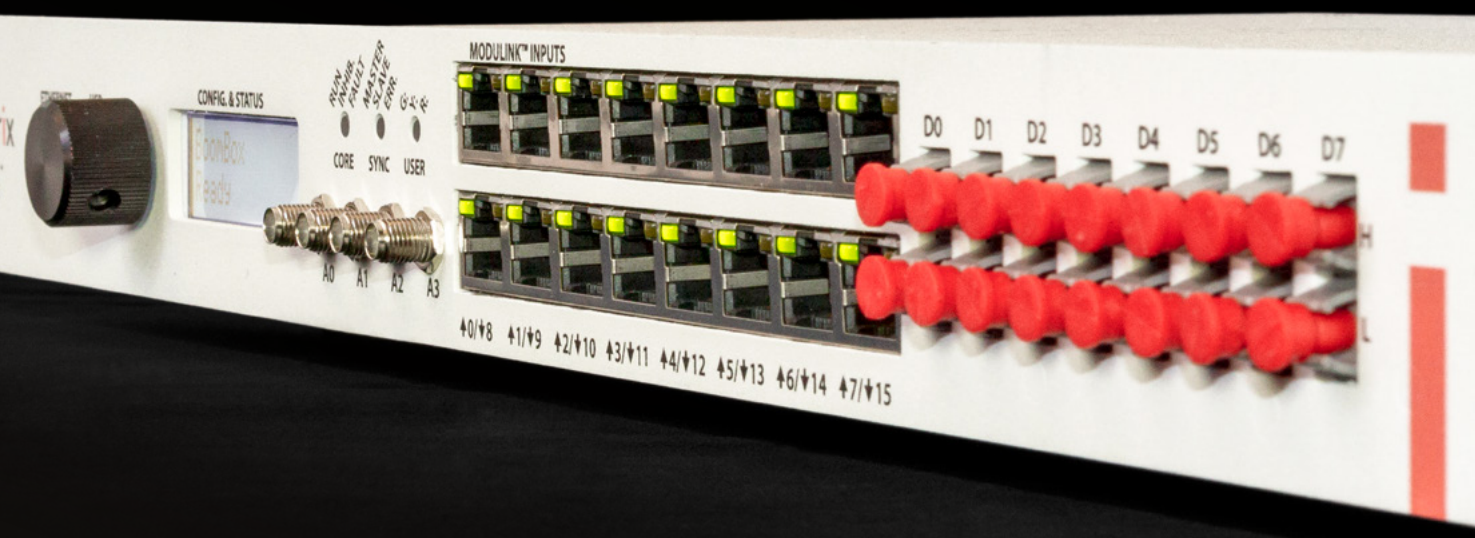
imperix
reshaping energy

BOOMBOX
USER MANUAL



Contact

imperix ltd.
Rue de la Dixence 10
1950 Sion

phone: +41 (0)27 552 06 60
fax: +41 (0)27 552 06 69

www.imperix.ch
support@imperix.ch

Note

Limited warranty

imperix ltd. (hereafter "imperix") warrants to the original purchaser or ultimate customer (hereafter "Customer") of the present product (hereafter "Product") that if any part proves to be defective in material or workmanship within two years, such defective part will be repaired or replaced, free of charge, at imperix's discretion, if shipped prepaid to imperix, Rue de la Dixence 10, CH-1950 Sion, in a package equal to or in the original container. The Product will be returned freight prepaid and repaired or replaced if it is determined by imperix that the part failed due to defective materials or workmanship. Otherwise, the fees will be charged to the Customer. The repair or replacement of any defective part shall be imperix's sole and exclusive responsibility and liability under this limited warranty.

The Customer must contact imperix's customer support team and obtain a Return Authorization Number prior to shipping any Product to imperix. The relevant contact information can be found online on the imperix website (www.imperix.ch).

If the Product is returned for repair more than 24 months after purchase, the Customer is responsible for the cost of repair. imperix will assess the repair and submit a quote to the Customer.

Return policy

The following fees will be applied when the Customer returns a Product for credit:

A full credit, less a 15% fee will be issued if the product is in perfect working condition and returned within 1 month following the shipping date. If repairs are required on the returned product, their cost will be deducted from the credit.

Warranty limitation and exclusion

imperix ltd. will have no further obligation under this limited warranty. All warranty obligations of imperix are void, in particular, but not limited to, if:

- The Product has been subject to abuse, misuse, negligence or accident;
- Repairs or changes have been done to the Product by any other person than by imperix or an authorized technician;
- The original Product identification (trademark, serial number) markings have been defaced, altered, or removed;
- The Customer fails to perform any of the duties set forth in this limited warranty or if the Product has not been operated in accordance with instructions.

Disclaimer of unstated warranties

The present warranty is the only warranty applicable to the Product. Other warranties, express or implied, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose are hereby disclaimed.

Limitation of liability

It is agreed that imperix's liability, whether in contract, in tort, under any warranty, in negligence or otherwise shall not exceed the purchase price payed by the Customer for the said Product and under no circumstances shall imperix be liable for special, indirect, or consequential damages. No action, regardless of form, arising out of the transactions under this warranty may be brought by the purchaser more than one year after the occurrence of the cause of actions.

The legacy from several decades of experience in academic teaching and research at EPFL has given the BoomBox a strong influence that makes this control platform rather different from most existing rapid prototyping systems. The BoomBox control platform reuses some of the key features of its academic predecessor, but goes much further by providing improved flexibility, I/O capability, simplicity and reliability.

Hence, the BoomBox is not only an advanced control platform, but is also, from a software point of view, a strongly layered, yet essentially open system that customers are welcome to self-appropriate and edit up to their own needs, while guaranteeing a safe and sound implementation. As a result, the BoomBox is the ultimate solution for its users to get actively involved in the control design of innovative power electronic systems, and to allow the fast implementation of high-end power converter prototypes.

Thanks to the same will to provide customers with the maximum possible flexibility and openness, numerous advanced features are already available in hardware and will be enabled in future software releases. By doing so, the flexibility of the BoomBox will be even greater, allowing to temporarily or permanently enhance its feature set within minutes depending on the varying needs.

THE BOOMBOX

Abstract — This chapter is an introduction to the BoomBox control platform. It provides a basic overview of the capabilities of the instrument. It is recommended to read this chapter first, as it introduces the terminology used in the rest of this document.

Keywords — *Control scheme, Regulator, Interfaces, Safety, Versatility, Modularity*

1.1 INTRODUCTION

The BoomBox is a new kind of modular control platform, which is tailored for the development of power electronic systems in R&D environments. When compared to existing Rapid Prototyping Systems (RPS), the presented system features similar flexibility, but superior performance and usability at lower cost.

The BoomBox clearly distinguishes itself by its extensive signal conditioning, specifically tailored for power electronic applications. When compared to most general-purpose control systems, faster implementation time and better signal integrity are guaranteed.

With a complete set of 100% tested software libraries, a sturdy 19" rackmount enclosure and several years of experience spent in the hands of users, the BoomBox control platform is surely the ultimate tool for the development of power electronic systems.

1.2 BLOCK DIAGRAM

In the block diagram of a typical control scheme, the only parts that are left up to the user to develop are the application-specific circuit and the accompanying regulation. All the other components of the control loop are embedded in the BoomBox, helping the user to quickly develop a safe working prototype of the intended application circuit.

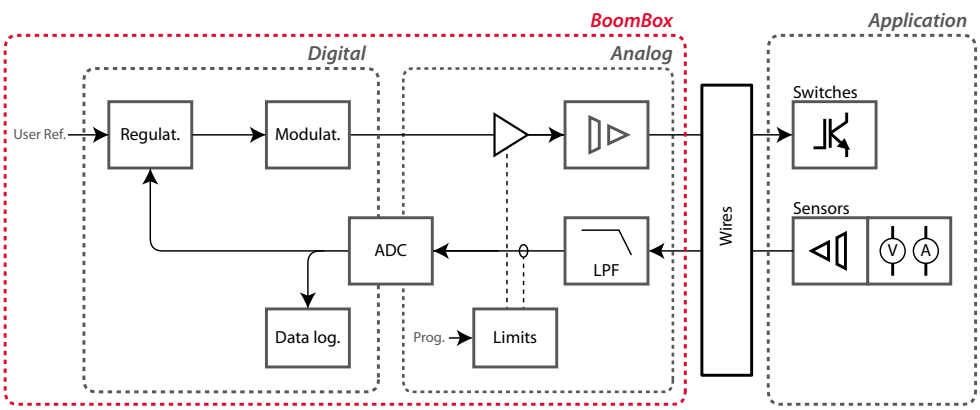


Fig. 1. Block diagram of the control scheme using the BoomBox.

1.3 CAPABILITIES

1.3.1 INTERFACES

In power electronic applications, the users are often forced to create application-specific interfaces between the control and power parts in order to guarantee proper signal integrity and galvanic isolation.

The BoomBox relies on two key concepts in order to speed up the development without any trade-off on safety : isolated fiber optic PWM outputs and versatile analog frontends that fit in with almost any sensor.

Additionally, a variety of digital communication interfaces allow the integration of BoomBox-controlled systems in a wide variety of industrial and development environments.

1.3.2 SAFETY

One of the key features of the BoomBox is the ability to block the entire application in case of inappropriate operation, which may be caused by an improper behavior of the controller, an unexpected event, a damaged device, etc. In such a way, the BoomBox guarantees the physical integrity of both the user and the application at all times.

In any case, the user can set specific limits that will be used to block all gating signals when an overvalue is detected. The crucial point in this protection mechanism is that it is completely *software-independent*, meaning that it is always operational, whatever faulty behavior the DSP or the FPGA may start having. Fig. 2 shows a typical oscillogram of this safety mechanism :

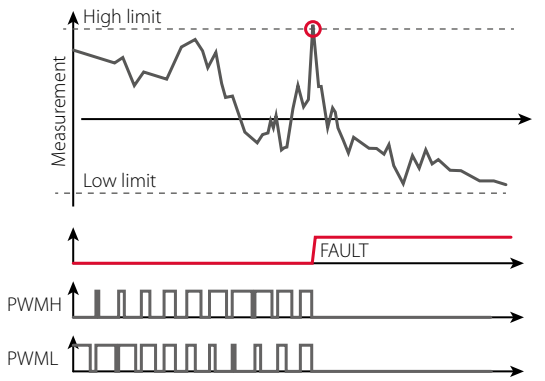


Fig. 2. **Operating principle of the fault detection mechanism and the protective measures.**

Whenever a value larger than the upper limit (or smaller than the lower limit) is detected, a fault flag is raised and latched, blocking all firing signals to their inactive state. The error remains flagged as long as the user doesn't acknowledge the fault.

1.4 CONTROLS AND CONNECTORS

1.4.1 FRONT PANEL

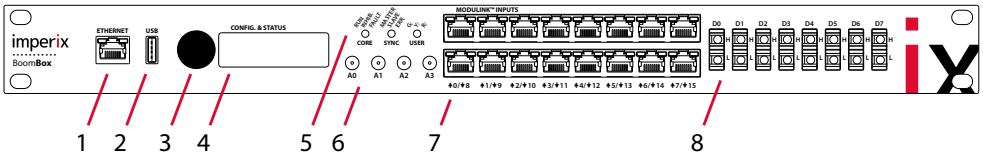


Fig. 3. Front panel view of the BoomBox.

The front panel is composed of the following elements :

1) Ethernet port
2) USB type B device port
3) Rotary and push button
4) LCD screen
5) System and user LEDs
6) SMA analog outputs
7) ModuLink analog inputs
8) Digital fiber-optic PWM outputs

1.4.2 BACK PANEL

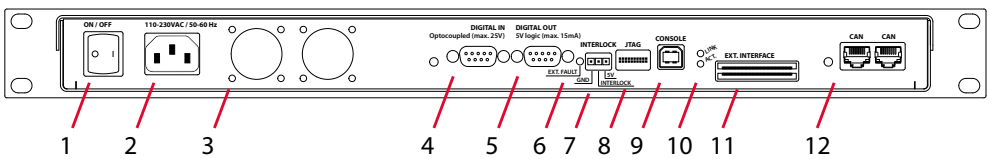


Fig. 4. Back panel view of the BoomBox.

The back panel is composed of the following elements :

1) AC mains switch
2) AC mains socket (IEC 60320 C14)
3) Fan outlets
4) Isolated general-purpose inputs connector
5) Isolated general-purpose outputs connector
6) External fault LED
7) Interlock connector
8) JTAG debugging interface
9) USB type A console port
10) Console communication status LEDs
11) External interface connectors
12) CAN connectors

INPUT / OUTPUT INTERFACES

Abstract — This chapter describes in detail the hardware aspects related to the various analog and digital interfaces of the BoomBox.

Keywords — *Analog input, Analog output, Safety limits, Low-pass filter, Analog stage, Analog gain, Front-panel configuration, Interlock, Fiber-optic, Optical, PWM, GPI, GPO, CAN*

2.1 FEATURES AND CAPABILITIES

Here are the analog interfaces of the BoomBox, along with their capabilities :

- 16 analog *inputs* featuring :
  - Selectable differential high impedance or single-ended low impedance providing compatibility with a broad range of industrial sensors.
  - Programmable analog gain ensuring an optimal use of the ADC full scale.
  - Programmable low-pass filter
  - Programmable software-independent high and low safety limits
  - 200 kHz analog bandwidth, 150 kHz maximum sampling frequency
  - ModuLink connectivity
  - Standard low-cost shielded connector and cabling for best noise immunity
  - Power supply for sensors
- 4 analog *outputs* featuring :
  - 200 kHz analog bandwidth
  - Real-time monitoring and debugging

2.2 ANALOG INPUTS

2.2.1 BASIC PRINCIPLE OF OPERATION

The equivalent schematic of the complete data acquisition chain is depicted in Fig. 5. All channels are stricly identical. Each channel consists of two parts :

- A *hardware part*, which contains the input resistor, a programmable-gain amplifier (PGA), a low-pass filter (LPF) and the analog-to-digital converter (ADC).
- A *software part*, implemented in the DSP, that transforms the digitally-converted measurement into a meaningful quantity that can be easily manipulated : voltage in Volts, temperature in Celcius, pressure in Pascals, etc.). This transformation is done by the driver layer.

The present chapter deals with the *hardware part* and explains how to configure the analog input chain. More information on how to properly use the software part is given in section "6.1 Analog data acquisition system (ADC)", page 41.

2.2.2 BLOCK DIAGRAM

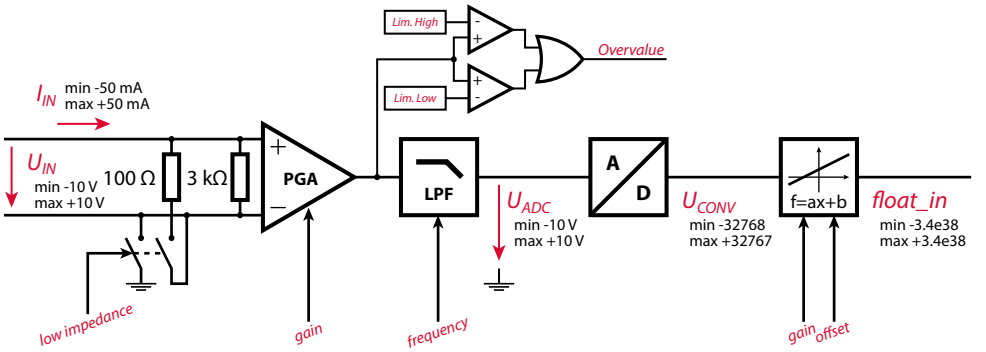


Fig. 5. Block diagram of each of the analog inputs.

2.2.3 ELECTRICAL SPECIFICATIONS

Input impedance	*100 Ω or 3 kΩ (user-configurable)*
Low-pass filter	*none or F_{filt} = [500 Hz .. 50 kHz] (user-configurable)*
Sampling frequency	*0 Hz .. 150 kHz (user-configurable)*
Sampling to IRQ delay	*4 µs, independently of number of channels used*
Resolution	*16 bits*
Measurement range	*± 1.25 V , ± 2.5 V , ± 5 V ,± 10 V (user-configurable)*
Safety	*Independent limits for each channel that trigger emergency shut-down. User-configurable comparison level V_{L+},V_{L-} = [-10.0 V .. 10.0 V]*
Sensor power supply	*± 15 V, 100 mA*
Overvalue detection delay	*< 4 µs (threshold crossing to optical outputs inhibited)*

2.2.4 CONFIGURABLE INPUT IMPEDANCE

Two input configurations are available. A 3 kΩ differential input or a single-ended 100 Ω input. The latter is typically intended to be used with current output sensors, such as LEMs. Each input can be individually configured from the "Analog inputs" menu in the front panel user interface. By choosing "Low impedance : YES", a 100 Ω resistor is switched across the input and the negative input is grounded.

2.2.5 CONFIGURABLE GAIN

For each channel, the hardware gain should be defined such that the voltage U_{ADC} fully exploits the input voltage range of the analog-to-digital converter. It is defined by the relation :

$$U_{ADC} = gain \cdot U_{IN}$$

Where U_{IN} is :

- The voltage provided by the sensor if its output stage behaves as a voltage source.
- $U_{IN} = 100 \cdot I_{IN}$ if the output stage of the sensor behaves as a current source. In this case, I_{IN} is the current provided by the sensor.

