9)
- *Data Type Conversion* (Section 4.3.2)
- *Matrix Concatenate* (Section 4.3.3)
- *ALSA Audio Playback* (Section 4.3.4)

9.3.1 Bernoully Binary Generator

The ***Bernoully Binary Generator*** block is described in Section 6.3.1. It represents the binary information source. Its task is to produce an output random sequence of independent bits with Bernoulli statistics. For the current project this block adopts the same configuration shown in Fig.6.4, that entails a bit rate $B_r = \frac{48000}{20} = 2400\ \frac{bit}{s}$.

9.3.2 M-PAM Modulator Baseband

The ***M-PAM Modulator Baseband*** block is described in Section 6.3.2. It converts the input bits $\in \{0, 1\}$ into 2-PAM symbols $\in \{-1, 1\}$.

For the current project this block adopts the same configuration shown in Fig.6.5.

As recalled in Section 6.3.2, despite the fact that 2-PAM symbols are purely real quantities, the ***M-PAM Modulator Baseband*** block generates each of them in the complex format, associating to each symbol an imaginary component with null value (e.g., $[..., 1+i0, -1+i0, ...]$). A ***Complex to Real-Imag*** block is thus needed in order to remove the imaginary components.

9.3.3 Complex to Real-Imag

The ***Complex to Real-Imag*** block is described in Section 6.3.3. Its task is to remove the symbols' imaginary component.

For the current project this block adopts the same configuration shown in Fig.6.7.

9.3.4 FIR Interpolation

The ***FIR Interpolation*** block is described in Section 6.6.1. This block acts as a square pulse shaping filter. Its output port generates a 2-PAM signal with square pulses, as the one shown in Fig.6.24.

For the current project this block adopts the same configuration shown in Fig.6.23.

9.3.5 Sine Wave

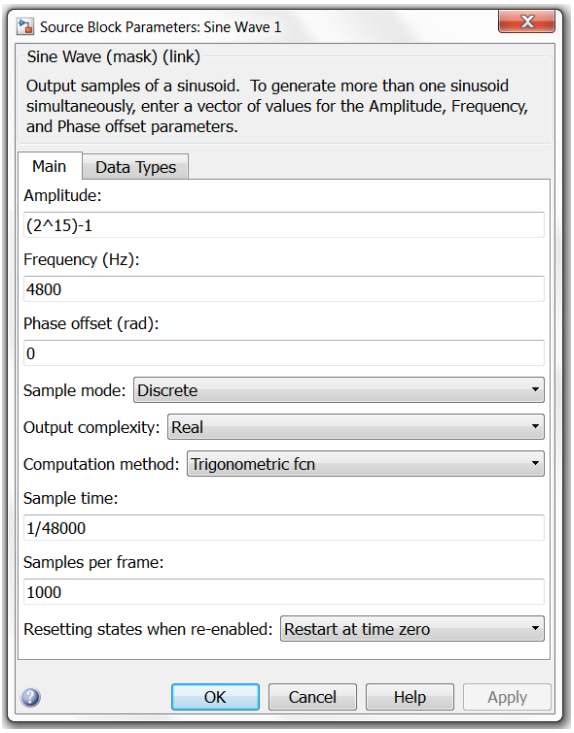
The ***Sine Wave*** block is described in Section 4.3.1. In the current project, the two ***Sine Wave*** blocks generates the two sine waves at 4.8 kHz and 9.6 kHz, associated to the symbols 1 and -1, respectively.

Their configuration windows are shown in Fig.9.5(a) and in Fig.9.5(b).

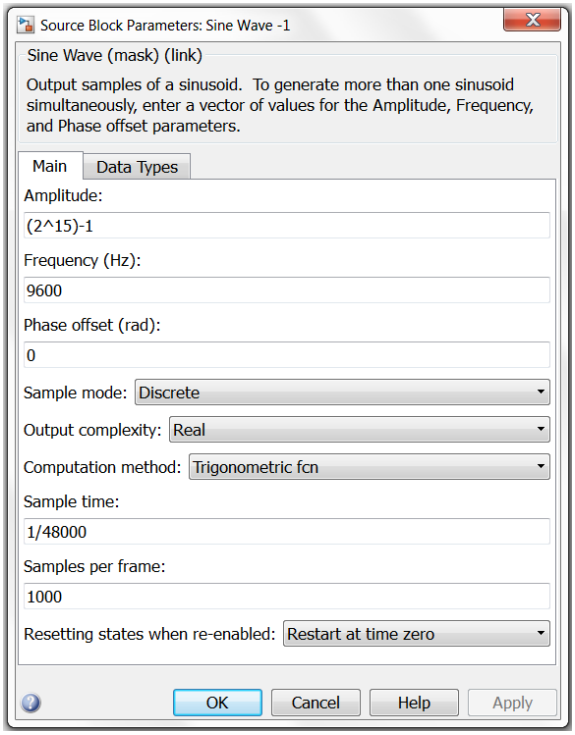
9.3.6 Saturation

The ***Saturation*** block limits its output signal within the range defined by the *Upper limit* and *Lower limit* parameters of its configuration window. In particular, when the input signal exceeds *Upper limit* the output is saturated at such value. On the other hand, the output is saturated at *Lower limit* when the input signal is under that value.

The configuration windows for the ***Saturation 1*** and ***Saturation -1*** blocks used in this project are shown in Fig. 9.6(a) and in Fig.9.6(b), respectively.

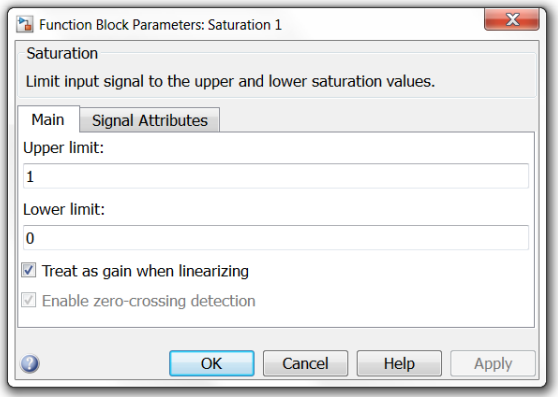


(a) Configuration window of the ***Sin Wave 1*** block

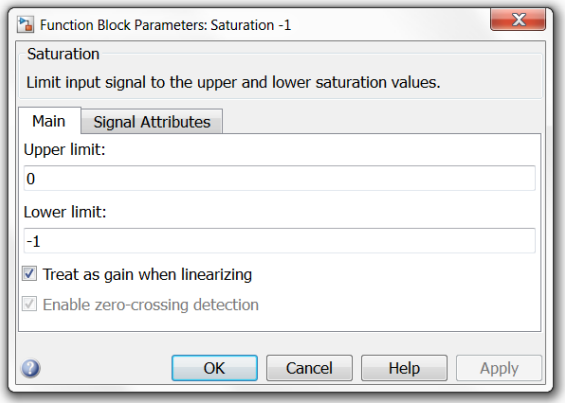


(b) Configuration window of the ***Sin Wave -1*** block

Figure 9.5: ***Sin Wave*** blocks' configuration windows



(a) Configuration window of the ***Saturation 1*** block



(b) Configuration window of the ***Saturation -1*** block

Figure 9.6: ***Saturation*** blocks' configuration windows

9.3.7 Gain

The ***Gain*** block reproduces, as output signal, the input signal multiplied by the *Gain* factor defined in its configuration window. The configuration adopted in the current project is shown in Fig.9.7.

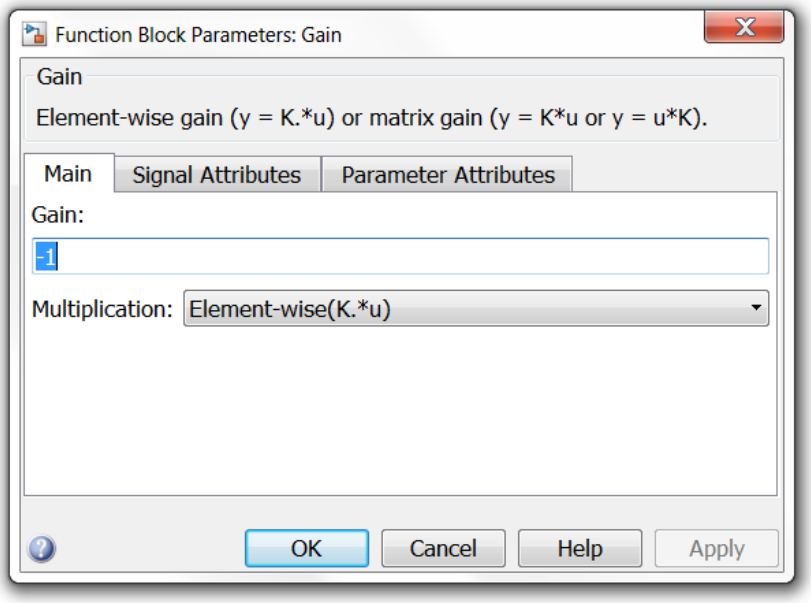


Figure 9.7: COnfiguration window of the ***Gain*** block

9.3.8 Product

The ***Product*** block performs the product of its inputs, whose number is defined by the *Number of inputs* parameter requested by the configuration window. This parameter is set at 2, as shown in Fig.7.5.