The possible hardware gains are shown in Table 1 :

Gain in the pass-band	*1 V/V*	*2 V/V*	*4 V/V*	*8 V/V*

Table 1. Possible hardware gains that can be programmed in the PGA stage of each analog input.

2.2.6 CONFIGURABLE LOW-PASS FILTER

A fifth-order programmable filter can be activated on each analog input channel. The cut-off frequencies of this filter can be set individually and independently for each input channel. A given frequency is considered to be in the stop band if it is larger than 8 times the selected cut-off frequency. The frequency response of the low-pass filter is given in Fig. 6.

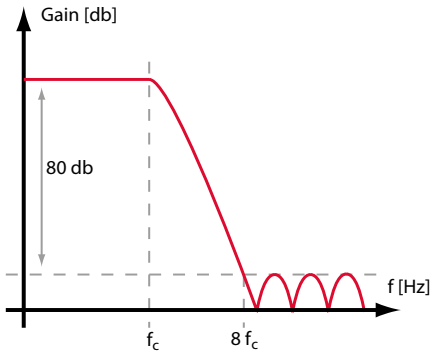


Fig. 6. Frequency response of the low-pass filter.

The cut-off frequencies can be freely adjusted among the values given in Table 2 :

0.5 kHz	*1 kHz*	*1.6 kHz*	*2.5 kHz*	*4 kHz*	*6.4 kHz*	*8 kHz*	*10 kHz*	*16 kHz*	*20 kHz*	*32 kHz*	*40 kHz*

Table 2. Possible cut-off frequencies that can be programmed in the LPF stage of each analog input.

2.2.7 CONFIGURABLE SAFETY LIMITS

The user can define two safety limits for each input channel : a high and a low one. If any of these limits is exceeded while the application is running, a hardware fault flag is triggered, leading to the blocking of the entire application. In order to configure this protective feature, two parameters must be configured for each input channel :

» Limit high defines the highest allowed value for U_{ADC}
» Limit low defines the lowest allowed value for U_{ADC}

These values can be freely set from -10.0 V to 10.0 V, by steps of 100 mV. The comparison voltage is the output of the programmable gain. This means the configuration of the PGA must be taken into account when setting the safety limits.

When a given limit is met or exceeded, the orange LED of the corresponding channel lights up and the CORE LED on the front panel turns red. The user can then access the fault list by selecting the “List hardware faults” option in the “Faults” menu of the front panel user interface. The fault can be acknowledged by selecting the “Acknowledge input faults” option.

Caution

Before acknowledging the fault, make sure that the application has returned to a safe state.

2.2.8 SAVING AND RESTORING FRONT-END CONFIGURATIONS

This feature is easily accessible from the frontpanel. It allows to save and restore the complete configuration of the analog inputs, to and from a USB key. This way, multiple users can concurrently use the same BoomBox, while having different personal configurations.

» When selecting the “Backup config.” option in the frontpanel user interface, the current frontpanel configuration is saved in the folder called “imperix” on the USB key. The filename format is “frontpanel#.bbox”, where # is a number that gets incremented each time a new configuration is saved.
» When selecting the “Restore config.” option, the last “frontpanel#.bbox” file is read and the parameters it contains are applied to the frontpanel configuration.
» When selecting the “Reset config.” option, only the current frontpanel configuration is reset, without touching any of the files on the USB key.

2.2.9 GETTING ADC MEASUREMENTS

Getting ADC measurements is only a matter of calling the correct function from the DSP code. More information on the usage of the Application Programming Interface (API) can be found in section “6.1 Analog data acquisition system (ADC)”, page 41.

2.2.10 ANALOG INPUT CONNECTOR PINOUT AND CABLE

The analog input connector is a standard low-cost 8P8C (8 positions, 8 contacts) shielded modular connector (RJ45). The recommended cabling to be used with this connector is cat. 5E 8-conductor twisted pair shielded cable. The following table gives the pin/pair assignment. The color standard used for the wiring pattern is the same as the one defined by the widespread industry standard T568B.

Pin	Pair	Wire	Color	Description
1	*2*	*1*	*orange stripe*	*+15 V*
2	*2*	*2*	*orange solid*	*+15 V*
3	*3*	*1*	*green stripe*	*0 V*
4	*1*	*2*	*blue solid*	*Positive input / current input*
5	*1*	*1*	*blue stripe*	*Negative input / ground*
6	*3*	*2*	*green solid*	*0 V*
7	*4*	*1*	*brown stripe*	*-15 V*
8	*4*	*2*	*brown solid*	*-15 V*

Table 3. Analog input cable pin/pair assignments (right to left when facing BoomBox front panel).

2.2.11 USAGE EXAMPLE

The ± 15 V supplied through the analog input cable enables the BoomBox to power sensors such as LEM sensors up to a maximum continuous current of 100 mA.

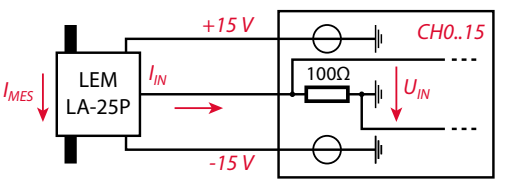


Fig. 7. Block diagram of the connection of a LEM LA-25P current sensor to a ModuLink analog input.

2.2.12 ABSOLUTE MAXIMUM RATINGS

3 kΩ differential input	*Max. input common-mode voltage*	*± 10 V*
	Max. input differential voltage	*± 10 V p-p*
100 Ω single-ended input	*Max. input current*	*50 mA RMS*
± 15 V sensor supply	*Max. output current*	*100 mA (short-circuit protected)*

2.3 ANALOG OUTPUTS

The four analog outputs available on the front panel (SMA connectors) provide a tool for real-time monitoring and debugging from the DSP application code.

Note:

In order to directly connect the analog outputs to an oscilloscope, an SMA-to-BNC adapter is usually required. Reference 744-1249-ND by Digikey is a possible option.

2.3.1 BLOCK DIAGRAM

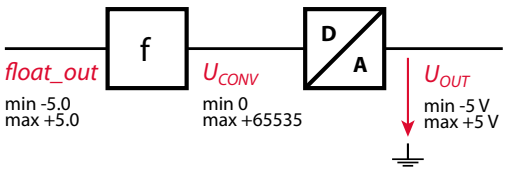


Fig. 8. Block diagram of each of the analog outputs.

2.3.2 ELECTRICAL SPECIFICATIONS

Output impedance	*37 Ω*
Update frequency	*0 Hz .. 100 kHz (user-configurable)*
Resolution	*16 bits*
Output range	*± 5 V*

2.3.3 SETTING THE OUTPUT VOLTAGE

Setting the output voltage is only a matter of calling the correct function from the DSP. The output voltage is then updated in real-time. More information on the usage of the API can be found in section "6.7 Digital to analog converter (DAC)", page 53.

2.4 INTERLOCK

The interlock functionality provides the most basic safety protection, in the form of a life signal. Whenever this signal is not present at the BoomBox input, the fault flag is triggered and the application is safely stopped.

2.4.1 BLOCK DIAGRAM

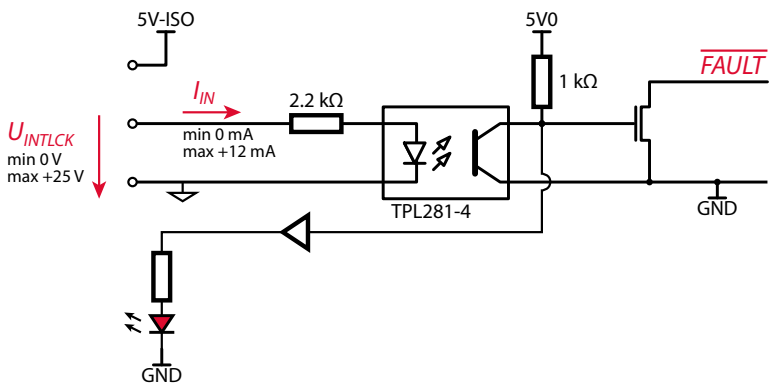


Fig. 9. Block diagram of the interlock input circuit.

2.4.2 ELECTRICAL SPECIFICATIONS

Input impedance	*2.2 kΩ*
High logic level range	*5 V .. 25 V*
Transmission delay	*< 5 µs (interlock rising edge to nFAULT falling edge, I_{DIODE} = 3 mA)*

2.4.3 CONNECTOR PINOUT

The connector used for the interlock line is a 3 position 3.5 mm pin pitch connector from the Weidmüller Omnimate SL-SMT series. An interlock bypass connector is included with the BoomBox, but any application specific circuit can be combined with the interlock functionality to provide an additional layer of safety.

Note :

Any compatible female mating connector can be used, such as Weidmüller nb. 1615680000, available by Digikey 281-1055-ND.

The pinout of the connector must be as shown in Table 4 :

Pin	Description
1	*0 V*
2	*Interlock input*
3	*5 V*

Table 4. Interlock pin assignments (left to right when facing BoomBox back panel).

2.4.4 USAGE EXAMPLE

The interlock input of the BoomBox can be typically used to connect an emergency stop button, as suggested in Fig. 10 :

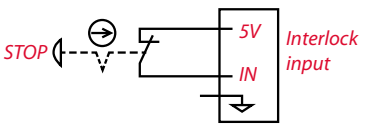


Fig. 10. Block diagram of the connection of a normally-closed emergency switch to the interlock line.

2.5 OPTICAL PWM OUTPUTS

The optical pulse-width modulated (PWM) outputs, along with their modulators implemented in FPGA, are used to drive the gates of the power switches.

2.5.1 BLOCK DIAGRAM

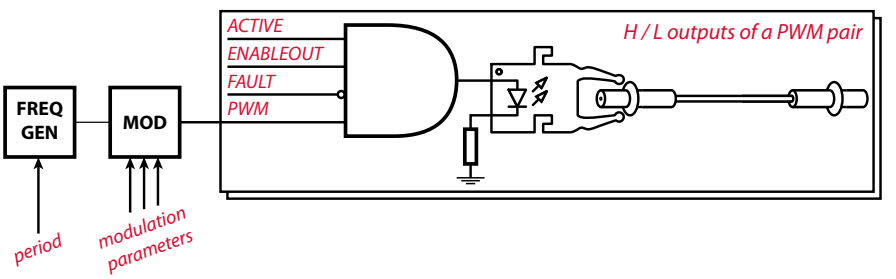


Fig. 11. Block diagram of each pair of fiber-optic PWM outputs.

2.5.2 SAFETY

A number of conditions must be met for a valid modulation to be present at the output of the optical transceiver :

» The channel must be activated by the user code.
» The core of the BoomBox must be in OPERATING state, enabling the outputs.
» No fault must be present.

More information on the BoomBox core and the safety mechanisms can be found in section “Software architecture and operating system”, page 25.

2.5.3 CONFIGURING THE MODULATION

The configuration of the modulation is done by the user in the DSP code. More information on how to use the corresponding API can be found in section “6.2 Pulse-width modulation system (PWM)”, page 43.

2.5.4 OPTICAL SPECIFICATIONS

Wavelength	*650 nm*
Logic type	*Active high*
Temporal resolution	*33.3 ns*

2.5.5 CONNECTOR TYPE

The transceiver used in the BoomBox is the industry-standard HFBR transceiver from Avago (manufacturer part num. HFBR-1528Z). This transceiver includes the Avago Versatile Link connector for the fiber. Instructions to properly crimp the correct type of plastic optical fiber (POF) to the plastic fiber connector can be found in Avago application note 1035. The recommended receiver is the Avago part num. HFBR-2528Z.

2.6 GENERAL-PURPOSE INPUTS (GPI)

The general-purpose inputs enable the user to combine any external digital hardware component with a system controlled using a BoomBox. They also serve as inputs for the incremental encoder interface (see pinout in table below).

2.6.1 BLOCK DIAGRAM

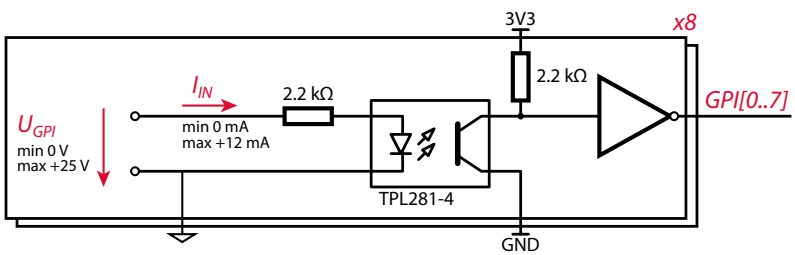


Fig. 12. **Block diagram of general-purpose inputs. This shows the optional galvanic isolation.**

2.6.2 ELECTRICAL SPECIFICATIONS

Input impedance	*2.2 kΩ*
High logic level range	*5 V .. 25 V*

2.6.3 CONNECTOR PINOUT

The connector used is a standard DE-9 shielded connector with the following pinout :

Pin	Description	Incremental Encoder
1	GPI0	A1
2	GPI1	B1
3	GPI2	Z1
4	GPI3	
5	GPI4	

Pin	Description	Incremental Encoder
6	GPI5	A2 or $\overline{A1}$
7	GPI6	B2 or $\overline{B1}$
8	GPI7	Z2 or $\overline{Z1}$
9	0V	

Table 5. GPI pin assignments.

2.7 GENERAL-PURPOSE OUTPUTS (GPO)

The general-purpose outputs enable the user to combine any external digital hardware component with a system controlled using a BoomBox.

Note :

These outputs are intended to control low-speed peripherals and are not suited for modulation output purposes.

2.7.1 BLOCK DIAGRAM

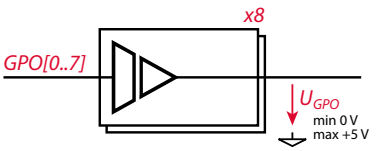


Fig. 13. Block diagram of general-purpose outputs.

2.7.2 ELECTRICAL SPECIFICATIONS

High logic level	5 V
Max. output current	15 mA

2.7.3 CONNECTOR PINOUT

The connector used is a standard DE-9 connector with the following pinout :

Pin	Description
1	GPO0
2	GPO1
3	GPO2
4	GPO3
5	GPO4

Pin	Description
6	GPO5
7	GPO6
8	GPO7
9	0V

Table 6. GPO pin assignments.

2.8 CONTROLLER AREA NETWORK (CAN)

The Controller Area Network (CAN) interfaces located at the back panel of the BoomBox allow communication with other CAN-enabled devices such as sensors or industrial

controllers as well as other BoomBoxes. Both CAN connectors of the BoomBox are tied together, offering a convenient way to daisy-chain CAN devices.

2.8.1 ELECTRICAL SPECIFICATIONS

Impedance	*120 Ω*
Input high logic level range	*5 V .. 20 V*
Output high logic level	*5 V*
Max. voltage on signal lines	*- 27 V .. + 40 V*

2.8.2 BLOCK DIAGRAM

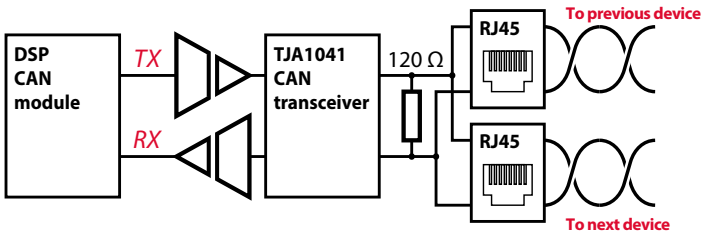


Fig. 14. Block diagram of the Controller Area Network peripheral.

2.8.3 CONNECTOR PINOUT

The connectors used are standard low-cost 8P8C (8 position 8 contact) shielded modular connector (RJ45). The recommended cabling to be used with this connector is cat. 5E 8-conductor twisted pair shielded cable. The following table gives the pin/pair assignment definition. The color standard used for the wiring pattern is the same as the one defined by the widespread industry standard T568B.

Pin	Pair	Wire	Color	Description
1	*2*	*1*	*orange stripe*	*CANH*
2	*2*	*2*	*orange solid*	*CANL*
3	*3*	*1*	*green stripe*	*0 V*
4	*1*	*2*	*blue solid*	*NC*
5	*1*	*1*	*blue stripe*	*NC*
6	*3*	*2*	*green solid*	*0 V*
7	*4*	*1*	*brown stripe*	*NC*
8	*4*	*2*	*brown solid*	*NC*

Table 7. CAN pin/pair assignments (left to right when facing BoomBox back panel).

2.9 ETHERNET

This feature is coming soon.

PROGRAMMING, DEBUGGING AND MONITORING INTERFACES

Abstract — This chapter describes the various interfaces that can be used to change the program running on the BoomBox, debug it and monitor its state.

Keywords — *JTAG, USB, Console, Command line, Indicators, LED, Screen, DAC*

3.1 DSP JTAG

The back panel JTAG connector allows the connection of a JTAG emulator for debugging purposes. For example, a Texas Instruments XDS100v2 USB JTAG Emulator can be used along with Texas Instruments Code Composer Studio IDE to inspect the DSP memory space, check the program flow and debug the code.