9.3.9 Add

The ***Add*** block performs the sum or the difference of its inputs. In the this case the block must sum up two inputs, that's why the *List of signs* field of its configuration window is set as "++", as shown in Fig.9.8.

9.3.10 Data Type Conversion

The ***Data Type Conversion*** block is described in Section 4.3.2. It converts an input signal of any Simulink data type to the data type specified in its configuration window.

For the project here considered, in particular, the ***Data Type Conversion*** block performs the conversion into the *int16* format, as required by the block ***ALSA Audio Playback***. For the current project this block adopts the same configuration shown in Fig.4.8.

9.3.11 Matrix Concatenate

The ***Matrix Concatenate*** block is described in Section 4.3.3. It is used to produce a two-channel output signal (that is, a stereo signal), starting from the two mono signals at its input. This operation is needed as the ***ALSA Audio Playback*** block, that follows the ***Matrix Concatenate*** block, requires a stereo input signal. This block adopts the same configuration shown in Fig.4.9.

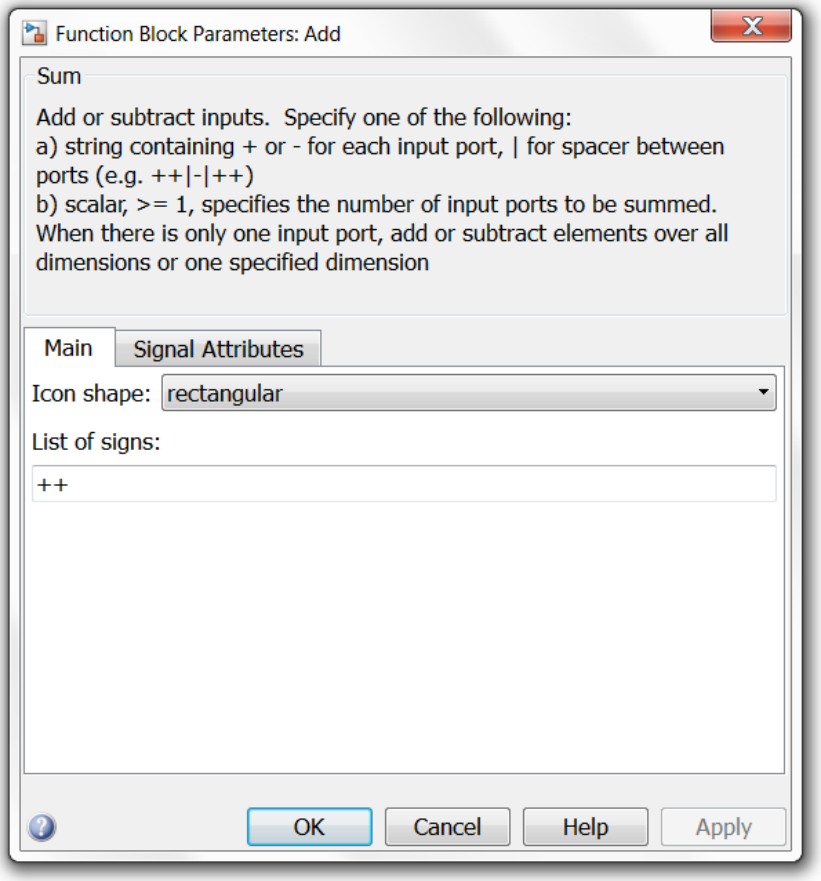


Figure 9.8: Configuration window of the ***Add*** block

9.3.12 ALSA Audio Playback

The ***ALSA Audio Playback*** block is described in Section 4.3.4. It represents the Raspberry Pi2's analog output. Its task is to perform the digital-to-analog conversion of the signal and send it to the sound card for playback.

For the current project this block adopts the same configuration shown in Fig.4.10.

9.3.13 Implementation and test of the 2-FSK transmitter

Once the Simulink project has been realized and tested through simulations, it is possible to carry out the *Deploy to Hardware* procedure, as described in Section 4.4. The project will be automatically executed as soon as the *deploy* procedure is completed. For the different execution modes see Section 4.5.

Connecting the Raspberry Pi2's analog output to the oscilloscope, according to the scheme in Fig.9.2, the 2-FSK signal can be finally observed, as shown in Fig.9.9.

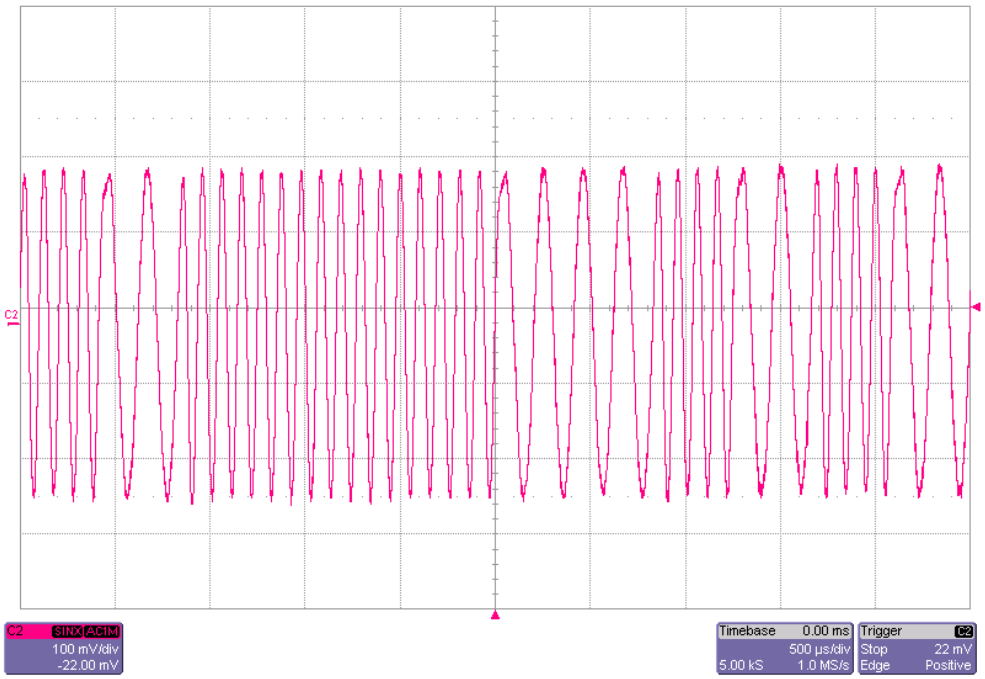
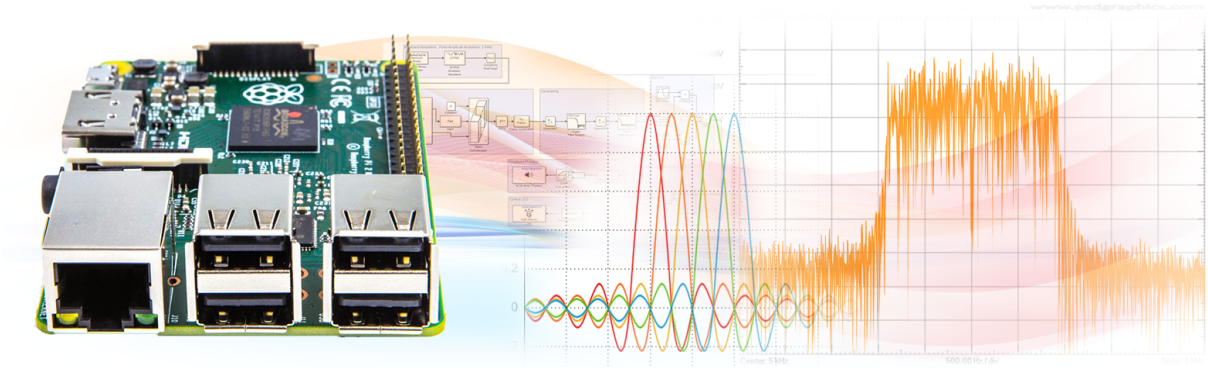


Figure 9.9: Measured 2-FSK signal

Chapter 10

Raspberry Pi2 as an OFDM transmitter



In this chapter we will describe the implementation of an Orthogonal Frequency Division Multiplexing (OFDM) transmitter on a Raspberry Pi2 board.