3.2 BACK PANEL USB CONSOLE

The back panel USB interface is a serial to USB link which enables the user to interact with the BoomBox using its console interface or the BoomBox Control graphical software (see section 5.3, page 36).

3.3 MONITORING

3.3.1 FRONT AND BACK PANEL INDICATORS

3.3.1.1 USER LED

The state of the **USER** LED can be controlled from the DSP code using the API described in section “6.8 User LED”, page 53.

3.3.1.2 SYSTEM LEDS

The **CORE** LED indicates the status of the BoomBox core :

» Orange means that the BoomBox core is in **BLOCKED** state. No faults are triggered, but the outputs are inhibited and the front panel user interface is available for the user to configure analog input parameters.

» Green means that the BoomBox core is in **OPERATING** state. In this state, a valid modulation is present at the ouput of the activated channels. The front panel is locked because the application is energized and no change of analog input configuration should be made.

» Red means that the BoomBox core is in FAULT state. All outputs have been disabled but the application might still be energized. While applying caution, the user should make sure the fault is not longer present at the BoomBox inputs. This will allow the fault to be acknowledged and the core to go back to BLOCKED state. At this point, the BoomBox can be enabled again to allow the application to be smoothly de-energized.

More information on the BoomBox core and the safety mechanisms can be found in section "Software architecture and operating system", page 25.

The SYNC LED is only used by the expert version of the BoomBox software. It indicates the status of the synchronization between several BoomBoxes. More information on the stacking of several BoomBoxes and the use of the expert software can be found in section "Expert version features", page 58.

3.3.1.3 SCREEN

When a fault occurs, the screen shows the fault type, which can be one of the following :

» Hardware : an event external to the DSP has triggered a fault. This can either be one of the safety limits which has been reached, or an interruption of the interlock line. To know the details of the fault, the user can press the button to access the "List hardware faults" menu.
» Software : the servicing of an interrupt was excessively long or an arithmetical error occured.
» User : a user-level function returned an UNSAFE state.

More information on the BoomBox core and the safety mechanisms can be found in section "Software architecture and operating system", page 25.

3.3.2 DAC INTERFACE

The DAC interface provides an analog output that is asynchronously and continually updated by the FPGA. The user can change the output value by calling the corresponding routine from the DSP code. For more information on the electrical specifications, see section "2.3 Analog outputs", page 17. For more information on the API, see section "6.7 Digital to analog converter (DAC)", page 53.

3.3.3 DATALOGGING USING A USB KEY

This feature is coming soon.

3.3.4 WEB INTERFACE

This feature is coming soon.

SOFTWARE ARCHITECTURE AND OPERATING SYSTEM

Abstract — This chapter describes the software architecture of the BoomBox operating system and its inherent safety mechanisms.

Keywords — *Software architecture, Driver layer, Core layer, User layer, Safety, Interrupts*

4.1 SOFTWARE ARCHITECTURE

The precompiled libraries contained in the base project implement an ultra lightweight Operating System (OS) that provides :

- A *driver layer* composed of easy-to-use routines to manage all peripherals.
- A *core layer* responsible for managing the state of the DSP and the application.
- A *user layer* that is intended to contain all the routines that are specific to the application. This latter layer is *not actually implemented*, but left available to the user to develop his own code. The actual implementation is done in the **user.h/.c** files.

4.1.1 DRIVER LAYER

The driver layer contains several routines that can be divided in the following tasks :

a) Handle the *serial communication* with the computer and provide a command line interface to interact with the user through the PC terminal.

b) Provide low-level routines in order to *configure the FPGA* and communicate with the logical peripherals. This makes the user's life easier by allowing to manipulate meaningful variables and guaranteeing the fast operation of repetitive low-level routines.

c) Provide low-level routines in order to *program the DSP* independently from the XDS100v2 emulator/programmer and boot-up the BoomBox in standalone mode.

4.1.2 CORE LAYER

The core layer is essentially responsible for :

- Initializing the DSP and all its peripherals.
- Detecting all potential fault flags and triggering the safety mechanisms.
- Ensuring the safe execution of the code in the user layer.

The core layer distinguishes three possible states of operation for the DSP :

- **BLOCKED** : The BoomBox is running safely but it is currently blocking the application for safety purposes (the gate signals are inhibited).
- **OPERATING** : The BoomBox is running safely and the application is working as well. The gate signals are sent to the application.

» **FAULT** : A fault flag has been detected and is being processed by the BoomBox. User's attention is required. In the meantime, the application is blocked.

4.1.3 USER LAYER

While the *core layer* focuses on the execution of the DSP, the *user layer* aims to control the execution of the application.

This layer is built on the top of the two others. It is intented to contain the *application-specific part of the DSP code*. It consists of a template of several routines that the user can freely edit and complete.

Chapter 6 "Peripheral drivers" contains more information about the basic principle of operation of the provided driver framework. For more information on how to get started with the editing of the DSP code, please read the "Quick-Start Guide".

4.2 BASIC PRINCIPLES OF OPERATION

When using the BoomBox, the following steps are typically made :

1) At startup, it is initially set in **BLOCKED** state through the **CoreInit()** routine. This routines calls **UserInit()** in order for the user to activate his own initialization procedures. The complete initialization procedure in depicted in Fig. 15.

2) Although interrupts are already serviced, no gate signal is actually produced until the user activates the application by sending an "enable" request in the command-line interface.

3) Once the user executes the "**enable**" command in the terminal console, PWM gate signals are physically generated. Consequently, the core state is changed from **BLOCKED** to **OPERATING** and the **CORE** LED on the front panel turns green, proving that the BoomBox has changed to its normal mode of operation.

4) At any time, by executing the "**disable**" command, the user can block the gate signals. Executing "**enable**" subsequently reactivates the PWM generation as well. Both "**enable**" and "**disable**" commands can be executed at any time and as often as desired.

5) In case of a fault, the BoomBox immediately switches to its **FAULT** mode of operation, which also prevents gate signals to be outputted.

Fig. 16 presents the complete core state machine with its states and transitions :

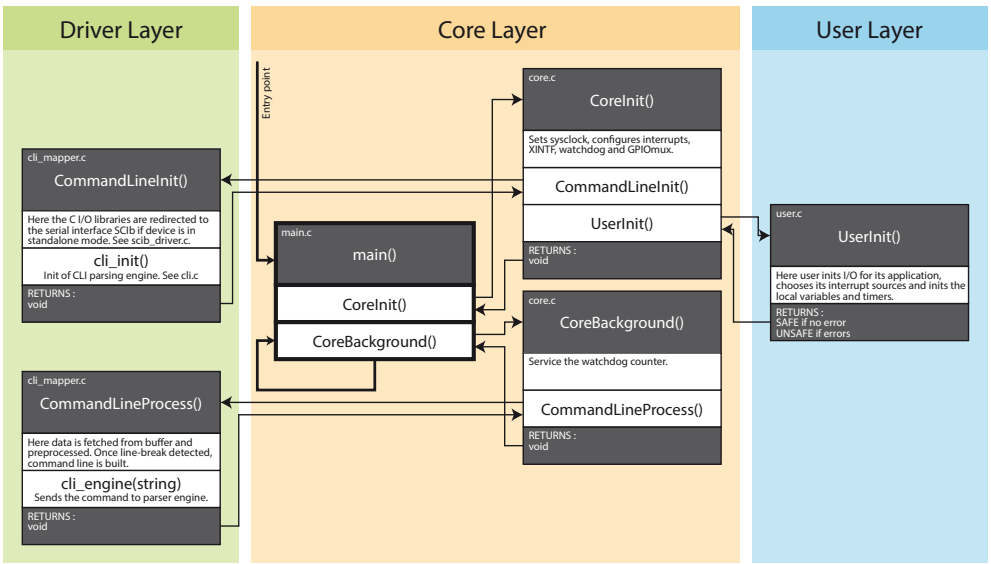


Fig. 15. Initialization process of the BoomBox.

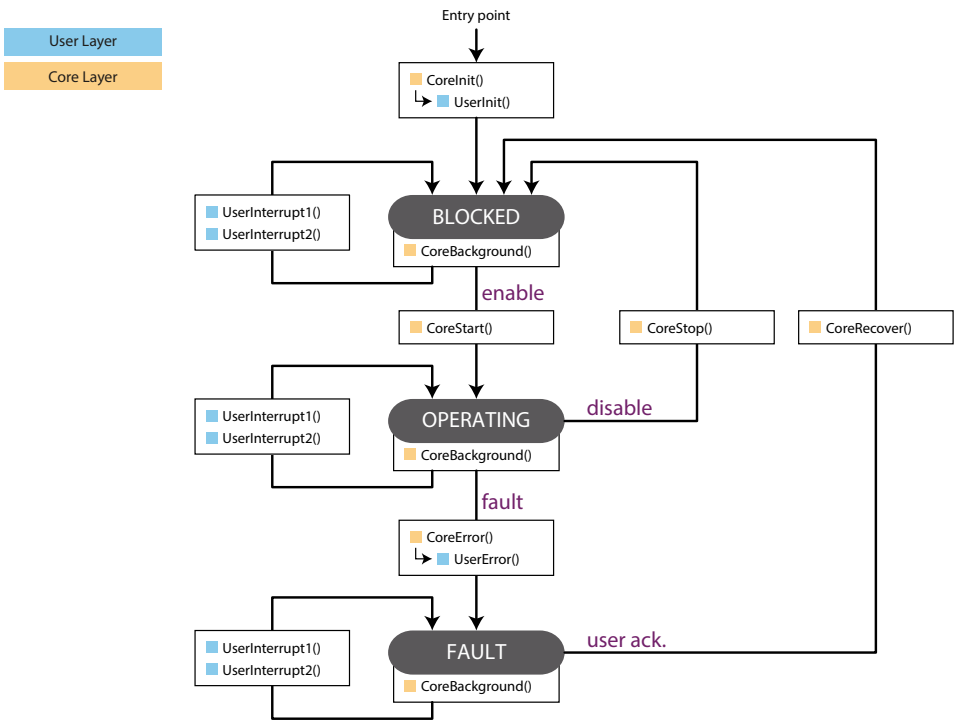


Fig. 16. Finite State Machine of the core layer of the BoomBox.

4.2.1 SAFETY MECHANISMS

In order to protect the user, the application and the BoomBox itself, *software-independent* safety mecanisms are implemented. They lead to the blocking of the entire application in case of overvalues. Besides, in addition to these basic mechanisms, *the DSP* may also trigger the blocking of the application in order to protect it against its own behaviour, when inappropriate.

In consequence, three different error sources can trigger the blocking of the application :

» **Hardware error :** corresponds to having an analog measurement exceeding one of the corresponding thresholds on one of the analog input channels. In such a case, the front panel displays which channel was responsible for activating the fault flag.

» **Software error :** corresponds to a potentially unsafe operation of the DSP. The core layer is responsible for triggering such an error flag. The possible causes are either a division by zero, a stack overflow or an exessive execution time in **UserInterrupt1()** or **UserInterrupt2()**.

» **User error :** corresponds to a request from the user himself to enter into the **FAULT** mode of operation. Such error can be triggered by returning an **UNSAFE** status from any of the user-leve routines. By doing so, the user decides – *on purpose* – that the code execution is becoming unsafe and that the safety mechanisms must be activated.

As shown in Fig. 17, if one of the events described above is triggered, the **CoreError()** routine is called and the core state is changed to **FAULT**. In this state, all PWM modules are blocked, a warning message is displayed in the console, the **CORE** LED on the front panel turns red and the **UserError()** function is called. At the end of the execution of **UserError()**, the error is cleared and the BoomBox returns in the **BLOCKED** state.

Warning :

When this procedure occurs, the application has already been blocked by the IO system of the BoomBox in a software-independant manner. This software procedure only makes the DSP code aware of the fault, so that the application can be safely shut down and/or properly restarted.

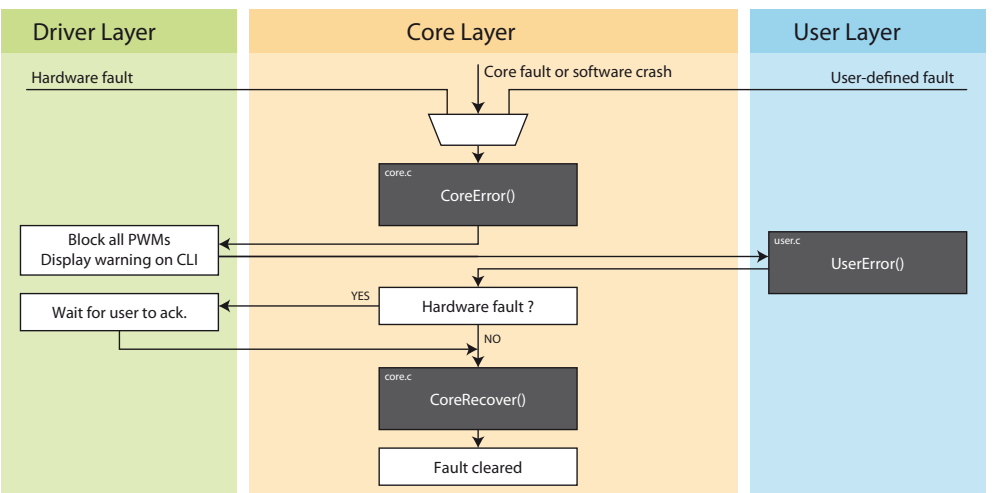


Fig. 17. Flow chart of the error management process.

4.2.2 INTERRUPTS AND SAMPLED OPERATION

As for any digital control system, the BoomBox features a sampled time operation. Therefore, it relies on the use of *interrupts* in order to schedule repetitive calls to control routines.

» *An interrupt* is an event interrupting the processor's execution, forcing it to switch to another task requiring immediate attention. In discrete-time controllers, interrupts are used to trigger the execution of control routines at precise time intervals. Typically, these interrupts are generated either by timers or by hardware events.

» *An Interrupt Service Routine (ISR)* is a software routine executed at the occurrence of an interrupt. In discrete-time controllers, its purpose is to execute the control tasks related to the corresponding interrupt.

» The process of associating an interrupt with an ISR is called "*mapping an interrupt*".

Detailed information on how to properly register and configure the user-level Interrupt Service Routines is given in section "6.6 Interrupt source selection (IRQ)", page 52.

PROGRAMMING AND CONTROL SOFTWARE ON THE PC

Abstract — This chapter describes the use of the PC software to code and program the BoomBox and interact with it once it is running.

Keywords — Simulink, Automated Code Generation, C++, Code Composer Studio, CCS, command line, CLI, Tera Term, BoomBox Control, Automated Code Generation, ACG, Simulink, graphical programming, Simulink toolbox, BoomBox block library

5.1 PROGRAMMING IN C/C++

Coding in C++ is typically done using Texas Instruments' integrated development environment called Code Composer Studio. For information on how to install Code Composer Studio and the required libraries and import a template project, please consult the BoomBox Quick Start Guide.

Documentation on the routines used to access the BoomBox's peripherals can be found in chapter "Peripheral drivers", page 41.

5.2 PROGRAMMING USING SIMULINK

5.2.1 PREREQUISITES

To use the BoomBox ACG Simulink package, the following software is required

- MATLAB r2015a or later, with the following additions
  - Simulink
  - Simulink Coder
  - Embedded Coder

Additionally, to properly open and execute our model examples, the following software may be required:

- PLECS Viewer v. 4 or later. This can be obtained free-of-charge by downloading the installer for the regular PLECS Blockset and executing it without a license.

5.2.2 GETTING STARTED

To start building a model using the Simulink BoomBox ACG package, the installer must be executed first. There, a license file is needed, which can be dowloaded from the customer area on imperix's website:

http://imperix.ch/support/customer-area

If no license is available at the time of the first installation, it remains possible to complete the installation successfully and to execute simulations, but the automated code generation process will not be available.

Note:
In order to freely evaluate the automated code generation process, trial licenses can be requested by simply writing to sales@imperix.ch.

With the package installed, a file called Bbox_template.slx is available in the MATLAB folder in My Documents. This file should provide a starting point for all BoomBox compatible Simulink models.

Note :
When starting a new project, it is absolutely necesarry to start either by the template file or code examples that are available on imperix.ch. Starting from a blank file would not work.

Note :
Several fully-functional examples can be found on imperix's website at the following URL:
http://www.imperix.ch/category/code-examples

5.2.3 MAIN CONCEPTS

The provided template file provides a basic model canvas consisting of a subsystem where to place the simulated model of the converter (Plant_model) and another subsystem to place the model of the controller (Closed_loop_control). The latter contains the only block that is absolutely mandatory for a BoomBox compatible file: the Configuration block:

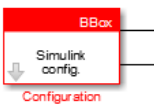


Fig. 18. Configuration block.

By double-clicking on this block, it is possible to set one of two execution modes:

- Simulation: used for simulating the circuit on the computer (off-line)
- Automated Code Generation: used for generating executable code and downloading it onto the BoomBox for real-time execution.

Additionally, this block sets the switching and control frequency, the simulation time step and other timing parameters. Further description of the available options can be found in the following paragraphs.

The Configuration block has two outputs that can be seen as trigger events for signal sampling and modulation. So that timing is correctly provided to the ADC and PWM blocks, their respective clock inputs have to be wired to the corresponding outputs of the Configuration block, as seen in Fig. 20.

Warning:
Attention must be paid to wire the upper signal to the clock input of ADC blocks and the lower signal to the clock input to clock input of PWM blocks. Failure in doing so would lead to incorrect simulation results.

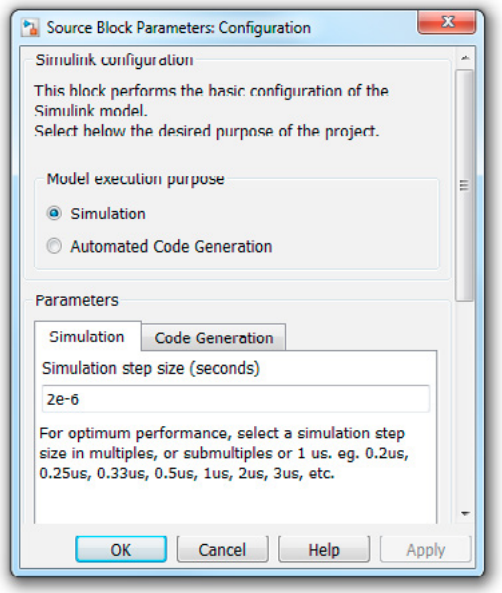


Fig. 19. Selecting the model execution mode in the Configuration block mask.

The following parameters in the **Configuration** block mask have an influence in simulation mode:

- **Simulation step size**: this defines the step size of the simulated signals coming out of the BoomBox **PWM** blocks, and ideally the plant simulation. For better simulation performance, the toolbox thais used used to simulate the power electronics behavior (e.g. PLECS or SimPowerSystems) should be set to an automatic step size or forced to use the **TSAMPLE** variable.
- **Switching frequency**: this defines the carrier frequency used to generate the simulated PWM signals.
- **Relative phase**: phase difference between the ADC and PWM simulation clocks (see more details on how to wire those clocks below).

Fig. 20. Connection of the ADC and PWM clocks necessary to provide timing in simulation mode.

5.2.4 SIMULATION

To simulate, simply set the model in **Simulation** mode as described above and press the **Run** button:

In simulation mode, the blocks placed in the **Plant_model** subsystem are executed to simulate the converter hardware. On the control side, blocks from the BoomBox blockset provide a simulation model of the corresponding BoomBox peripheral. For example,

these blocks are ensuring that the real-world variables coming from the converter simulation are sampled at the correct instant.

Note:

The simulation of a model configured in Automated Code Generation mode would lead to unexpected results.

5.2.5 AUTOMATED CODE GENERATION

To prepare a model to generate code, set the model execution mode in **Automated Code Generation** within the Configuration block properties, as depicted in Fig. 19.

In code generation mode, the **Plant_model** subsystem is ignored and the **Closed_loop_control** is used to generate code and execute it on the BoomBox control platform.

In order to launch the Automated Code Generation process, simply press the **Build-and-run** button:

The process usually takes up to 20 seconds. It performs several actions such as compilation, linking and code upload to the BoomBox control platform. Once the generated code is running in the BoomBox, its behavior can be monitored using the BoomBox Control graphical software. Please refer to section 5.3 for BoomBox Control usage.

Note:

For the code to load, the BoomBox must be powered and connected to the PC using the XDS100 JTAG adapter. Furthermore, to enable the BoomBox, control parameters and monitor the running code using BoomBox Control, the CONSOLE USB connection must be connected as well.

To make a signal available to the user through BoomBox Control, connect it to a **Probe** block. The signal can then be added to the watch list in the Debugging tab by typing its name.

Fig. 21. Probe block.

For parameters that need to be modified on-the-fly, place a **Tunable parameter** block. Similarly to the **Probe** block, it is possible to add it to the watch list and alter its value by double-clicking its name.

Fig. 22. Tunable parameter and Data Store Read blocks.

It is also possible to use its value multiple times in the Simulink model using a **Data Store Read** block with the same variable name (from the standard Simulink library).

5.2.6 COMMON ISSUES USING SIMULINK

5.2.6.1 SIGNAL TYPE

Sometimes, some blocks require a specific data type to operate, which might create some incompatibilities, leading to error messages similar to the following:

Error example:

Data type mismatch. Input port 1 of 'Bbox_template/Closed_loop_control/PWM' expects a signal of data type 'single'. However, it is driven by a signal of data type 'double'.

To solve this issue, use a **Data Type Conversion** block set to output the correct type, in this case, **single**:

Fig. 23. Data Type Conversion block.

5.2.6.2 SAMPLE TIME

Blocks with continuous sample times are not supported, producing errors such as:

Error:

Block 'Bbox_template/Closed_loop_control/Step' uses continuous time, which is not supported with the current configuration.

In other cases, sample time mismatches can happen such as:

Error:

Sample time mismatch. Sample time (2.5e-05) of signal at input port 1 of 'test/Closed_loop_control/PWM1/simulation/Triangle/S//H' does not match the sample time (5e-05) specified for this signal by 'test/Closed_loop_control/PWM1/simulation/Triangle/Signal Specification'. Consider using a Rate Transition block to resolve the mismatch.

In both those cases, it is worth checking the **Sample time** parameter of the blocks in the signal path and try setting them to -1 (inherited). If needed, the sample time can be set to **SWPERIOD**, which corresponds to the switching frequency set in the **Configuration** block.

To check the sample times graphically, press **Ctrl+J** to show the sample time color legend. Essentially, the model of the controller should be uniformly sampled at the switching frequency for the simulation to behave in a similar way as the real-time code.

5.2.6.3 CODE COMPILATION AND SIMULATION ERRORS

Make sure that the model is configured in the mode which corresponds to the wanted action (**Simulation** or **Code Generation**). Configure the model in **Simulation** mode and press **Run**, or set it to **Automated Code Generation** and use the **Build Model** button.

5.2.7 CORRESPONDANCE OF API TO LIBRARY BLOCKS

The table below shows the C++ functions that are used in the BoomBox Simulink library, in which block and in which callback they are called. The callback field can take one of the following values:

» **Start**: this corresponds to calling the function once per instance of the block, at code initialization (C/C++ equivalent to **UserInit()**).

» **Output**: this corresponds to calling the function at each execution of the control period for each instance of the block (C/C++ equivalent to **UserInterrupt1()**).

C++ function	Simulink block	Simulink callback
SetADCAdjustments	*ADC*	*Start*
GetADC	*ADC*	*Outputs*
ConfigPWMChannel	*PWM*	*Start*
SetPWMPhase	*PWM*	*Start*
SetPWMDutyCycle	*PWM*	*Outputs*
ActivatePWMChannel	*PWM*	*Start*
RegisterExt1Interrupt	*Configuration*	*Start*
ConfigSampling	*Configuration*	*Start*
SetFreqGenPeriod	*Configuration*	*Start*
GetGPIbit	*GPI*	*Outputs*
InitAllGPO	*GPO*	*Start*
ForceGPObit	*GPO*	*Outputs*
SetDACVoltage	*DAC*	*Outputs*
SetUserLED	*LED*	*Outputs*
ConfigDECModule	*DEC*	*Start*
GetDECAngle	*DEC*	*Outputs*

Table 8. Correspondance table of C++ API to Simulink block

5.3 BOOMBOX CONTROL

BoomBox Control is a graphical software that enables the user to monitor, control, and display data from the BoomBox control platform. The program runs on a PC connected to a BoomBox through its CONSOLE USB connection. It replaces a traditional terminal software, such as **TeraTerm**, while providing enhanced functionality.

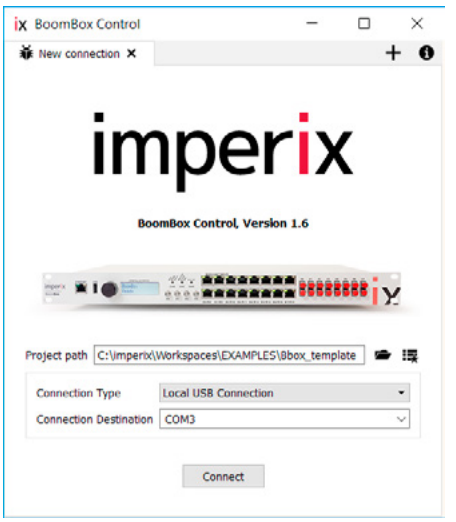


Fig. 24. BoomBox Control connection selection screen.

5.3.1 BASIC PRINCIPLES

When BoomBox Control is launched from the Start Menu, the working project folder must be selected from the disk or a recently used list. The selection of this folder serves to indicate to BoomBox Control in which folder it should look for the generated code files.

Next, a connection to a BoomBox unit must be established. After choosing the COM port which corresponds to the target device, the program parses the generated code (from Simulink or written by hand) to extract the list of variables and their location inside the DSP's memory.

5.3.2 MAIN CONTROLS

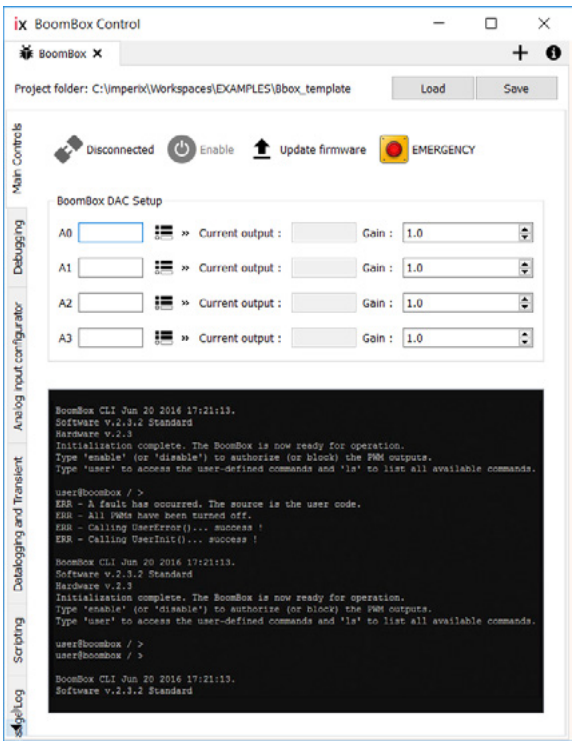


Fig. 25. BoomBox Control Main Controls tab.

5.3.2.1 BOOMBOX CORE STATUS

The current status of the BoomBox can be changed using the **Enable / Disable** button. Likewise, at any time, pressing the **Emergency** button generates a **SOFT** error.

More information on the CORE states can be found in §4.2.

5.3.2.2 UPDATING THE FIRMWARE

Updating the firmware can be done using the dedicated button. The file that must be chosen is the **.a00** file in the **Debug_BoomBox** folder located in the project folder.

Note:

As BoomBox Control is not (yet) a multi-threaded application and the transfer of the firmware used 100% of the communication bandwitdth, the program may apprear to "freeze" to Windows's task manager during transfer. This may take up to 20 seconds. BoomBox Control will nevertheless return to normal operation once the transfer is complete.

5.3.2.3 CONFIGURING THE DAC OUTPUTS

This feature can be used to send any global variable to one of the analog outputs. The update of the outputs is done periodically at main control interrupt frequency.

To use this feature, type the name of a variable in one of the four DAC output fields and press Enter or click on the button immediately on the right of the field. The gain of each output can be adjusted individually to be able to output the intended values using the +/- 5V full scale of the DAC.

More information of the analog ouputs can be found in §2.3.

5.3.2.4 COMMAND LINE INTERFACE

The BoomBox command line interface organises commands in different directories. The user can navigate in this directory structure by typing the directory name to enter a sub-directory, or **..** to go back to the parent directory.

Anywhere in the command line interface, the command <**ls**> shows which subdirectories and commands are available in the current working directory.

To find the custom commands defined in the file **cli_commands.c**, navigate to the /**user** directory. For example, our tutorial n°1 code example defines a custom command called **setipv** to change a controller set point.

Even though these actions can be more conveniently done using the graphical user interface, the two following fundamental commands are still available :

» **enable** : enable the PWM outputs that have been activated to produce a modulated output. Makes the core transition from the **BLOCKED** to the **OPERATING** state.
» **disable** : disable the PWM outputs. Makes the core transition from the **OPERATING** to the **BLOCKED** state.

This command line interface can also be useful to get status updates from the BoomBox, for example regarding the source of a fault.

5.3.3 ANALOG INPUT CONFIGURATOR

This tab enables the user to load existing Analog frontpanel configuration files (.bbox), alter them, or create them from scratch. These files can then be loaded on the BoomBox frontpanel by storing them on a USB drive.

To upload a frontpanel configuration file to the BoomBox, place it on a USB drive in a folder called **imperix**. The filename must be of the form **frontpanel#.bbox**. By choosing "Restore config." from the frontpanel menu, the BoomBox will read the file with the highest number (#) starting from 0.

More information on the use of the frontpanel to save and restore configurations from a USB stick can be found in 2.2.8.

5.3.4 DEBUGGING

This tab provides a way to monitor and alter any global variable during run time. Variables can be added to the watch list by typing their name in the top field and pressing Enter.

Their current values can then be monitored in real-time in the list below. The user can also alter a variable's value by double-clicking on it (see Fig. 26). Additionally, by dragging

a variable from the watch list and dropping it over the plot below, the variable's evolution in time can be viewed.

The maximum update rate of this plot is 10 Hz so it is only suited to slow variables. For fast phenomena, please use the datalogging feature described below.

Fig. 26. Altering a variable using the BoomBox Control Debugging tab.

5.3.5 DATALOGGING AND GENERATION OF TRANSIENTS

5.3.5.1 DATALOGGING

The datalogging can be used to log up to six variables of type float. These are sampled periodically at main control interrupt frequency, similarly to the analog outputs.

After configuring the variables and the window length (max. 2048 samples per variable), the datalogging module can be enabled by pressing the **Run/Stop** button. This checks the variable list and locks the configuration. The trigger can then be fired manually by pressing the **Force trigger** button or configured using the associated section the same way as a physical oscilloscope.

If data is available in the BoomBox's buffer, the **Save data** button becomes active (see Fig. 27) and the data can be saved to disk in comma-separated value format (.csv). Optionally, if MATLAB is installed, it is possible to directly launch it, import and plot the data.

The MATLAB function file used to import and plot the data can be found in the BoomBox Control install folder (**boomboximport.m**).

5.3.5.2 TRANSIENT GENERATOR

When enabled, this feature can be used to produce up to 6 step events on any global variable defined in the code. The steps are applied at the specified sample instant each time a datalogging event is triggered, either manually by the user, or automatically depending on the configuration of the datalogging module.

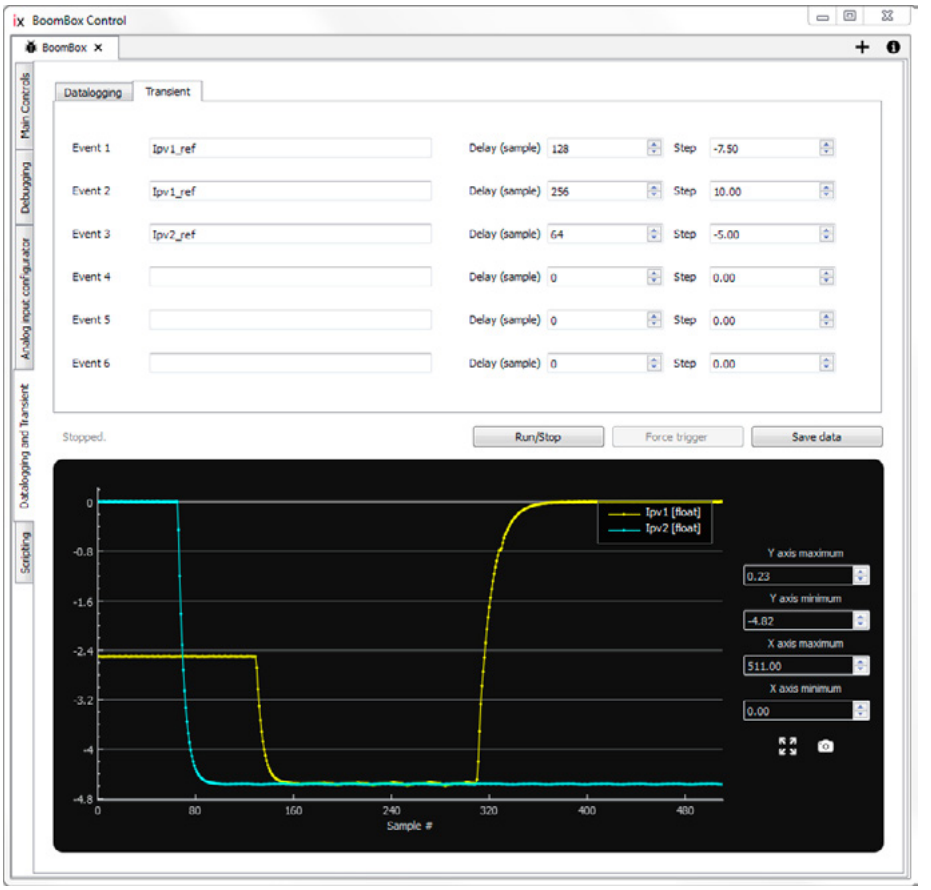


Fig. 27. BoomBox Control Transient generator screen.