The OFDM modulation is a digital modulation scheme that divides the transmission among N different carriers at adjacent frequencies, usually called subcarriers, that are transmitted simultaneously.

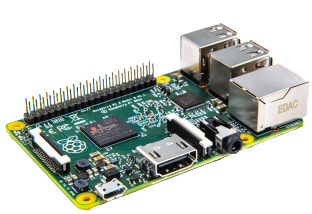
In the following we will describe the implementation of an OFDM transmitter operating with $N = 64$ subcarriers, $N_u = 48$ of which actually used to transmit the digital data.

The OFDM modulator described in the following does not implement the cyclic prefix, that will be introduced in future versions.

10.1 Equipment required for the realization of this experience

The experimental activity described in this chapter requires a spectrum analyser and the following equipment:

- *Nr.1 Raspberry Pi2* (Fig.10.1(a))
- *Nr.1 micro SD memory card* (Fig.10.1(c))
- *Nr.1 Network cable* (Fig.10.1(e))
- *Nr.1 3.5mm-RCA jack cable* (Fig.10.1(g))
- *Nr.1 USB-micro USB cable* (Fig.10.1(b))
- *Nr.1 USB-LAN adapter* (Fig.10.1(d))
- *Nr.1 External audio card* (Fig.10.1(f))
- *Nr.1 BNC-RCA adapter* (Fig.10.1(h))



(a) Raspberry Pi2

(b) USB-micro USB cable

(c) Micro SD memory card

(d) USB to LAN adapter

(e) LAN cable

(f) External audio card

(g) Stereo 3.5mm-RCA jack cable

(h) RCA-BNC adapter

Figure 10.1: Equipment

10.2 Raspberry Pi2 as OFDM transmitter

In Fig.10.2 the interconnection of the different devices composing the workstation is represented. The model of the OFDM transmitter is shown, instead, in Fig.10.3.

In order to be as clear as possible, seven macroblocks have been highlighted in the scheme shown in Fig.10.3: ***Baseband Modulation***, ***OFDM***, ***Upsampling***, ***BBtoIF***, ***AGC-Automatic Gain Control***, ***Raspberry Pi output*** and ***Control LED***[1], described below.

10.2.1 Baseband Modulation

The ***Baseband Modulation*** macroblock generates real symbols $a_i \in \{-1, 1\}$, starting from the bits produced by the ***Bernoully Binary Generator*** block. Such symbols will be used to modulate the subcarriers of the OFDM signal; as the alphabet used has just two symbols, the modulation adopted for the subcarriers is 2-ASK.

The ***M-PAM Modulator Baseband*** and ***Complex to Real-Imag*** elementary blocks have been described in Sections 6.3.2 and 6.3.3, where they are used for the generation of a 2-PAM signal. The corresponding configurations do not change for the project here considered.

[1]The ***Control LED*** macroblock has been discussed in Section 4.2.3. In the following, therefore, the details of its functioning will not be provided any more.

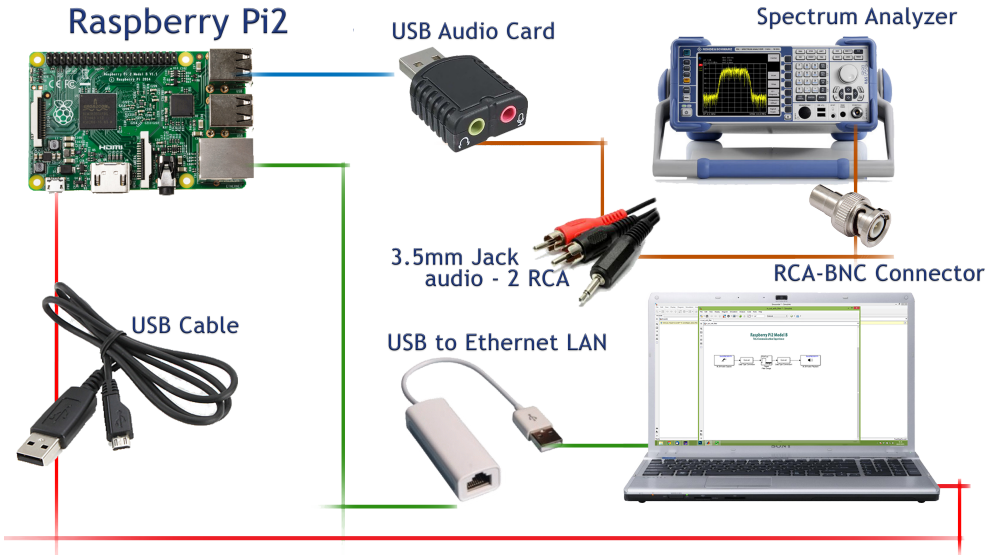


Figure 10.2: Connection scheme

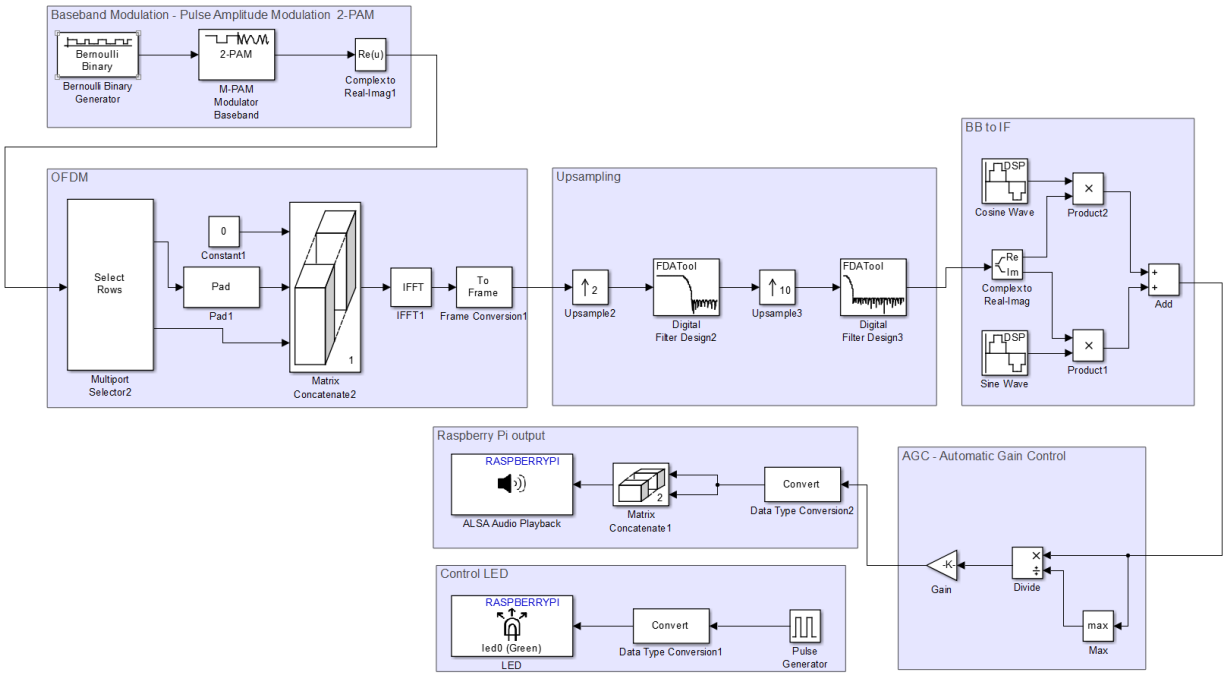


Figure 10.3: Simulink model of the OFDM modulator

On the contrary, the ***Bernoulli Binary Generator*** block requires a specific configuration, that will be discussed in the following section.