PERIPHERAL DRIVERS

Abstract — This chapter describes the driver framework provided to the user to access the various BoomBox peripherals from the DSP in a transparent manner.

Keywords — *ADC, Acquisition, PWM, Frequency generator, Freq gen, Interrupt, ISR, IRQ, Sampling clock, GPI, GPO, DAC, User LED*

6.1 ANALOG DATA ACQUISITION SYSTEM (ADC)

As previously presented in section 2.2, the configuration of the analog input chain is split into a hardware part and a software part. The present chapter deals with the *software part* of the data acquisition and explains how to use it from the C code. More information on how to properly set-up the hardware part is given in section "2.2 Analog inputs", page 13.

From the 16-bit digitally-converted value U_{CONV} (see Fig. 5, page 14) the software driver computes the floating-point quantity float_in such that :

$$\text{float\_in} = U_{CONV} \cdot \text{gain} + \text{offset}$$

Note :

Apart from allowing the manipulation of meaningful quantities, the gain and offset parameters also provide an easy way of compensating for sensitivity errors in the input chain.

Warning :

The hardware and software gains should not be confused. The first aims to fully exploit the input voltage range of the analog-to-digital converter and thus increase the resolution and reduce quantization effects. The second aims to ease-of-use and calibration purposes only.

6.1.1 TYPICAL WORKFLOW

When configuring the analog inputs, the following steps are typically required :

1) For each channel, configure the necessary *hardware* gain, cut-off frequency and security limits *on the front panel*. Refer to section 2.2 for more information.
2) For each channel, determine the necessary *software* gain and offset between the converted voltage U_{CONV} and its corresponding quantity float_in.

   If necessary, these parameters should be determined through a *calibration procedure*. If using an imperix sensor, the gain parameter can simply be read on the sensor case.
3) For each channel, apply the computed software parameters through the routine :

   a) SetADCAdjustments();
4) Configure the sampling instant with respect to other operations (PWM, interrupts) using the routine

   b) ConfigSampling(freqgen, phase);

5) Get the desired measurements in real time (during the interrupts) through the routine :

c) GetADC();

6.1.2 CONFIGURING THE DATA ACQUISITION SYSTEM

a) SetADCAdjustments(channel, gain, offset);

This routine sets the parameters of the software affine transformation for each channel separately. The corresponding arguments are given in Table 9 :

Argument	Description
channel	*Index of the channel that is addressed (0 ≤ channel ≤ 15)*
gain	*Desired gain/sensitivity*
offset	*Desired corrective offset*

Table 9. Configuration parameters of an ADC channel.

The default values for the affine transformation parameters are the following :

Code :

```
float gain = 1.0;
float offset = 0.0;
```

6.1.3 CONFIGURING THE SAMPLING CLOCK

The sampling clock is produced in the same way as the two external interrupt sources. The corresponding principles of operation are described in section 6.2 and 6.6. To configure it, the following routine should be called in UserInit() :

b) ConfigSampling(freqgen, phase);

The phase parameter indicates the instant at which the ADC measurements are available in memory, with respect to the period of the frequency generator used as source. The physical sampling is therefore made slightly before that instant. In practice, *4 µs, what corresponds to the conversion and acquisition delay.* With a sampling clock and an interrupt configured on the same frequency generator and with the same phase, the data read in the interrupt will be the latest measurement it is possible to sample.

Example :

The following code configures the sampling clock so that the sample is available at the middle of the period of the frequency generator used to generate the PWM outputs :

```
ConfigSampling(1, 0.5);  // Phase of 180° between frequency generator #1 and sampling
```

6.1.4 RETRIEVING CONVERTED MEASUREMENTS

c) GetADC(channel);

At any time, the converted value can be easily retrieved by specifying which channel should be addressed. Table 10 shows the necessary arguments :

Argument	Description
channel	*Index of the channel that is addressed (0 ≤ channel ≤ 15)*

Table 10. Arguments of the GetADC driver routine.

Note :

Please also refer to the tutorials on *www.imperix.ch* for detailed configuration examples.

6.2 PULSE-WIDTH MODULATION SYSTEM (PWM)

6.2.1 BASIC PRINCIPLE OF OPERATION

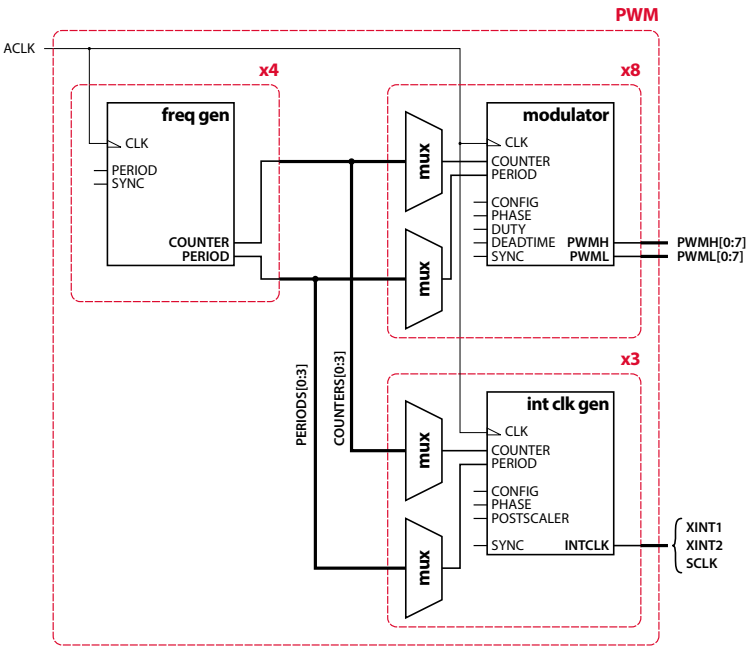


Fig. 28. Block diagram of the PWM system.

The BoomBox provides a monolithic PWM system which is implemented in FPGA. As seen in Fig. 28, it is composed of the following subsystems :

» 4 **frequency generators** (freq gen), which act as common *frequency* sources for other blocks that can be mapped to them. The latter can be either **modulators** or **interrupt clock generators**. The block diagram of each frequency generator is shown in Fig. 29 :

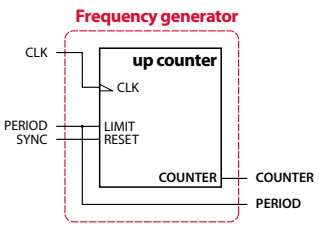


Fig. 29. Block diagram of a frequency generator.

» 8 **modulators**, corresponding to each PWM output *channel*, that define the *switching instants* of the gate-drive signals. Each channel features two complementary signals that correspond to one *switching cell*. Fig. 30 shows the corresponding block diagram.

For each switching cell, only three possible states are defined, as shown in Table 11 :

State	Description
HIGH	*The upper signal is ON, the lower is OFF*
LOW	*The upper signal is OFF, the lower is ON*
BLOCKED	*Both signals are OFF*

Table 11. Switching states of each PWM channel.

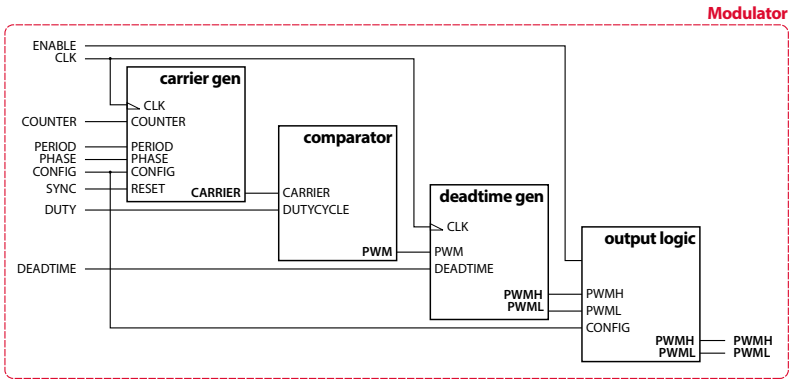


Fig. 30. Block diagram of a modulator.

» 3 **interrupt clock generators** (int clk gen) provide clocks to drive the two DSP external *interrupt sources* XINT1 and XINT2, as well as the ADC *sampling clock* SCLK. These are connected to the INTCLK output of each interrupt clock generator, as shown in Fig. 31 :

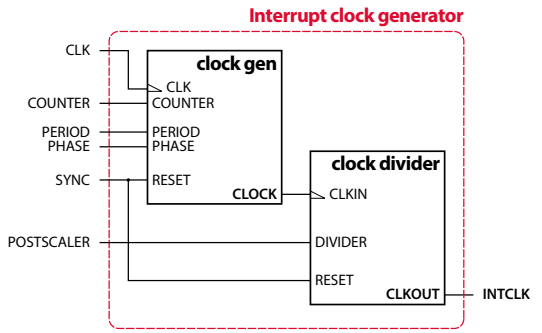


Fig. 31. Block diagram of an interrupt clock generator.

6.2.2 GENERATED PWM PATTERNS

Depending on the configuration of the carrier and their relative phase shift, various PWM patterns can be generated :

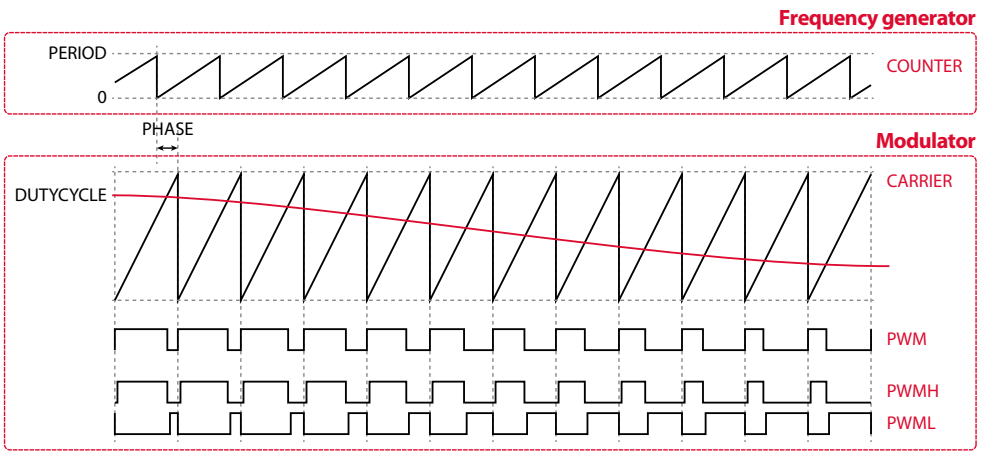


Fig. 32. Typical PWM pattern corresponding to a single-edge modulation strategy, with a sawtooth carrier.

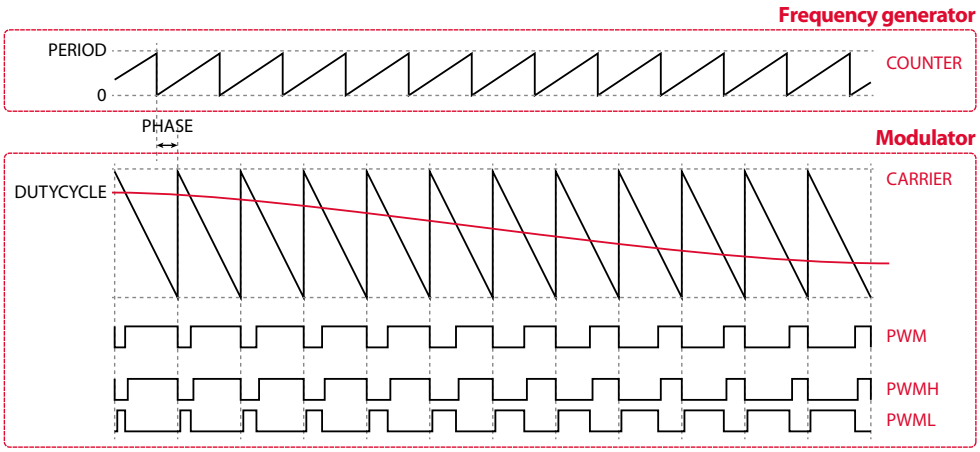


Fig. 33. Typical PWM pattern corresponding to a single-edge modulation strategy, with an inverted sawtooth carrier.

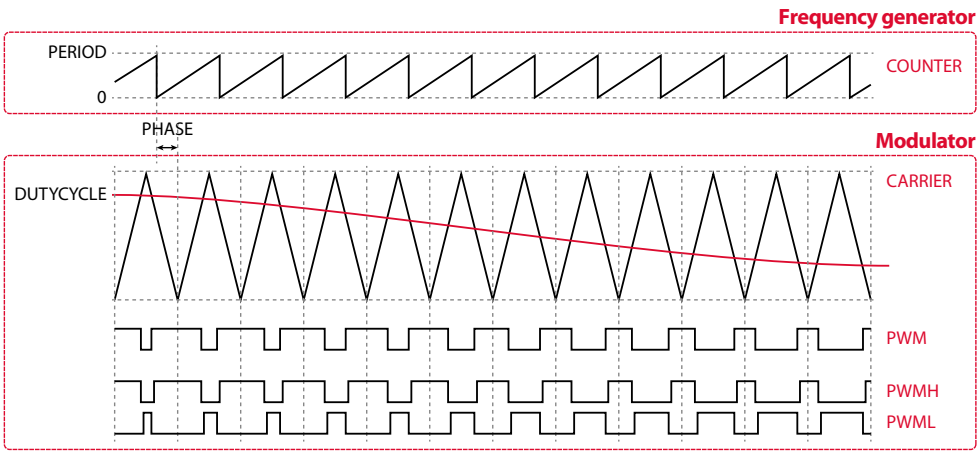


Fig. 34. Typical PWM pattern corresponding to a double-edge modulation strategy, with a triangle carrier.

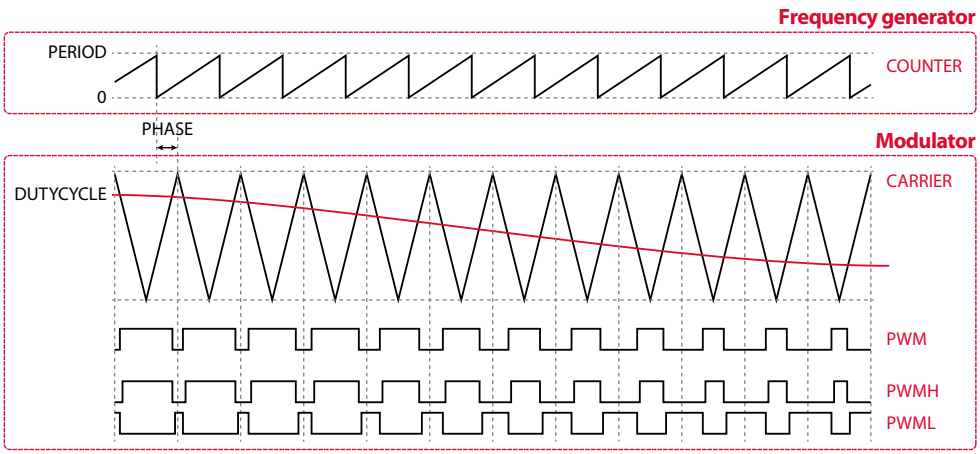


Fig. 35. Typical PWM pattern corresponding to a double-edge modulation strategy, with an inverted triangle carrier.

Using multiple modulators on a single common frequency generators enables the user to generate interleaved PWM signals by varying each individual modulator's phase.

6.2.3 INTERRUPT CLOCKS

Similarly to the PWM modulators, interrupt clock generators can be mapped to one of the frequency generator modules. They are used to drive one of the two external DSP interrupts (XINT1 and XINT2) and the ADC sampling clock (SCLK). The produced interrupt clocks can be seen in Fig. 36. More information on the API to configure the interrupts and the sampling clock can be found in sections 6.6 and 6.1.3.

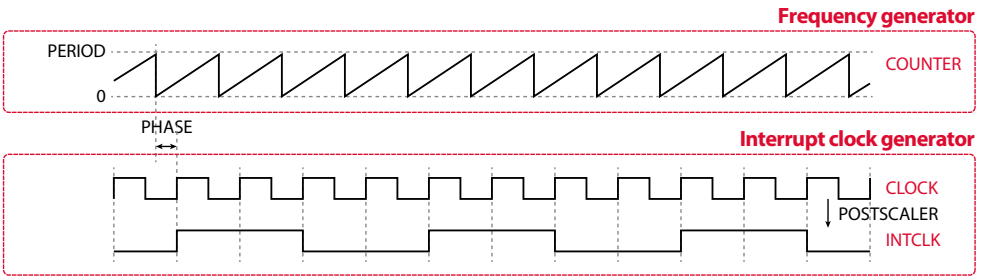


Fig. 36. Interrupt clock generated from a frequency generator module.

6.2.4 SYNCHRONIZATION OF FREQUENCY GENERATORS

Using the SYNC signal, all the frequency generator modules can be synchronized as seen in Fig. 37. This synchronization mechanism is triggered by calling the SyncFreqGens() routine of the PWM system.

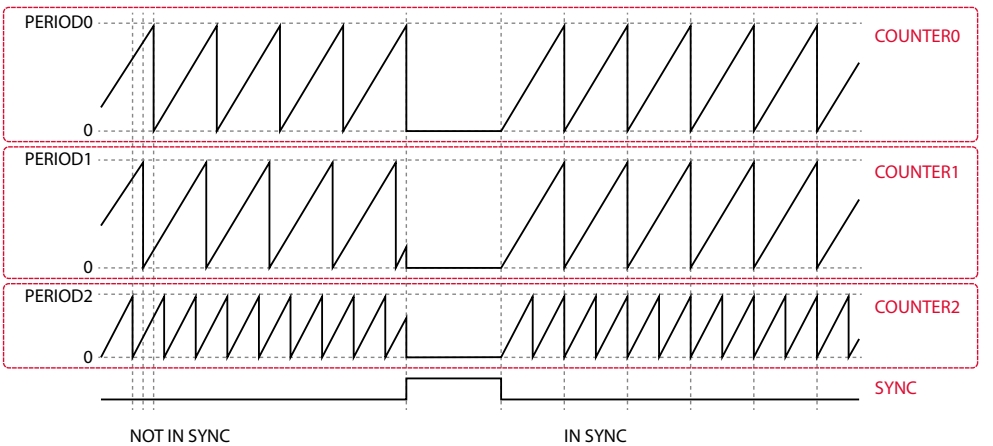


Fig. 37. Synchronization of multiple frequency generators of same and different periods.

6.2.5 TYPICAL WORKFLOW

In order to properly use the PWM signal generation system, the user is advised to make use of the following workflow and driver-layer routines :

1) Configure an available frequency generator module :

a) SetFreqGenPeriod(...);

2) Configure the PWM channel :

b) ConfigPWMChannel(...);

c) SetPWMPhase(...);

3) Activate the PWM channel :

d) ActivatePWMChannel(...);

4) Update the duty-cycle during the interrupts :

   e) SetPWMDutyCycle(...);

5) After each configuration change, the FPGA registers have to be updated :

   f) UpdatePWMData();

Note :

Please also refer to tutorial n°1 for a detailed example on how to use and configure the PWM system. The tutorials are available at the address *http ://imperix.ch/category/code-examples*.

6.2.6 CONFIGURING THE FREQUENCY GENERATORS

A frequency generator is characterized by a unique PERIOD, which can be configured using the following routine :

   a) SetFreqGenPeriod(freqgen, period);

This routine sets the PERIOD parameter of the chosen frequency generator (freqgen). Inside the FPGA, a timer counts up at a rate of 30 MHz. The switching frequency f_{sw} for each frequency generator is given by the following relation :

$$f_{sw} = 30\ MHz\ /\ PERIOD$$

The range of achievable switching frequencies depends on the required angular resolution. Table 12 gives an overview of the achievable performance for a few switching frequencies :

f_{sw} frequency	Relative resolution	Angular resolution
1 kHz	*0.03 ‰*	*< 0.02°*
50 kHz	*1.6 ‰*	*0.6°*
200 kHz	*6.6 ‰*	*2.4°*
1 MHz	*33 ‰*	*12°*

Table 12. Angular resolution with respect to the switching frequency

Example :

A PWM switching frequency of 15 kHz on frequency generator #1 can be achieved using the following code:

```
SetFreqGenPeriod(1, 2000) ;  // switching frequency = 30 MHz/2000 = 15 kHz
```

6.2.7 CONFIGURING THE PWM CHANNELS

The following routines are used to configure each PWM channel :

   b) ConfigPWMChannel(channel, freqgen, style, deadtime);

This function configures a PWM channel without enabling it. It sets all parameters of the given PWM channel, except the phase. The style parameter can be one of the following :

- TRIANGLE
- SAWTOOTH
- INVTRIANGLE
- INVSAWTOOTH

c) SetPWMPhase(channel, phase);

This function sets the phase of the PWM channel with respect to the main counter of the associated frequency generator. The phase parameter is given as a **float** and the actual value written to the configuration register is computed depending on the period and the configured deadtime. A value of **1.0** corresponds to a 360° phase shift.

Note :

These two configuration routines should be called in "UserInit()". It is recommended not to change these parameters in any of the UserInterrupt() routines.

Example :

Assuming that the switching frequency of frequency generator #1 has been set to 15 kHz as described in section 6.2.6, the following code configures channels 2 and 3 to use frequency generator #1 with a dead-time of 300 ns and sawtooth carriers. Besides, channel 2 has a phase of 0°, while channel 3 is phase-shifted by 90°:

```
ConfigPWMChannel(2, 1, SAWTOOTH, 10);  // DT of 10x TPWM clk = 10x(1/30MHz) = 300ns
ConfigPWMChannel(3, 1, SAWTOOTH, 10);  // DT of 10x TPWM clk = 10x(1/30MHz) = 300ns
SetPWMPhase(2, 0.0);                   // phase shift of 0.0x360° = 0°
SetPWMPhase(3, 0.25);                  // phase shift of 0.25x360° = 90°
```

6.2.8 ACTIVATING THE PWM CHANNEL

d) ActivatePWMChannel(channel);

This routine is used to indicate to the FPGA PWM system that it should produce a valid modulation output on the specified channel when the BoomBox goes into the **OPERATING** state. This can also be used in multi-converter applications to enable or disable parts of the application circuit depending on a user-level state machine. For this purpose, the corresponding routine **DeactivatePWMChannel(channel)** is available.

6.2.9 UPDATING THE DUTY-CYCLE

e) SetPWMDutyCycle(channel, dutycycle);

This routine computes the correct value for the **DUTYCYCLE** register depending on the carrier type and period. It is meant to be used in the **UserInterrupt()** routines to update the PWM outputs in real-time.

6.2.10 UPDATING THE CONFIGURATION INSIDE THE FPGA

f) UpdatePWMData();

This last step is necessary in order to actually update the PWM data inside the FPGA but in a local buffer. By doing so, it is easily guaranteed that the modulation parameters are always updated simultaneously among all PWM channels.

Note :

In fact, the PWM parameters are actually applied only when the frequency generator counter reaches zeros. This guarantees the integrity of the switching pattern even when multiple switching frequencies are used.

6.2.11 ENABLING THE OUTPUTS

Outputs are automatically enabled by the core in the **OPERATING** state and disabled in the **BLOCKED** and **FAULT** states.

6.3 ADVANCED PWM MODES

In addition to the standard pulse-width modulation system described in the previous section, the BoomBoox authorizes two advanced modes for PWM generation. Although these alternative implementations are done inside the FPGA, the activation and configuration of these features are accessible through G-code.

6.3.1 MODULATORS WITH SINGLE PWM OUTPUT

Unlike the standard PWM configuration that provides 8 PWM modulators with complimentary signals output (PWMH, PWML), this advanced mode activates 16 modulators per BoomBox, each generating a single PWM signal. In this mode, each modulator benefits from all the parameters described in the chapter above (source, phase, carrier type, deadtime,...) and are configured in a similar way.

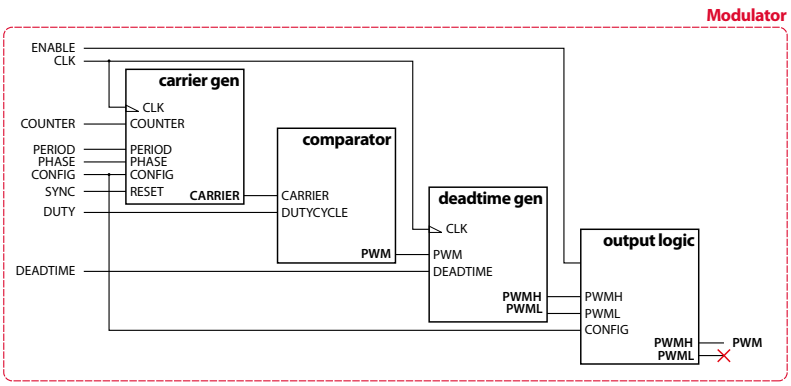


Fig. 38. Block diagram of a modulator.

This advanced mode can be enabled during the initialization by calling the following function once: SetMuxMode(MUX16). After calling this command, all the 16 optical outputs are driven by 16 independent modulators. Then, the procedure for configuring individual modulators is identical as explained in section 6.2.7, page 47. The following table lists the outputs used on the front panel for each modulator output.

Modulator #	Front panel optical output
0	*PWM0H*
1	*PWM1H*
...	...
7	*PWM7H*
8	*PWM0L*
9	*PWM1L*
...	...
15	*PWM7L*

Table 13. Front panel PWM channel mapping for single-output mode.

6.3.2 MODULATORS WITH PWM AND ACTIVE OUTPUTS

In the normal MUX8 mode, it is also possible to have the regular PWMH output with an ACTIVE signal instead of PWML. This is useful in some specific use cases where half-bridge drivers require one PWM output and an enable signal.

The ACTIVE signal of a PWM channel output pair is on if that channel is configured in that mode, has been activated, and the BoomBox is enabled.

Note

When using this mode, it is assumed that the gate drivers connected at the other side of the fiberoptic links re-generate an adequate deadtime.

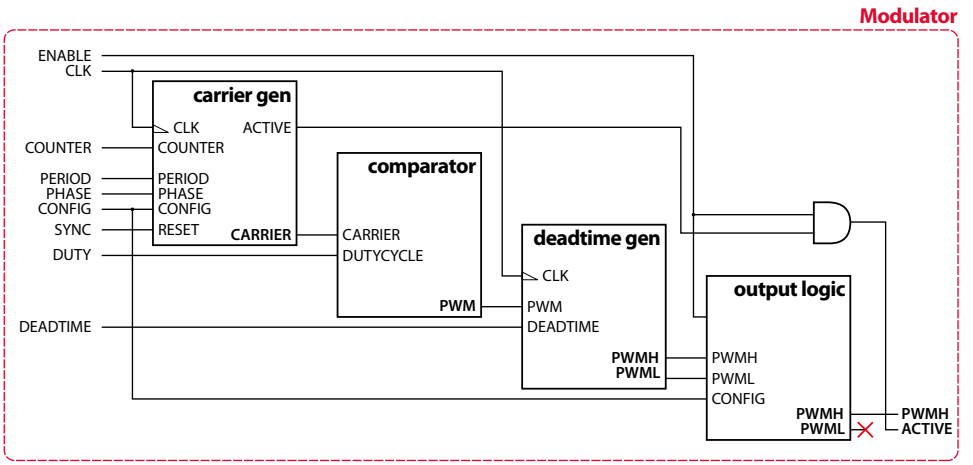


Fig. 39. Block diagram of a modulator.

To use this mode, the following overloaded version of the ConfigPWMChannel routine must be used:

g) ConfigPWMChannel(channel, freqgen, style, deadtime, outmode);

with the outmode parameter set to PWMH_ACTIVE.

6.3.3 DIRECT ACCESS TO OPTICAL OUTPUTS

The BoomBox also provides a specific mode where the state of each optical output can be directly written by the user's control routines, hence bypassing the modulators. As shown by the following figure, this mode can be enabled on a partial set of PWM lines, while the others are still driven by standard modulators.

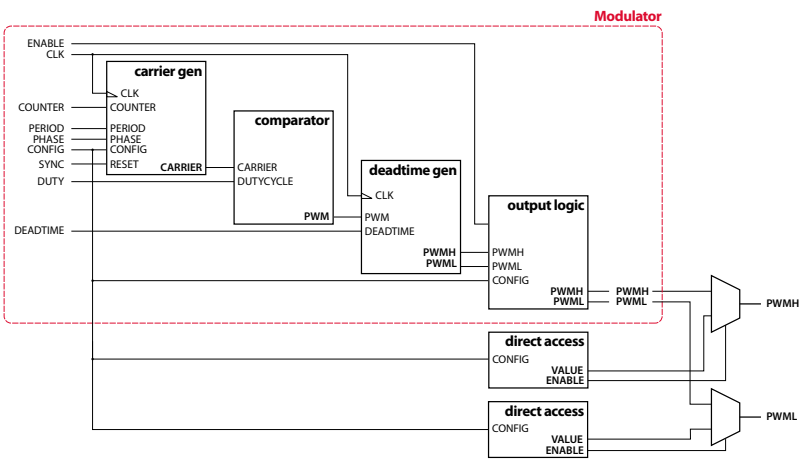


Fig. 40. Block diagram of direct access logic for a given PWM-pair.

This mode is disabled by default. To be used, it must be enabled (allowed) calling **AllowOpticalOutputDirectAccess()**. Then, each ouput line has to be configured by calling the following function during the initialization : **UnlockOpticalOutputDirectAccess(int output).**

The following figure summarizes the physical PWM line attribution as a function of the optical output number passed to the function.

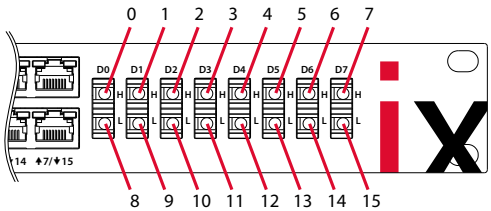


Fig. 41. Overview of front panel optical outputs numbering for direct access.

Once the direct access is unlocked, changing the state of a optical output can be done by calling **ForceOpticalOutput(int output, int value)**. This function directly updates the FPGA (after a 200-600ns of communication delay).

Apart from the possibility to force the state of a unique optical output, there is also a possibility to update the state of all direct access-enabled outputs. This can be done by calling **ForceOpticalOutputRegister(unsigned int value)**. This method should be preferred when numerous calls to this routines are necessary (avoiding performance issues) or when the switching of multiple optical outputs must to be guaranteed to be simultaneous.

Note :

Upon a fault detection, all optical outputs configured for direct access are automatically reset at '0' state (inactive).

Warning:

Unlike with normal PWM output modes, the direct access to optical outputs does NOT guarantee the complementarity of gating signals by pairs. As such, it is of the responsibility of the user to make sure that no inappropriate gating signals may ever be produced in order to avoid shoot-trough in any switching cell.

6.4 GENERAL-PURPOSE INPUTS (GPI)

The GPI module is constituted by a single 8-bit register. The two related routines are presented in Table 14.

Name	Argument	Functionality
GetGPIbit	*n : the number of the bit (7 to 0)*	*Get the value of the chosen bit*
GetGPI	*void*	*Get the value of the complete register*

Table 14. General-purpose Input routines

6.5 GENERAL-PURPOSE OUTPUTS (GPO)

The GPO module is constituted by a single 8-bit register. Several driver routines are associated with it :

Name	Arguments	Functionality
SetGPObit(n)	*n : the number of the bit (7 to 0)*	*Set the chosen bit to 1*
ClearGPObit(n)	*n : the number of the bit (7 to 0)*	*Set the chosen bit to 0*
ToggleGPObit(n)	*n : the number of the bit (7 to 0)*	*Toggle (swap) the value of the chosen bit*
ForceGPObit(n,val)	*n : the number of the bit (7 to 0)* *val : the new value of the bit*	*Force a given bit to a given value*
SetGPO(val)	*val : the new value of the register*	*Set a value on the whole GPO register*

Table 15. General-purpose Output routines

6.6 INTERRUPT SOURCE SELECTION (IRQ)

6.6.1 BASIC PRINCIPLE OF OPERATION

The BoomBox has two external hardware interrupt lines. They are connected to the FPGA PWM core, which makes it possible to run code synchronously with the operation of the modulation. Additionally, one timer-based interrupt is available for low-priority tasks. Fig. 42 shows a flow chart of the interrupt management process.

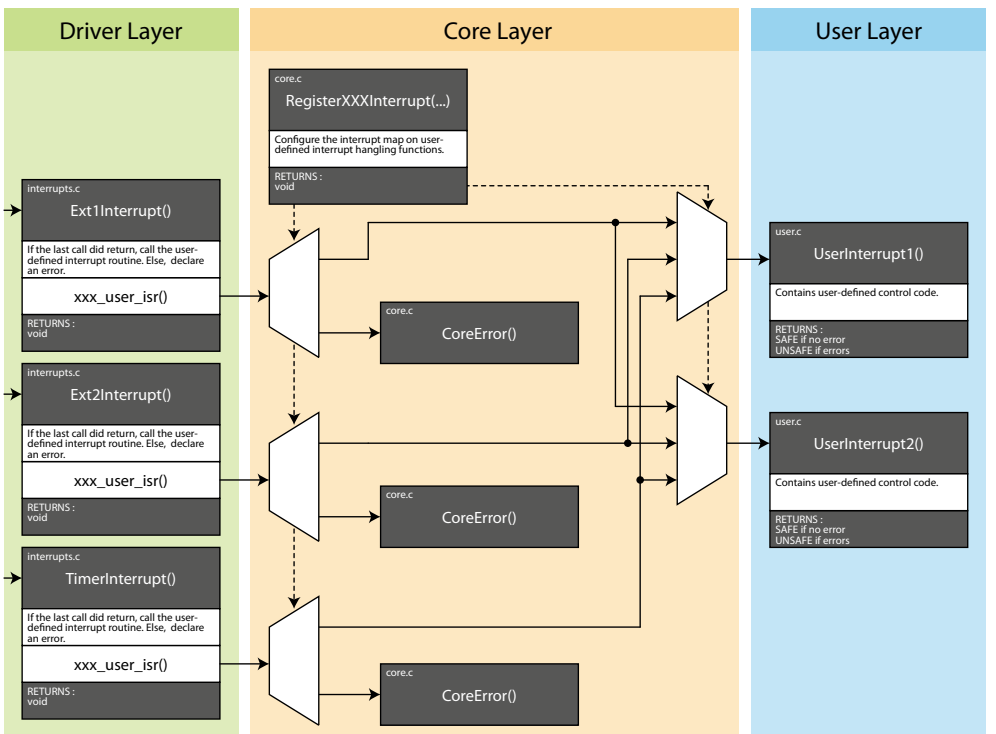


Fig. 42. Flow chart of the interrupt management process.