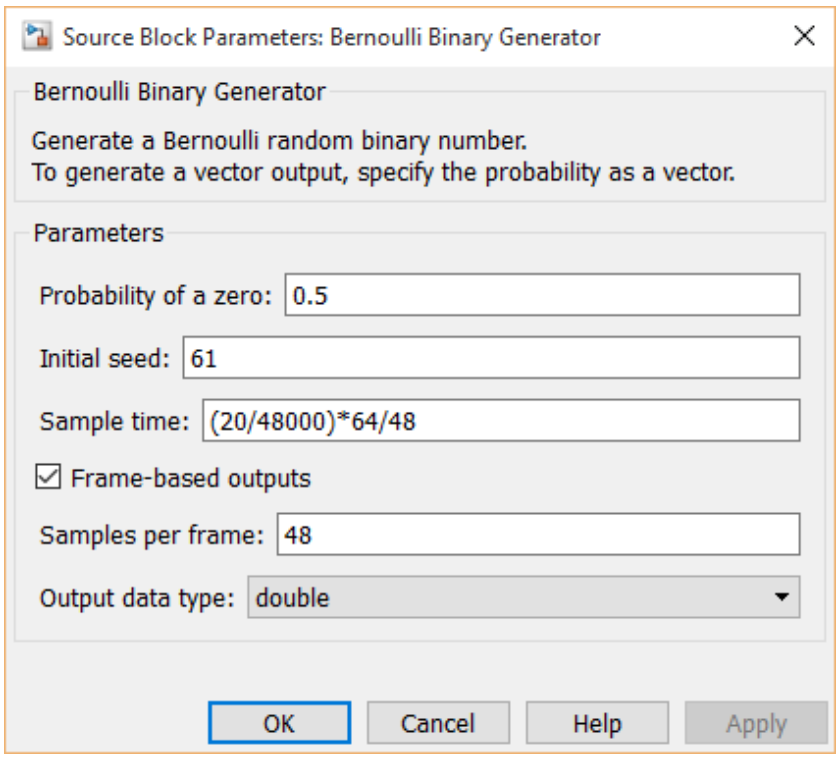


Figure 10.4: Configuration window of the ***Bernoulli Binary Generator*** block

Bernoulli Binary Generator

The ***Bernoulli Binary Generator*** block represents, as in the previous cases, the binary information source. With reference to the configuration window shown in Fig.10.4, the bits are generated at a rate of one bit every *Sample time* seconds and are grouped, before passing them to the following block, in *frames* with length equal to *Sample per frame*.

In this particular case, the value assigned to the *Sample time* parameter must fulfil the condition:

$$Sample\ Time = \frac{Upsampling\ factor}{b_{symbol} \cdot Audio\ Sampling\ Frequency} \cdot \frac{N}{N_u}\,, \tag{10.1}$$

where

- *Upsampling factor* represents the upsampling factor introduced by the ***Upsampling*** macroblock, that we will discuss in section 10.2.3;
- *Audio Sampling Frequency* is the sampling frequency defined in the ***ALSA Audio Playback*** block (section 4.3.4);
- $b_{symbol} = \log_2 M$ is the number of bits used to generate modulation symbols for each subcarrier, with M denoting the number of symbols of the adopted modulation;
- N is the total number of subcarriers used;

- N_u is the number of subcarriers actually used to transmit data.

In this specific case, as *Audio Sampling Frequency*=48000, *Upsampling factor*=20, $b_{symbol} = \log_2 M = 2$, $N = 64$ and N_u=48, it results $Sample\ Time = \frac{20}{48000} \cdot \frac{64}{48}$.

The *Sample per Frame* parameter must be assigned a value that corresponds to the number N_u of subcarriers actually used to transmit data. In this specific case, therefore, *Sample per Frame*=48.

The remaining parameters of the ***Bernoulli Binary Generator*** configuration window, whose meaning was discussed in paragraph 6.3.1, are defined as shown in Fig.10.4.

The output of the ***Baseband Modulation*** macroblock is thus a sequence of *frames*, one following the other, containing the 48 symbols that modulate the $N_u = 48$ useful subcarriers, as shown in (10.2):

$$[\underbrace{symbol\#1,\ symbol\#2, ...,\ symbol\#48}_{\text{frame}}], \tag{10.2}$$

10.2.2 OFDM

Starting from the modulation symbols a_i at its input, the ***OFDM*** macroblock generates the baseband signal corresponding to an OFDM modulation with $N = 64$ subcarriers, $N_u = 48$ of which are actually used to transmit modulation symbols (useful subcarriers) whereas the remaining $N_z = 16$ have null amplitude (15 virtual subcarriers + 1 DC subcarrier in correspondence of the frequency zero[2]). In this specific case, the $N = 64$ subcarriers are used as follows:

- 8 virtual subcarriers (from 1 to 8) with null amplitude;
- 24 data subcarriers (from 9 to 32) with 2-ASK modulation;
- 1 DC subcarrier (number 33) with null amplitude (corresponding to the zero frequency);
- 24 data subcarriers (from 34 to 57) with 2-ASK modulation;
- 7 virtual subcarriers (from 58 to 64) with null amplitude.

In Fig.10.5 the baseband spectrum[3] of the signal generated by the ***OFDM*** macroblock is shown, along with a reference number indicating the position of each subcarrier in the subcarriers' sequence.

Denoting with Fs the sampling frequency of the signal generated by the ***OFDM*** macroblock and with Δf the interval between two consecutive subcarriers, the spectral component contained in the Nyquist band $[-\frac{Fs}{2}, \frac{Fs}{2}]$ is represented with a continuous line, while the periodic repetitions with period $Fs = N\Delta f$, caused by the discrete-time nature of the signal, are represented with a dashed line. In Fig.10.6 the spectrum has been enriched with the indication of the symbol number[4] associated to each subcarriers. A null amplitude is assigned to the remaining subcarriers (DC and virtual).

[2] In order to facilitate the receiver in the research of the band center, the subcarrier corresponding to the zero frequency (in the baseband) is usually assigned a null amplitude. The acronym DC means Direct Current.

[3] It is obviously an ideal schematic representation.

[4] The symbol number $\in \{1, 2, ..., 48\}$ represents the position of the symbol in the *frame*.

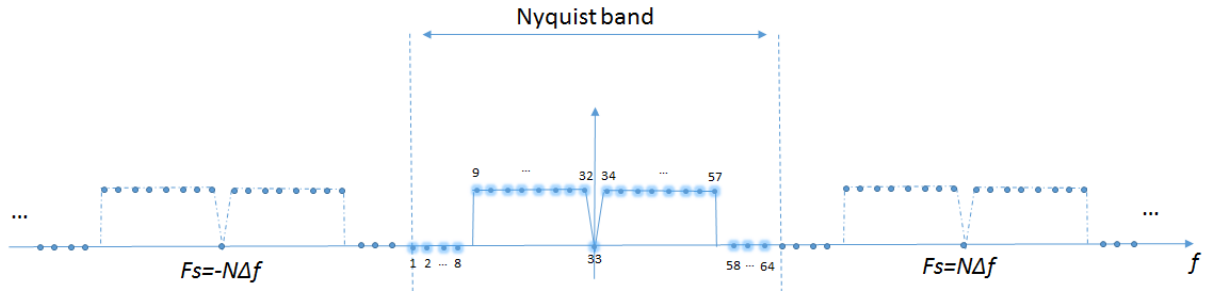


Figure 10.5: Signal spectrum

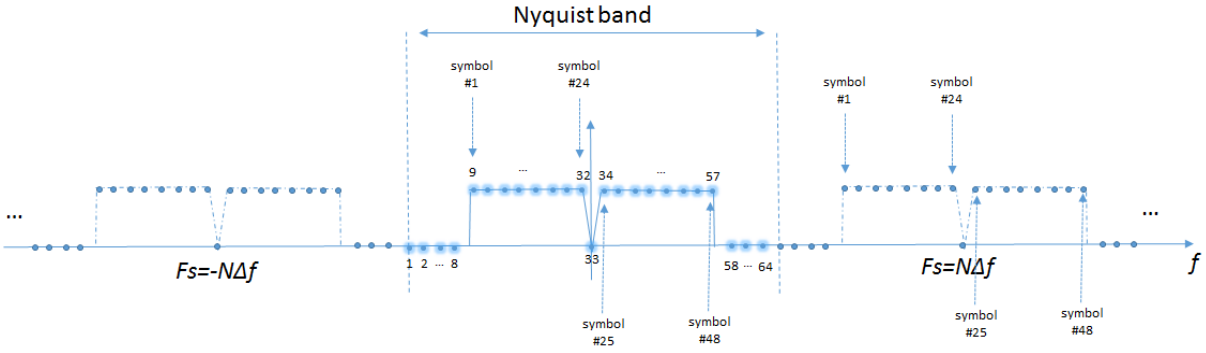


Figure 10.6: Signal spectrum with the symbol number associated to each subcarrier

The generation of this signal is the task of the ***IFFT*** block, that receives the N symbols associated to all the N subcarriers (48 useful, 15 virtual and 1 DC) and produces the baseband OFDM signal.

The ***IFFT*** block, however, needs to receive the symbols associated to the subcarriers according to a particular order: the first symbol must be the one associated to the subcarrier with 0 frequency, the second symbol the one associated to the subcarrier with frequency Δf and so on, until the last symbol, that must be the one associated to the subcarrier with frequency $(N-1)\Delta f$. In other words, the order of the $N = 64$ symbols at the input of the ***IFFT*** block must be the one shown in Fig.10.7, that is:

$$[0, \quad symbol\#25, ..., symbol\#48, \quad \underbrace{0, .., 0}_{15 \text{ zeros}}, \quad symbol\#1, ..., symbol\#24]. \tag{10.3}$$

The first step to pass from the 48 symbols frame (10.2) generated by the ***Baseband Modulation*** macroblock, to the 64 symbols frame (composed by 48 symbols+16 zeroes) reported in (10.3), is performed by the ***Multiport Selector2*** block. This block splits the 48 symbols *frame* (10.2) at its input into two *frames* containing 24 symbols each. In particular, the *frame* it generates at the first output port contains those symbols occupying positions 25 to 48 in the original *frame*, while the *frame* at the second output port contains those symbols occupying positions 1 to 24 in the original *frame*. The corresponding settings of the block's configuration window are shown in Fig.10.8.

The next step consists in the construction of the frame (10.3) by inserting in the proper

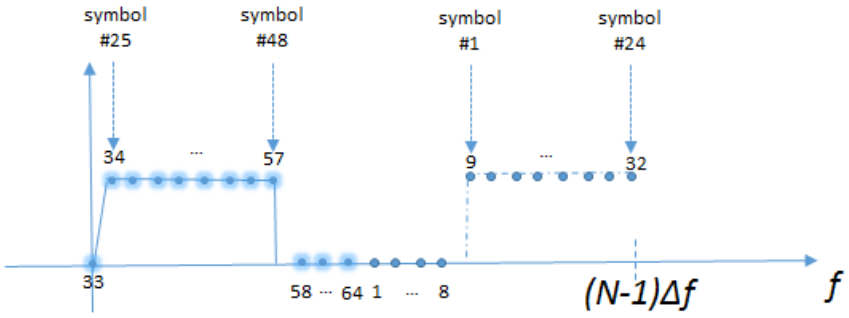


Figure 10.7: Signal spectrum with the symbol number associated to each subcarrier

position the 16 zeroes corresponding to the DC subcarrier and to the 15 virtual subcarrier. The block ***Pad1*** has the task to create a vector of 39 elements, the first 24 of which correspond to the symbols from 25 to 48 (that is, the first output of ***Multiport Selector2***), and the remaining 15 are 0s and represent the 15 virtual carriers. The configuration setting for this block is shown in Fig.10.9.

In order to eventually get to the frame (10.3), the block ***Matrix Concatenate*** is used, concatenating in a single *frame* its 3 inputs, corresponding to

- the constant 0, corresponding to the amplitude of the DC subcarriers;
- the *frame* with the symbols 25 to 48 followed by 15 zeroes, generated by ***Pad1*** block;
- the *frame* with the symbols 1 to 24, provided by the second output of the ***Multiport Selector2*** block.

The settings of the ***Matrix Concatenate*** block is shown in Fig.10.10.

Starting from the *frame* (10.3), it is finally possible to generate the baseband OFDM signal simply using the inverse discrete Fourier transform ***IFFT*** block, that associates the input symbols in the time domain to the corresponding orthogonal subcarriers in the frequency domain. The configuration of this block is shown in Fig.10.11.

In order to keep the elaboration in *frame based* mode, a ***Frame conversion*** block is finally inserted, following the ***IFFT*** block. The corresponding configuration window is shown in Fig.10.12.

10.2.3 Upsampling

The baseband signal generated by the ***OFDM*** macroblock has a sampling frequency of $Fs = \frac{48000}{20} = 2400\ \frac{samples}{s}$. The sampling rate $\frac{1}{Sample\ Time} = \frac{48000}{20}\frac{48}{64} = 1800\ \frac{samples}{s}$ chosen in the ***Bernoully Binary Generator*** block has been, in fact, increased by a factor $\frac{64}{48}$ by the ***IFFT*** block, that outputs a *frame* of 64 elements for each *frame* of 48 elements received as input.

In order to modulate the baseband OFDM signal, the sampling frequency must be, therefore, increased. It is convenient, on this regard, to increase it by a factor of 20, taking it to the value $48000\ \frac{samples}{s}$ required by Raspberry Pi2's DAC.

The upsampling operation is performed by the sequence of blocks included in the ***Upsampling*** macroblock, that performs an upsampling by a factor of 2 at first and then by a factor of

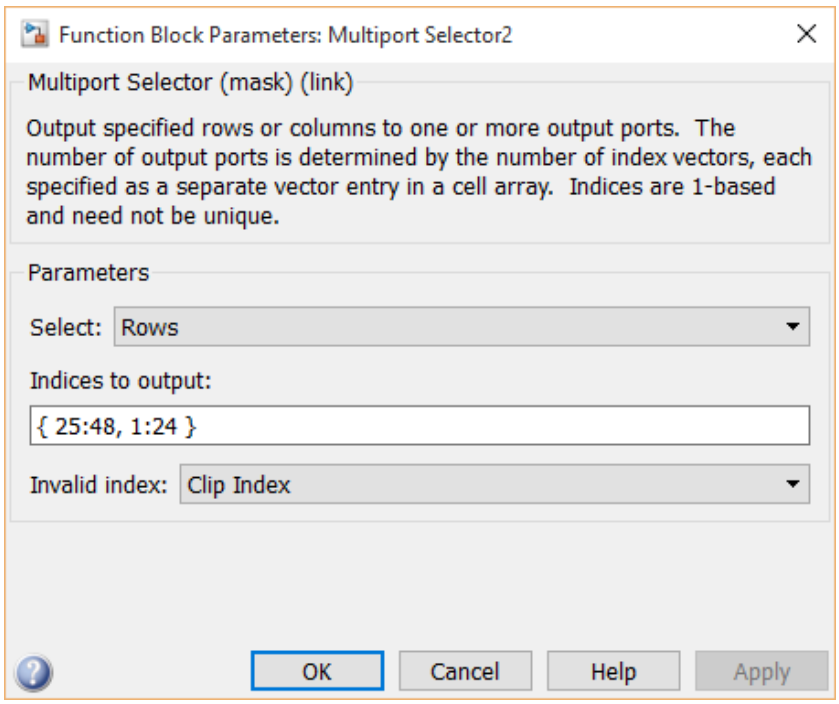


Figure 10.8: ***Multiport Selector***'s configuration window

10. The two stage procedure is convenient, compared with a single stage upsampling by a factor of 20, because it reduces the overall computational burden.

The first stage of the ***Upsampling*** macroblock introduces a 0 between one sample and the other of the input signal. This operation increases the sampling frequency by a factor of 2, changing it from 2400 $\frac{samples}{s}$ to 4800 $\frac{samples}{s}$, without modifying the signal spectrum. The following ***Digital Filter Design 2*** filter has therefore the task to remove one periodic repetition out of two in the signal spectrum, as shown in Fig.10.13.

The signal spectrum at the filter output shows periodic repetitions with period equal to 4800 Hz, as it is appropriate for a signal with a sampling frequency of 4800 $\frac{samples}{s}$.

The next stage, with the block performing the upsampling by a factor of 10 followed by a filter, works according to the same principle: the upsampling stage inserts nine 0s after each input signal sample, increasing the sampling frequency by a factor of 10, and the filter removes 9 periodic spectral repetitions out of 10, so that the first periodic repetition of the spectrum is centred at 48000 $\frac{samples}{s}$.

In order to properly design the filters, set *Fs* at the new sampling frequency, assign *Fpass* the upper limit of the band to be preserved and assign *Fstop* the value of the old sampling frequency minus the band to be preserved.

Fig. 10.14 shows the settings of both filters.

10.2.4 BBtoIF

The ***BBtoIF*** macroblock modulates the signal, translating it from baseband to intermediate frequency. Such operation is performed through a quadrature modulator with internally generated carriers at a frequency $f_{IF} = 15$ kHz.