6.6.2 REGISTERING INTERRUPTS

It is possible to register an interrupt service routine for any of the three available interrupt sources using the following functions :

» First event-based external interrupt source (XINT1) :

```
RegisterExt1Interrupt(user_isr, source, phase, postscaler);
```

» Second event-based external interrupt source (XINT2) :

RegisterExt2Interrupt(user_isr, source, phase, postscaler);

» DSP timer-based interrupt :

RegisterTimerInterrupt(user_isr, period);

The priority of these interrupts is highest for **XINT1**, intermediate for **XINT2**, and lowest for the timer-based interrupt.

The two interrupt service routines **UserInterrupt1** and **UserInterrupt2** are pre-defined in the provided **user.c/.h** canvas, but any function based on the same template can be used.

The external interrupt sources **XINT1** and **XINT2** are connected to the FPGA and mapped to the chosen frequency generator module (**source**) by the above functions, enabling DSP code to be executed synchronously with the PWM operation.

Note :

When both interrupt service routines are registered simultaneously, it must be reminded that the XINT1 source has a higher level of priority than XINT2.

Example :

Assuming that the switching frequency of frequency generator #1 has been set to 15 kHz as described in section 6.2.6, the following code configures UserInterrupt1 to be executed every 500 us using the DSP timer-based interrupt, while UserInterrupt2 is executed synchronously to frequency generator #1, phase-shifted by 90°, once every two PWM periods:
RegisterTimerInterrupt(&UserInterrupt1, 500); // 500 us => 2 kHz
RegisterExt1Interrupt(&UserInterrupt2, 1, 0.5, 1);

Table 16.

6.7 DIGITAL TO ANALOG CONVERTER (DAC)

The DSP code can write new values in the 16-bit DAC registers at any time. The FPGA module is responsible for updating the DAC chip at a rate of approximately 150 kHz. Two routines are provided :

» **SetDACValue(channel, value);** writes the raw **unsigned int** value directly to the register. A value of **0** will output a voltage of -5V, while a value of **65535** will output +5V.

» **SetDACVoltage(channel, voltage);** computes the correct register value necessary to output the **float** voltage provided as parameter.

Example :

The following code executed periodically outputs three 120° phased sine waves:
SetDACVoltage(0, 5.0 * sin(angle + 0.0 * 2.0 * PI / 3.0)); // A0
SetDACVoltage(1, 5.0 * sin(angle + 1.0 * 2.0 * PI / 3.0)); // A1

6.8 USER LED

The **USER** LED can be controlled using the following routine :

SetUserLED(color);

where **color** can be one of the pre-defined enums : **LED_OFF, LED_RED, LED_GREEN** or **LED_ORANGE**).

6.9 INCREMENTAL ENCODER INPUT

6.9.1 BASIC PRINCIPLE OF OPERATION

The encoder input allows interfacing with one or two incremental encoders providing quadrature outputs (usually called A and B), with or without a reset line (usually called Z). These outputs have to be connected to the GPI lines at the back of the BoomBox (see pinout in "Table 5. GPI pin assignments.", page 20).

The encoder module counts all 4 edges of the A and B inputs, meaning that the angular resolution is actually 4 times better than the ppr value usually specified for a given encoder. Additionally, the value returned by the BoomBox to the user is not the instantaneous value of the counter, but the value latched at the last rising edge of the SCLK, similar to the S&H (sample and hold) feature of the analog inputs. This is done to ensure the timing accuracy of the measured position.

The following figures show typical timing diagrams for clockwise and counterclockwise rotation of standard incremental encoders:

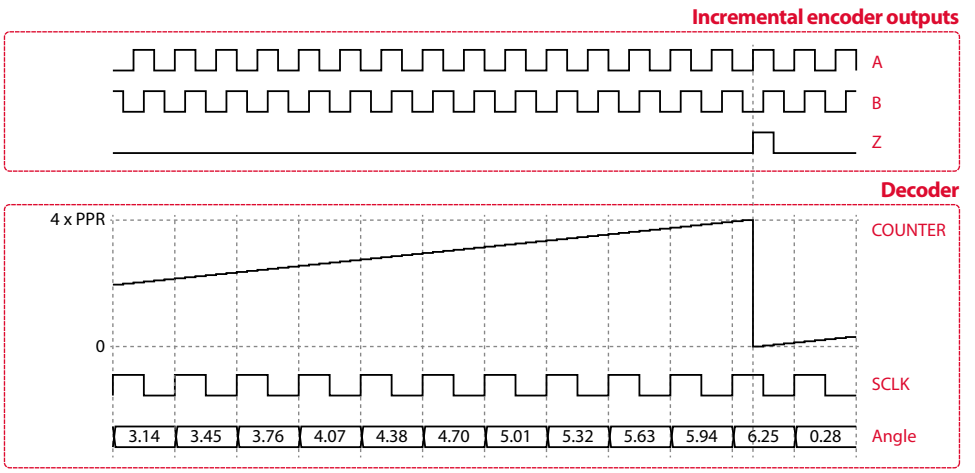


Fig. 43. Typical timing diagram of an incremental encoder rotating clockwise

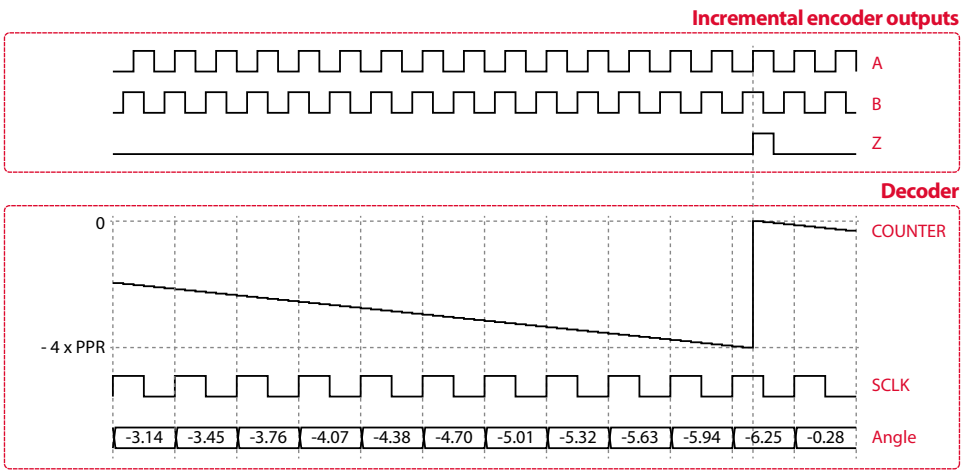


Fig. 44. Typical timing diagram of an incremental encoder rotating counterclockwise

6.9.2 CONFIGURING A DECODER MODULE

Configuring the decoder modules is done using one of the following routines:

a) ConfigDECModule(ppr, inputmode, module)

This routine configures decoder module 0 or 1 (according to the module parameter) with the given pulses per rotation (ppr parameter). Only the ppr parameter is mandatory. If not specified, inputmode is SINGLEENDED and module is 0 by default.

Note:

If the specified inputmode is DIFFERENTIAL, then the configured module is always number 0. Indeed, as all physical input signals are used by module 0, those of module 1 become inaccessible, and module 1 is disabled.

b) ConfigDECModule(ppr, inputmode, module, resetmode, direction, invert)

This routine provides additional configuration parameters. resetmode can be set to MAXVALUE, which disables the Z input and resets the counter each turn based on the provided ppr value. Additionally, the counting direction can be reversed or the inputs inverted using the direction and invert parameters respectively.

6.9.3 ACCESSING THE COUNTER VALUE

Accessing the counter value is done using the following routine:

float GetDECAngle(module)

This returns the last sampled counter value, converted to radian according to the configured ppr value.

SOFTWARE LICENSE VERSIONS

Abstract — This chapter describes the different versions of the BoomBox software, their advantages and limitations. Some features are exclusively present in the expert version of the BoomBox software package. This version essentially aims to operate multiple BoomBoxes as a unique control platform, making use of one BoomBox as a master as well as additional BoomBoxes as I/O extension units.

Keywords — *Installer, license, Standard software, Lite software, Expert software, Multi-BoomBox, Master, Slave, Synchronization, Several BoomBoxes, I/O extension*

7.1 INSTALLING THE LICENSE FILE

When installing BoomBox software, be it the C/C++ development environment or the Simulink Automated Code Generation toolbox, the user is asked to input the license file provided by imperix.

The contents of this license file determines which version will be installed on your computer. The following paragraph describes the specificities of each version.

7.2 SOFTWARE PACKAGES

The BoomBox is available in three variants, based on the same hardware, but with different software features:

- A **standard** version, allowing the operation of one BoomBox with full functionality.
- A **lite** version, allowing the operation of one BoomBox with several limitations, providing a low-cost alternative to the standard version.
- An **expert** version, addressing the operation of multiple BoomBoxes in *I/O extension mode*, using the multi-pin connector present on the rear side of the BoomBox.

> *Note:*
> **When operated in *I/O extension mode*, the stacked BoomBoxes behave as a single monolithic system which possesses further I/O capabilities. Concretely, the stack of BoomBoxes operates using one DSP (that of the master) and several FPGAs (in the master and the other units). This distinguishes from the *parallel operation* of BoomBoxes, that communicate through a rather slow medium (e.g. CAN) and remain essentially independent controllers.**

All versions are fully compatible and inter-changeable.

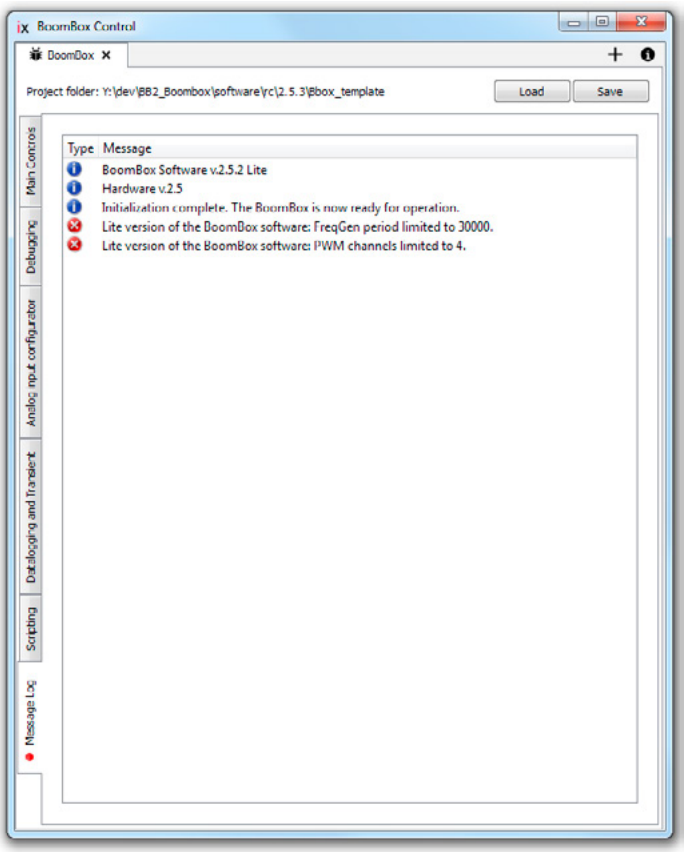
7.3 LITE VERSION LIMITATIONS

The Lite software version has the following limitations:

- Switching and interrupt frequency : limited to 1 kHz

- PWM channels : limited to 4 modulators (8 or 4 outputs depending on output mode)
- ADC channels : limited to 8 inputs
- DAC channels : limited to 2 outputs
- Sampling configuration : phase fixed at 0.5

If a C/C++ code or SImulink model containing access to unavailable channels or trying to switch faster that 1 kHz is executed, warning messages will be displayed in BoomBox Control in the Message Log tab, showing which parameters have been intentionally altered to comply with the limitations of the Lite version.



7.4 EXPERT VERSION FEATURES

7.4.1 STACKING SEVERAL BOOMBOXES

To stack multiple BoomBoxes together, the following steps must be followed :

1) All devices must be daisy-chained using short flat cables (supplied by imperix).
2) The *master BoomBox* must contain a DSP carrier board from Texas Instrument.
3) The address of the master BoomBox must be set to **'0000'**.
4) The *slave BoomBoxes* must not contain a DSP carrier board
5) The slave BoomBoxes must have their addresses set in increasing order starting from **'0001'**.

Connection of the multi-BoomBox setup with the PC through JTAG or USB is done on the master device. The CAN ports are only available on the master BoomBox.

Fig. 45 shows an example of a stack constituted by three BoomBoxes.

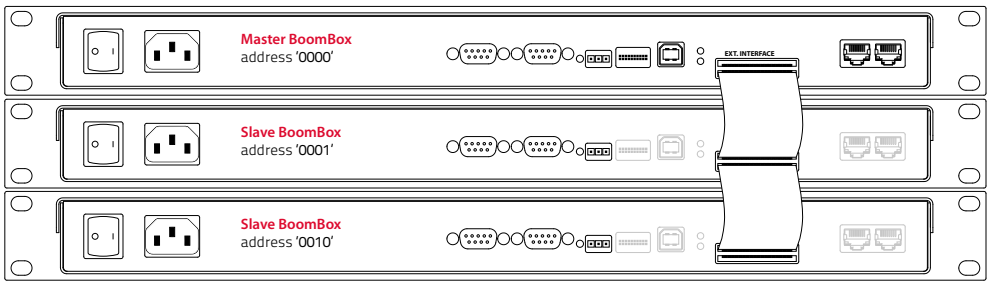


Fig. 45. Stack of three BoomBoxes using two units as I/O extensions. Disabled connectors on slave devices are shown in gray.

In order to change the address of a given BoomBox, the enclosure must be opened and the DIP switch present of the motherboard configured with the correct address. The floorplan of the motherboard is shown in Fig. 46, highlighting the location of the DIP switch. The corresponding configuration of the DIP switches inside the BoomBoxes is shown in Fig. 47.

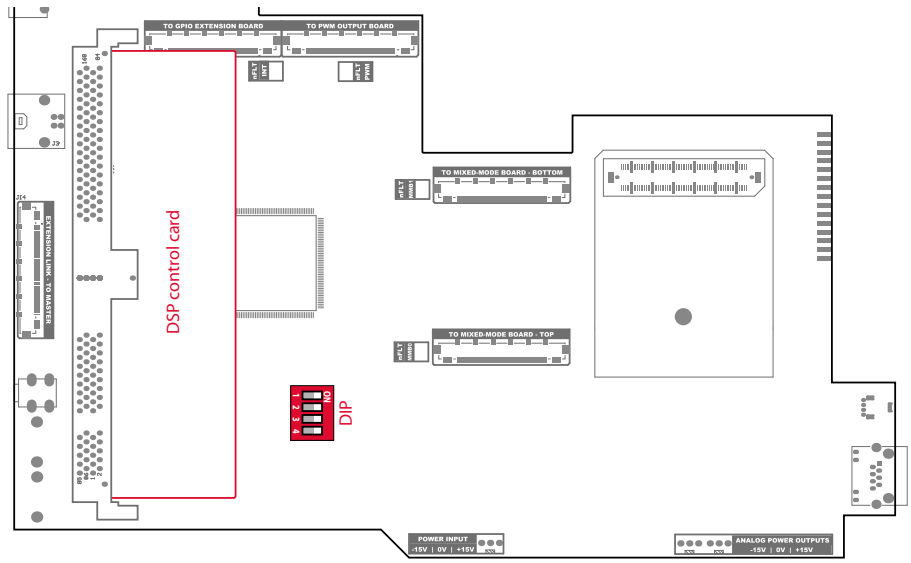


Fig. 46. Floorplan of the motherboard of the BoomBox, showing the location of the DSP control card and the DIP switch programming the address.

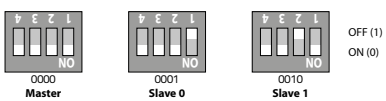


Fig. 47. Positions of the DIP switches inside the three stacked BoomBoxes.

When properly configured, the SYNC LED of the front panel should display the appropriate color, namely:

» **GREEN** for the master BoomBox
» **ORANGE** on the slave BoomBoxes

» **RED** if the software running on the DSP is incompatible with multi-BoomBox operation or in case of synchronization error between the BoomBoxes.

Note:

This LED remains permanently off with the standard version of the BoomBox software.