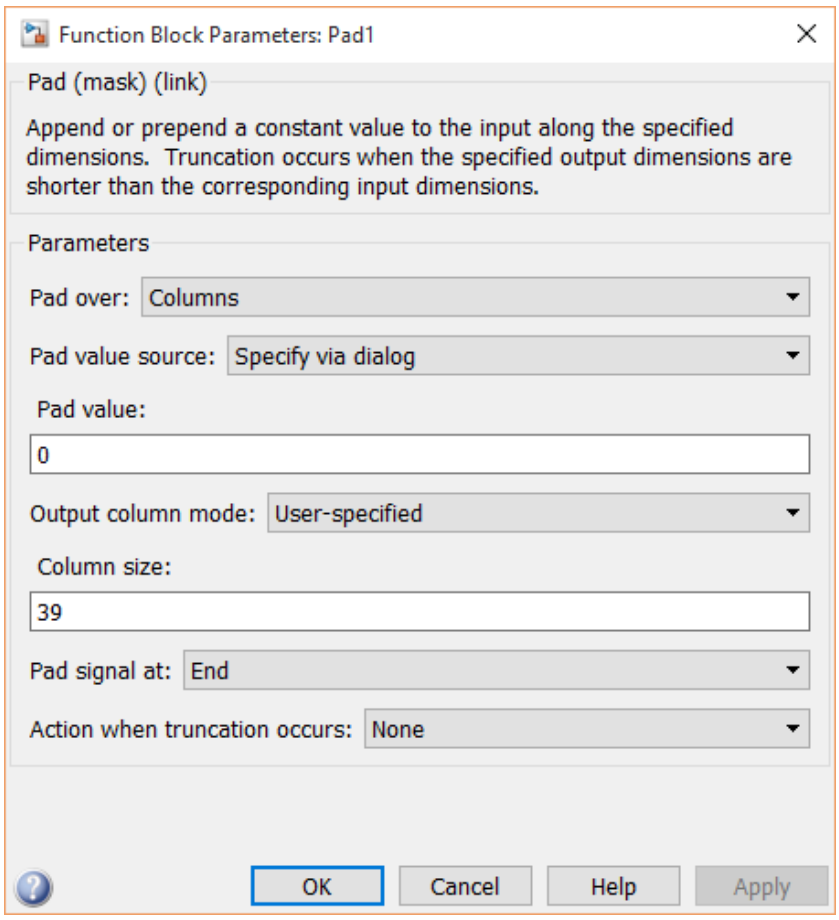


Figure 10.9: ***Pad*** block configuration

The ***BBtoIF*** macroblock is a classic quadrature modulator, with two separate paths for the real (in-phase) and the imaginary (quadrature) components of the signal. Each component is upconverted by a mixer (represented by the ***Product*** block) driven by a cosine or a sine carrier, generated by the ***Cosine Wave*** and ***Sine Wave*** blocks. Both modulated signals are then summed up and taken out.

We conventionally use $cos(2\pi f_{IF}t)$ as carrier for the in-phase signal and $-sin(2\pi f_{IF}t)$ as carrier for the quadrature signal.

Fig.10.15 shows the settings for the carriers' generating blocks. In particular, the $-sin(2\pi ft)$ signal is generated introducing a π phase offset to the $sin(2\pi ft)$ signal that would be generated by default, while the $cos(2\pi ft)$ is generated introducing a $\frac{\pi}{2}$ phase offset.

10.2.5 Max-Divide-Gain (Automatic Gain Control)

The ***Max-Divide-Gain*** blocks, realizing the automatic-gain-control macroblock, are described in Section 6.3.5. The OFDM transmitter here discussed adopts the same configuration shown in Fig.6.10 and in Fig.6.11.

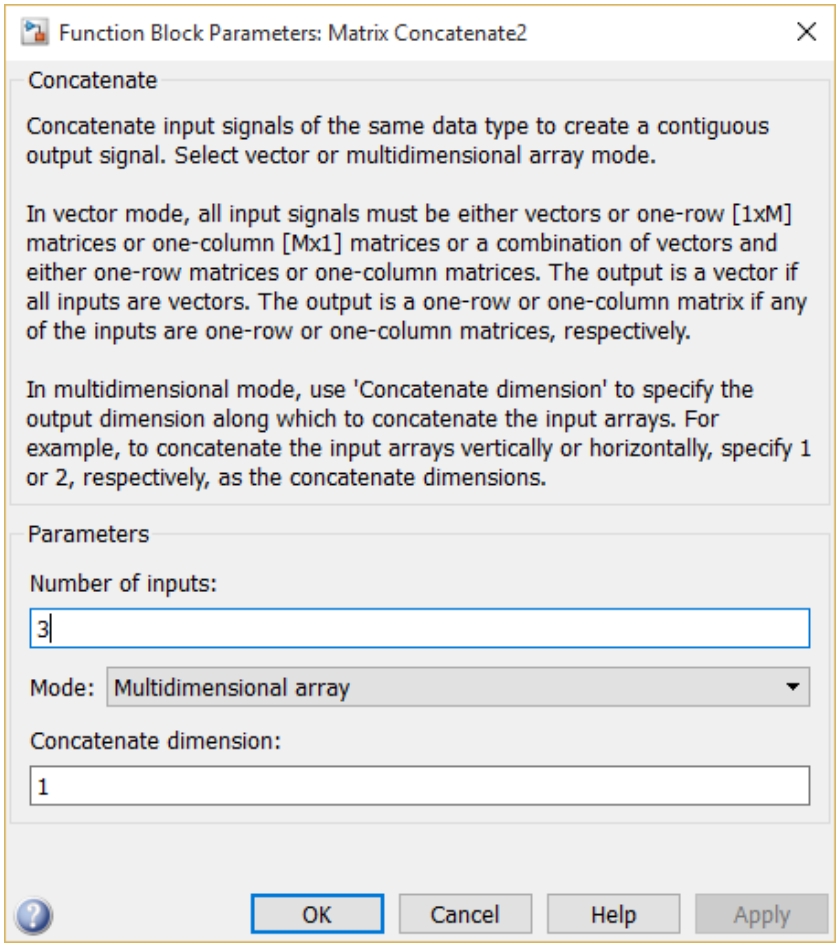


Figure 10.10: Configuration window of the ***Matrix Concatenate*** block

10.2.6 Raspberry Pi output

The **Raspberry Pi output** macroblock represents the signal output port. The elementary blocks ***Data Type Conversion***, ***Matrix Concatenate*** and ***ALSA Audio Playback*** are described in Sections 4.3.2, 4.3.3 and 4.3.4.

10.2.7 Control LED

The ***Control LED*** macroblock has the only aim to intermittently turn on and off the Raspberry Pi2's led during the model execution. Its functioning is described in Section 4.3.6.

10.3 Elementary blocks used

The list of the elementary blocks used for the project realization is provided hereafter, along with the reference to the sections in which their functioning is described.

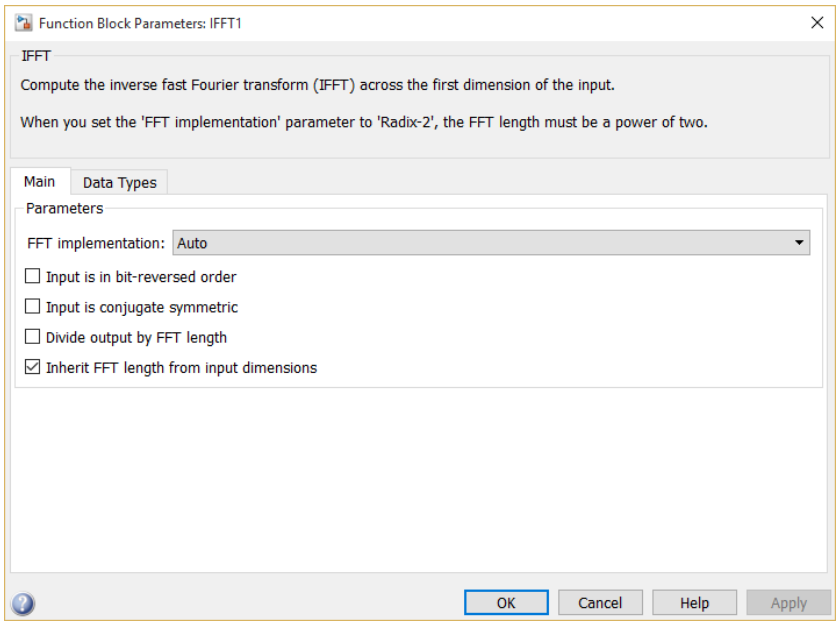


Figure 10.11: Configuration window of the ***IFFT*** block

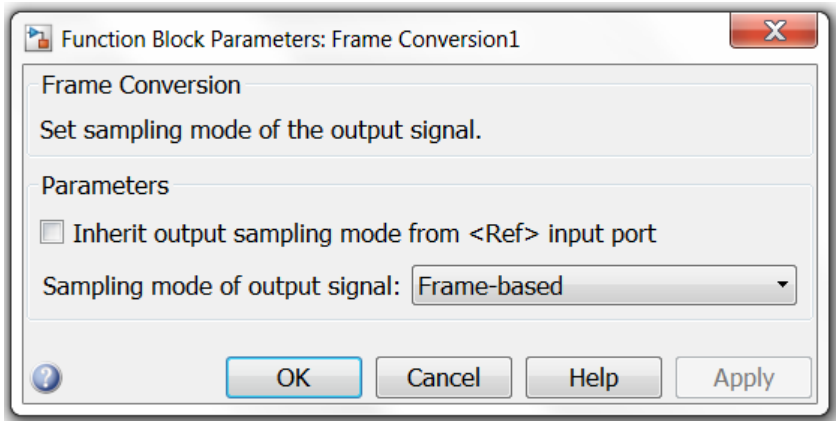


Figure 10.12: Configuration window of the ***To Frame*** block

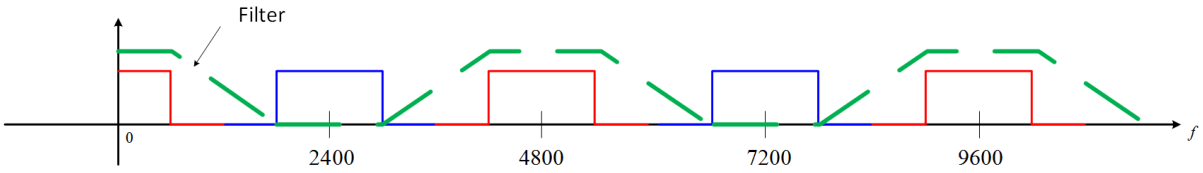
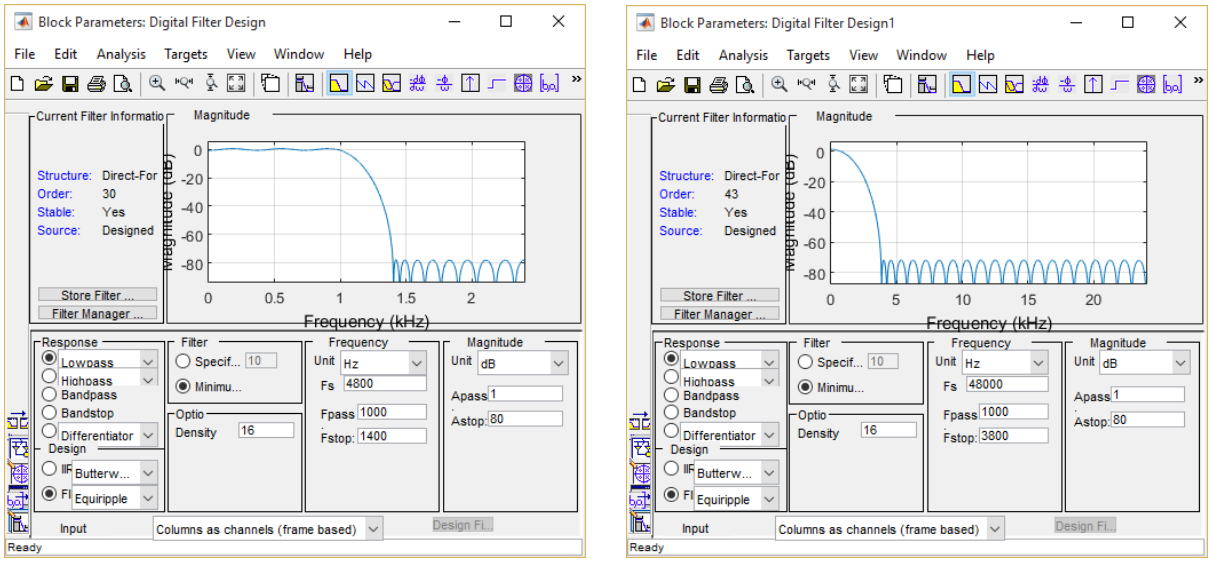


Figure 10.13: Filtering after the upsampling by a factor of 2



(a) Filtro Fir con Fs=4800Hz

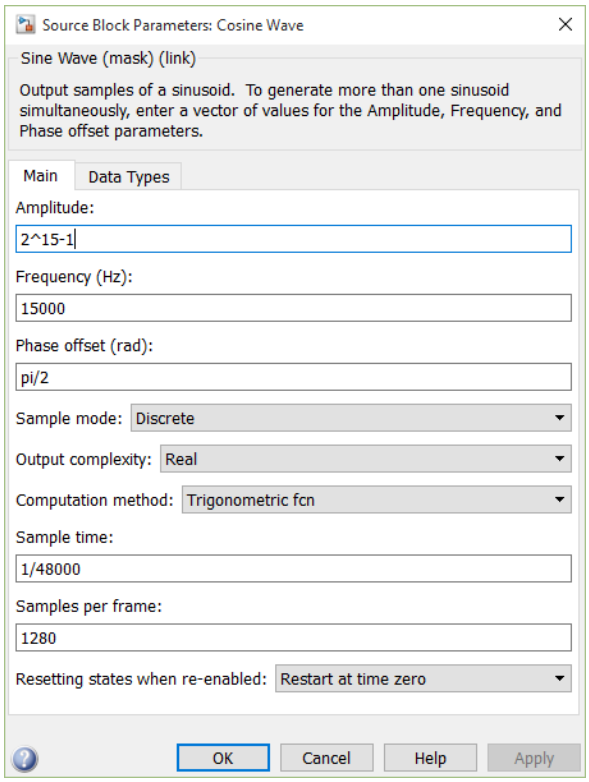
(b) Filtro Fir con Fs=48000Hz

Figure 10.14: Filters'settings

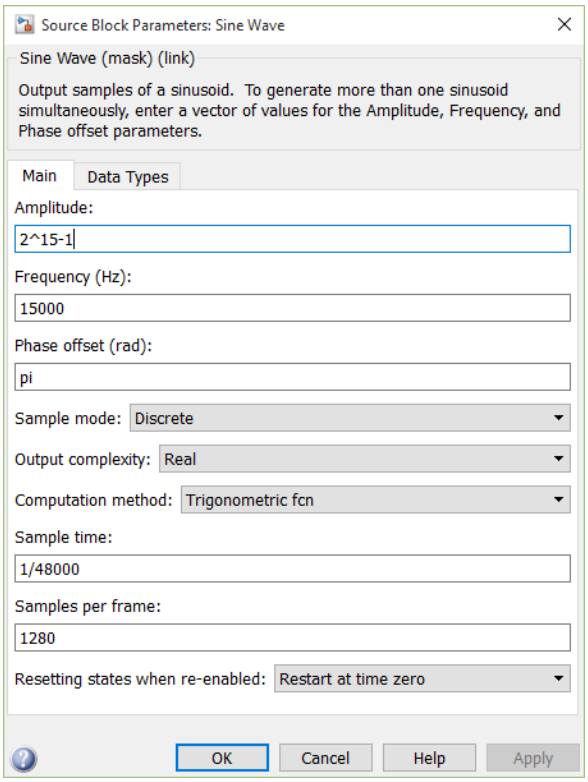
- *Bernoulli Binary Generator* (Section 10.3.1)
- *M-PAM Modulator Baseband* (Section 6.3.2)
- *Complex to Real-Imag* (Section 6.3.3)
- *Multiport Selector* (Section 10.3.4)
- *Pad* (Section 10.3.5)
- *Constant* (Section 10.3.6)
- *Matrix Concatenate* (Section 4.3.3)
- *IFFT* (Section 10.3.8)
- *To Frame* (Section 10.3.9)
- *Upsample* (Section 10.3.10)
- *Digital Filter Design* (Section 10.3.11)
- *Sine Wave/Cosine Wave* (Section 10.3.12)
- *Product* (Section 7.2.3)
- *Add* (Section 9.3.9)
- *Max - Divide - Gain* (Section 6.3.5)
- *Data Type Conversion* (Section 4.3.2)
- *Matrix Concatenate1* (Section 4.3.3)
- *ALSA Audio Playback* (Section 4.3.4)

10.3.1 Bernoully Binary Generator

The ***Bernoully Binary Generator*** block, with the specific configuration for the OFDM transmitter, is described in Section 10.2.1. It represents the binary information source. Its task is to produce a random sequence of independent bits with Bernoulli statistics.



(a) Settings for the $cos(2\pi ft)$ generation



(b) Settings for the $-sin(2\pi ft)$ generation

Figure 10.15: Configuration windows for the ***Cosine Wave*** block and the ***Sine Wave*** block

10.3.2 M-PAM Modulator Baseband

The ***M-PAM Modulator Baseband*** block is described in Section 6.3.2. It converts the input bits $\in$ {0, 1} into 2-PAM symbols $\in$ {-1, 1}.