Additionally, at the end of the boot sequence of the multi-BoomBox setup, the version of the BoomBox library is shown on the command line prompt, as well as the list of the detected BoomBoxes (FPGA devices), for instance:

Console:

```
BoomBox CLI Oct 19 2015 10:57:35.
Software v.2.2.0 Expert
BoomBox #0 : Hardware v.2.2
BoomBox #1 : Hardware v.2.2
Initialization complete. The BoomBox is now ready for operation.
Type 'enable' (or 'disable') to authorize (or block) the PWM outputs.
Type 'user' to access the user-defined commands and 'ls' to list all available commands.
user@boombox / >
```

7.4.2 PRINCIPLES OF OPERATION

7.4.2.1 PROGRAMMING OF THE SETUP

As only the master BoomBox contains a DSP, the programming of the overall system, regardless of the number of stacked units, is absolutely identical as the programming of one unit only.

7.4.2.2 USE OF THE PERIPHERAL DRIVERS

The operation of a multi-BoomBox setup using the expert software package is very similar to the operation of a single BoomBox, except that all function prototypes possess a optional argument indicating which BoomBox shall be addressed. The code below gives a simple comparative example:

Standard version, one BoomBox

```
I1 = GetADC(2);                 // ch2, bb0
I2 = GetADC(4,0);               // ch4, bb0
SetDACVoltage(3,...,0);         // ch3, bb0

SetPWMDutyCycle(7,...);         // ch7, bb0
SetPWMDutyCycle(6,...,0);       // ch6, bb0

UpdatePWMData();                // bbox0
```

Expert version, several BoomBoxes

```
I1 = GetADC(2,0);               // ch2, bb0
I2 = GetADC(3,1);               // ch3, bb1
SetDACVoltage(0,...,1);         // ch0, bb1

SetPWMDutyCycle(7,...,2);       // ch7, bb2
SetPWMDutyCycle(6,...,1);       // ch6, bb1

UpdatePWMData(0);               // bbox0
UpdatePWMData(1);               // bbox1
UpdatePWMData(2);               // bbox2
```

Naturally, exceptions to this rule are the routines directly related to the DSP and the software core, which are dependent upon the DSP to perform their function and are subsequently not available on slave BoomBoxes. Examples are given by CAN functions, interrupt configuration functions, etc.

In any case, the header files provide detailed information on all functions and should be able to clarity any doubts about differences between the standard and expert versions.

7.4.2.3 FAULT MANAGEMENT

When a hardware fault[1] occurs on any input of any BoomBox, the PWM outputs of all BoomBoxes are immediately blocked. The fault can subsequently be acknowledged on the corresponding BoomBox device, as in single BoomBox applications.

7.4.2.4 SYNCHRONIZATION ACROSS BOOMBOXES

In a multi-BoomBox setup, 4x N clock generators are available to the user, where N is the number of stacked BoomBoxes. However, when it is desired to synchronize some operations across multiple BoomBoxes (e.g. modulation, sampling), some constrains exist so as to guarantee some frequency and phase relationships across the BoomBoxes.

In order to run some operation such as sampling or modulation with the exact same phase and frequency across two or more BoomBoxes, their respective frequency generators must be configured in the same manner. The code below shows an example that mirrors the frequency generator #2 across three BoomBoxes:

Code example :

```
SetFreqGenPeriod(2, PERIOD, 0);                  // PERIOD on freq gen #2 of bbox #0
SetFreqGenPeriod(2, PERIOD, 1);                  // PERIOD on freq gen #2 of bbox #1
SetFreqGenPeriod(2, PERIOD, 2);                  // PERIOD on freq gen #2 of bbox #2

RegisterExt1Interrupt(&MyUserISR, 2, 0.0, 0);    // an ISR is registered on freq gen #2
```

In order to maintain a perfect relationship of all frequency generators #2, the BoomBoxes run internally some synchronization mechanisms that guarantee that all the events associated with these frequency generators remain perfectly synchronous, with a absolute maximum timing difference of ±125 ns (3 σ, 1 hour).

These mechanisms only operate when a DSP interrupt has been registered on the corresponding frequency generator. Therefore, in order to guarantee the synchronization across several BoomBoxes, the following rules must be followed:

» The same period must be configured on the same frequency generator in all Boomboxes accross which the synchronization is desired.
» An DSP interruption routine must be registered on that frequency generator (either EXT1 or EXT2) so that synchronization is guaranteed.

These principles of operation are illustrated in Fig. 48:

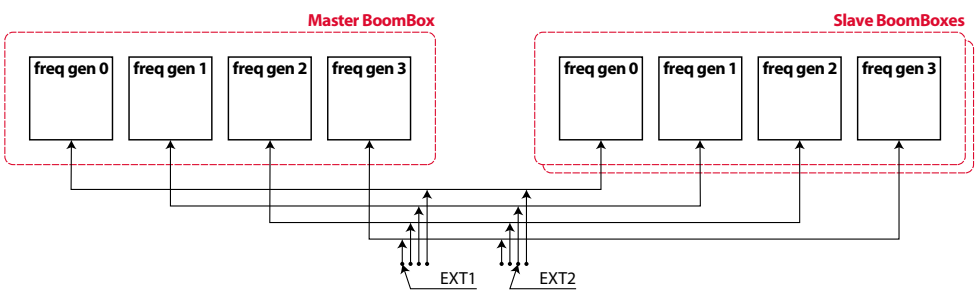


Fig. 48. Configuration of the frequency generator synchronization in multi-BoomBox operation.

A consequence of these principles of operation is that only *up to two* frequency generators can be fully synchronized across multiple BoomBoxes. On the other hand, this also

1. More information of the different types of fault can be found in §4.2.1, page 27.

means that the remaining *non-synchronized* frequency generators can be freely configured with different parameters between different BoomBoxes. This is particularly important, as this allows to *configure the sampling differently* as well.

7.4.2.5 CODE EXAMPLE

The following code example shows how to initialize a synchronous multi-BoomBox system with two interleaved channels located in two different BoomBoxes:

Code example :

```
SetFreqGenPeriod(1, 3750, 0);                  // 8 kHz on freq gen #1 of bbox #0
SetFreqGenPeriod(1, 3750, 1);                  // 8 kHz on freq gen #1 of bbox #1

RegisterExt1Interrupt(&MyUserISR, 1, 0.0, 0); // timebases are synchronized by ISR

ConfigSampling(1, 0.0, 0);                     // Bbox #0 samples at 0° of freq gen #1
ConfigSampling(1, 0.0, 1);                     // Bbox #1 samples at 0° of freq gen #1

ConfigPWMChannel(2, 1, SAWTOOTH, 10, 0);       // DT of 10x TPWM clk = 300ns
ConfigPWMChannel(2, 1, SAWTOOTH, 10, 1);       // DT of 10x TPWM clk = 300ns

SetPWMPhase(2, 0.0, 0);                        // phase shift of 0.0x360° = 0°
SetPWMPhase(2, 0.5, 1);                        // phase shift of 0.5x360° = 180°
```

Note:

It is necessary to call the SetFreqGenPeriod routines before the RegisterExt#Interrupt to enable synchronization of timebases across BoomBoxes.

In this case, the user-level interrupt service routine typically contains the following actions:

Code example :

```
MyUserISR{
    SetPWMDutyCycle(2, duty, 0);               // duty is applied on ch #2 of bbox #0
    SetPWMDutyCycle(2, duty, 1);               // duty is applied on ch #2 of bbox #1

    UpdatePWMData(0);                          // bbox 0
    UpdatePWMData(1);                          // bbox 1
}
```

7.4.3 ADVANCED SAMPLING OPTIONS

In a single BoomBox environment, all analog inputs are necessarily sampled simultaneously. The sampling instant is chosen by calling the ConfigSampling routine with the desired phase argument.

Respectively, in a multi-BoomBox environment, the BoomBoxes can be setup to sample their analog inputs either at *different instants* or at the *same instant*. This simply depends on the configuration of the corresponding frequency generator. In both cases, the configuration of the sampling requires to call the ConfigSampling routine for each device.

The three examples below illustrate different options:

Synchronous sampling:

```
SetFreqGenPeriod(1, 3750, 0);          // 8 kHz on freq gen #1 of bbox #0
SetFreqGenPeriod(1, 3750, 1);          // 8 kHz on freq gen #1 of bbox #1

RegisterExt1Interrupt(&MyISR, 1, 0.0, 0);

ConfigSampling(1, 0.0, 0)              // Bbox #0 is 4us before 0° of freq gen #1
ConfigSampling(1, 0.0, 1)              // Bbox #1 is 4us before 0° of freq gen #1
```

Correlated sampling:

```
SetFreqGenPeriod(1, 3750, 0);          // 8 kHz on freq gen #1 of bbox #0
SetFreqGenPeriod(1, 3750, 1);          // 8 kHz on freq gen #1 of bbox #1

RegisterExt1Interrupt(&MyISR, 1, 0.0, 0);

ConfigSampling(1, 0.0, 0)              // Bbox #0 is 4us before 0° of freq gen #1
ConfigSampling(1, 0.5, 1)              // Bbox #1 is 4us before 180° of freq gen #1
```

Independent sampling:

```
SetFreqGenPeriod(1, 3750, 0);          // 8 kHz on freq gen #1 of bbox #0
SetFreqGenPeriod(2, 3000, 0);          // 10 kHz on freq gen #2 of bbox #0
SetFreqGenPeriod(1, 3750, 1);          // 8 kHz on freq gen #1 of bbox #1
SetFreqGenPeriod(2, 3000, 1);          // 10 kHz on freq gen #2 of bbox #1

RegisterExt1Interrupt(&MyISR1, 1, 0.0, 0);   // MyISR1 is triggered by freq gen #1
RegisterExt2Interrupt(&MyISR2, 2, 0.0, 0);   // MyISR2 is triggered by freq gen #2

ConfigSampling(1, 0.0, 0)              // Bbox #0 is sampled on freq gen #1
ConfigSampling(2, 0.0, 1)              // Bbox #1 is sampled on freq gen #2
```

The synchronous sampling option is shown in Fig. 49:

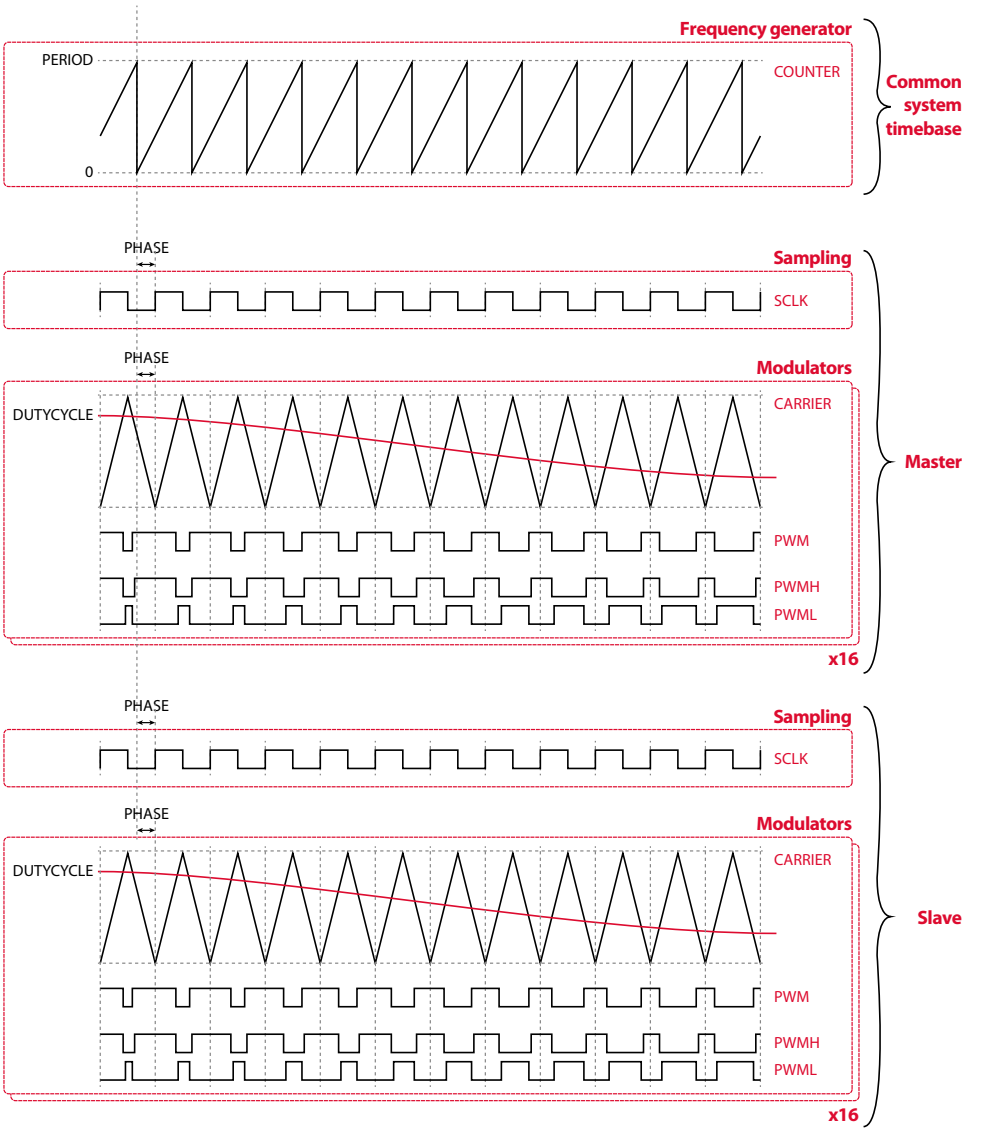


Fig. 49. Sampling and modulation in a multi-BoomBox environment.

SOFTWARE CHANGELOG

BoomBox software v2.0.0 :

» initial release.

BoomBox software v2.0.1 :

» FIX : In some cases, interrupts would never get enabled when DSP was started with fault present.
» FIX : GetADC return value was wrong when ADC saturated at high end.

BoomBox software v2.2.0 :

» NEW : Multi-BoomBox operation with synchronization of frequency generators.
» NEW : SetMuxMode and OpticalOutputDirectAccess provide two new ways of controlling the state of the output gating signals.
» NEW : SetCLIVars/GetCLIVars functions and corresponding CLI commands to conveniently read and write any global variable from the command line.
» NEW : Preparation of the command line interface for BBcontrol GUI software.
» NEW : Added the CLI silent mode (SendCLIMsg & ForceCLIMsg).
» NEW : Added an MPPT tracker to the API in the BoomBox template project.
» FIX : SetPWMDutyCycle : added saturation of dutycycle parameter.
» FIX : Improved robustness of CAN module in case of high error rate situations due to noise.
» FIX : In a very specific case, generated PWM signal was not able to reach the absolute off state; the smallest achievable duty cycle was 0.01 %.

BoomBox software v2.3.0

» NEW : Timestamping module to measure code execution times.

BoomBox software v2.3.1

» NEW : Support for BoomBox Control software with firmware updates and datalogging.

BoomBox software v2.3.2

» NEW : Support for BoomBox Control transient generator.

BoomBox software v2.4.0

» NEW : PWM alternate output mode : PWMH_ACTIVE.

DOCUMENT REVISION HISTORY

- 01.02.2013 : Initial version derived from prior EPFL version, N. Cherix.
- 22.05.2013 : Revision of the software part to comply with the new core layer, N. Cherix.
- 11.10.2014 : Complete revision of document structure, M. Lambert.
- 21.11.2014 : Revision of ACQ-related parts of the documentation, M. Lambert.
- 17.12.2014 : Revision of the software section to comply with the new PWM modulator, M. Lambert.
- 13.02.2015 : Update of the document template with contact information and warranty statement, N. Cherix.
- 26.03.2015 : Revision of PWM sawtooth modulation waveform polarity, M. Lambert.
- 14.04.2015 : Added v2.0.1 software changelog, M. Lambert.
- 04.09.2015 : Specification of ACQ timing, M. Lambert.
- 04.09.2015 : Documentation of PWM advanced modes, S. Delalay.
- 22.10.2015 : Revision of interlock and general-purpose inputs schematics; specification of overvalue detection delay, N. Cherix.
- 15.11.2015 : Addition of the expert software section, M. Lambert.
- 26.11.2015 : Added v2.2.0 software changelog, M. Lambert.
- 28.01.2016 : Typo fixes in code example §7.3.5, S. Delalay.
- 05.02.2016 : Update of Frontpanel USB paragraph, M. Lambert.
- 23.02.2016 : Added a note about the required function call sequence for Multi-Boom-Box frequency generator synchronization and update of code examples, M. Lambert.
- 12.07.2016 : Updated software changelog and added a chapter on PC software describing BoomBox Control, M. Lambert.
- 08.08.2016 : Updated 'Advanced PWM modes'. AllowOpticalOutputDirectAccess() was missing, S. Delalay.
- 30.11.2016 : Added documentation of Quadrature Encoder Inputs, M. Lambert.
- 22.12.2016 : Added documentation of the incremental decoder, PWMH_ACTIVE output mode, and Simulink Automated Code Generation feature, M. Lambert.
- 26.12.2017 : Revised outdated elements of the document, N. Cherix.

Contact

imperix ltd.
Rue de la Dixence 10
1950 Sion

phone: +41 (0)27 552 06 60
fax: +41 (0)27 552 06 69

www.imperix.ch
support@imperix.ch