For the current project this block adopts the same configuration shown in Fig.6.5.

As recalled in Section 6.3.2, despite the fact that 2-PAM symbols are purely real quantities, the ***M-PAM Modulator Baseband*** block generates each of them in the complex format, associating to each symbol an imaginary component with null value (es. $[..., 1+i0, -1+i0, ...]$). A ***Complex to Real-Imag*** block is thus needed in order to remove the imaginary components.

10.3.3 Complex to Real-Imag

The ***Complex to Real-Imag*** block is described in Section 6.3.3. Its task is to remove the symbols' imaginary component. This block adopts the same configuration shown in Fig.6.7.

10.3.4 Multiport Selector

The ***Multiport Selector*** block divides the content of the single *frame* a its input into two different output *frame*. Its functioning is described in Section 10.2.2. The corresponding configuration window is shown in Fig.10.8.

10.3.5 Pad

The ***Pad*** block is described in Section 10.2.2: it postpones a sequence of 15 zeroes, corresponding to the 15 virtual carriers, to the symbols in its input frame. The number of added zeroes is the result of the difference between the output *frame* length (39) and the input *frame* length (24). The corresponding configuration window is shown in Fig.10.9.

10.3.6 Constant

The ***Constant*** block generates the output value 0, corresponding to the amplitude of DC carrier. The corresponding configuration window is shown in Fig.10.16.

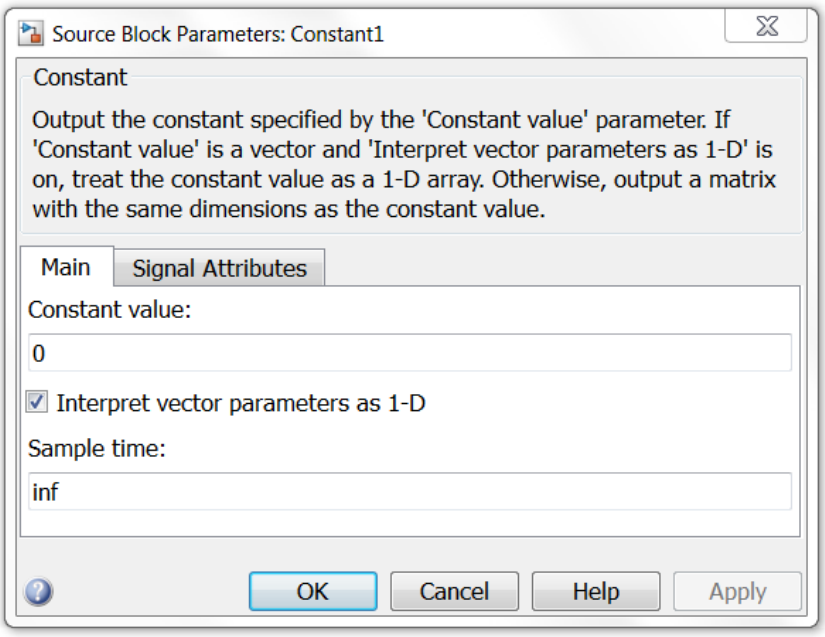


Figure 10.16: Configuration window of the ***Constant*** block

10.3.7 Matrix Concatenate

The ***Matrix Concatenate*** block is discussed in Section 10.2.2. It is used to concatenate in a single output *frame* the three inputs in which the symbols associated to the subcarriers are divided. The corresponding configuration window is shown in Fig.10.10.

10.3.8 IFFT

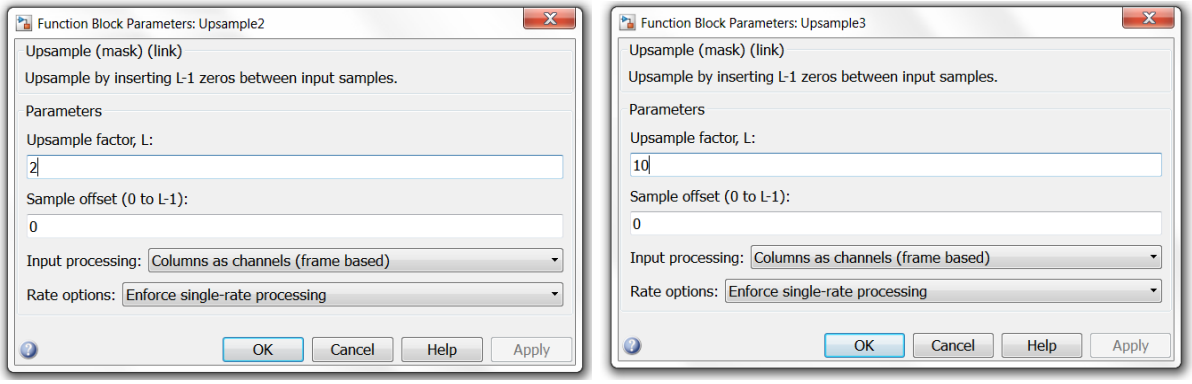
The ***IFFT*** block, described in Section 10.2.2, performs the inverse discrete Fourier transform of its input signal. This block adopts the configuration shown in Fig.10.11.

10.3.9 Frame Conversion

The ***Frame conversion1*** block sets the sampling mode of the output signal. In this case the *frame based* mode is chosen. The corresponding configuration window is shown in Fig.10.12.

10.3.10 Upsample

For a given upsampling factor L, the ***Upsample*** block adds L-1 zeroes after each input value. In this way, the sampling frequency of the output signal increases by a factor of L, without causing any distortion of the spectrum. The configuration windows of the ***Upsample2*** and ***Upsample3*** blocks are shown in Fig.10.17.



(a) ***Upsample2***. (b) ***Upsample3***.

Figure 10.17: Configuration windows

10.3.11 Digital Filter Design

The ***Digital Filter Design*** block, whose functioning is described in Section 10.2.3, represents a filtering stage. Its aim is to remove the undesired spectral components following the ***Upsample*** block. The configuration windows of the ***Digital Filter Design*** and ***Digital Filter Design1*** blocks are shown in Fig.10.14.

10.3.12 Sine Wave/Cosine Wave

The ***Sine Wave*** and ***Cosine Wave*** blocks generate the quadrature carriers for the upconversion of the signal at intermediate frequency. The corresponding configuration windows, similar to those discussed in Section 4.3.1, are shown in Fig.10.15.

10.3.13 Product

The ***Product*** block performs the product of its inputs, whose number is defined by the *Number of inputs* parameter of its configuration window. In this project this parameter is set at 2, as shown in Fig.7.5. The two ***Product1*** and ***Product2*** blocks act as *mixers*, performing product modulations.

10.3.14 Add

The ***Add*** block performs the sum or the difference of its inputs. In this project the block must sum up two inputs, that's why the *List of signs* field of its configuration window is set as "++", as shown in Fig.9.8. The block performs, therefore, the sum of the two signals.

10.3.15 Max-Divide-Gain (Automatic Gain Control)

The ***Max-Divide-Gain*** block, composing the automatic-gain-control macroblock, are described in Section 6.3.5. These blocks adopt the same configurations shown in Fig.6.10 and in Fig.6.11.

10.3.16 Data Type Conversion

The ***Data Type Conversion*** block is described in Section 4.3.2. It converts an input signal of any Simulink data type to the data type specified in its configuration window. This block adopts the same configurations shown in Fig.4.8.

10.3.17 Matrix Concatenate

The ***Matrix Concatenate*** block is described in Section 4.3.3. It is used to produce a two-channel output signal (that is, a stereo signal), starting from the two mono signals at its input. This operation is needed as the ***ALSA Audio Playback*** block, that follows the ***Matrix Concatenate*** block, requires a stereo input signal.

This block adopts the same configuration shown in Fig.4.9.

10.3.18 ALSA Audio Playback

The ***ALSA Audio Playback*** block is described in Section 4.3.4. It represents the Raspberry Pi2's analog output. Its task is to perform the digital-to-analog conversion of the signal and send it to the sound card for playback.

This block adopts the same configuration shown in Fig.4.10.

10.3.19 Implementation and test of the OFDM transmitter

Once the Simulink model is realized and checked through Simulink simulations, it is possible to carry out the *Deploy to Hardware*, as described in Section 4.4. The implemented system starts as soon as the download of the corresponding files on the device is completed.

Connecting the Raspberry Pi2 to the spectrum analyser, according to the scheme in Fig.10.2, the signal spectrum should appear as shown in Fig.10.18.

Figure 10.18: OFDM signal spectrum