UNIVERSITÀ POLITECNICA DELLE MARCHE

FACULTY OF ENGINEERING
MASTER DEGREE IN ELECTRONIC ENGINEERING

HW-SW cosimulation of FPGA-based sigma-delta controllers for triphase AC-AC matrix converters

Cosimulazione HW-SW di controllori sigma-delta basati su FPGA per convertitori trifase a matrice

Candidate:
Alessandro Fermanelli

Advisor:
Prof. Biagetti G.

Coadvisor:
Prof. Orcioni S.

Academic Year 2022-2023

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Faculty of Engineering
Master Degree in Electronic Engineering

HW-SW cosimulation of FPGA-based sigma-delta controllers for triphase AC-AC matrix converters

Cosimulazione HW-SW di controllori sigma-delta basati su FPGA per convertitori trifase a matrice

Candidate:
Alessandro Fermanelli

Advisor:
Prof. Biagetti G.

Coadvisor:
Prof. Orcioni S.

Academic Year 2022-2023

Università Politecnica delle Marche
Faculty of Engineering
Master Degree in Electronic Engineering
Via Brecce Bianche – 60131 Ancona (AN), Italy

Abstract

The work focuses on the implementation, both hardware and software, of a novel piloting technique for a class of AC-AC converters that goes under the name of 'Matrix Converters'. The strategy makes use of a well-established technology, the $\Sigma\Delta$ modulators, and their noise spreading ability. Moreover, the design served as a stress test for an open-source cosimulation environment, developed for VHDL users who wish to speed-up their prototyping phase.

Keywords: Matrix Converters, Sigma-Delta Modulators, Space Vector Modulation, Cosimulation, FPGA

Sommario

Il lavoro è focalizzato sull'implementazione, sia hardware che software, di un'innovativa tecnica di pilotaggio per una classe di convertitori AC-AC che va sotto il nome di 'Convertitori a matrice'. La strategia fa uso di una consolidata tecnologia, i modulatori $\Sigma\Delta$, e la loro abilità nel ridurre rumore di quantizzazione. Inoltre, il design è servito come stress test per un ambiente di cosimulazione open-source, progettato per chi sviluppa in VHDL con lo scopo di velocizzare i tempi di prototipazione.

Parole chiave: Convertitori a Matrice, Modulatori Sigma-Delta, Space Vector Modulation, Cosimulazione, FPGA

Contents

List of Figures

List of Tables

Acronym List

SPST	Single Pole Single Throw
DMC	Direct Matrix Converter
SVM	Space Vector Modulation
DPRAM	Dual Port RAM
RAMC	RAM Controller
CMB	Clock Modifying Block
MMCM	Mixed Mode Clock Manager
DAC	Digital to Analog Converter
ADC	Analog to Digital Converter
OD	Output Driver
XDC	Xilinx Design Constraints
IP	Intellectual Property
LOD	Leading One Detector
DSP	Digital Signal Processing
SDC	Synopsys Design Constraints
TCL	Tool Command Language
PS	Programmable System
PL	Programmable Logic
DDS	Digital Direct Synthesizer
CLI	Command Line Interface
SoC	System On Chip
COCO	Continuous Conversion
IRQ	Interrupt Request
ISR	Interrupt Service Routine
IPC	Inter Process Protocol
MSB	Most Significant Bit

LSB	Least Significant Bit
FP	Floating Point
FPAU	Floating Point Arithmetic Unit

Chapter 1

Introduction

The AC-AC power conversion is engaged in various market applications: lighting and heating control, online transformer tap changing, soft-starting, and speed control of pump and fan drives,[4] wireless power transfers [5], ship propellers, voltage restorers and more. To keep up with the various destinations, the AC-AC converters specialized and gave birth to different families. Some proposed topologies comprehend AC voltage controllers, cycloconverters and matrix converters. In particular this last one topology counts several pros like allowing independent sinusoidal modulation of output voltages and input currents, removing the need for a DC-link which impacts on the final weight and volume, enhancing the service lifetime, and, ultimately, controlling the power factor. But introduces some cons, for example the increased number of semiconductor switches (18 IGBTs/MOSFETs and 18 diodes) or the lower peak voltage-ratio ($\frac{\sqrt{3}}{2}$). In those occasions where trading weight for performance seems viable, the matrix converters look like an attractive alternative.
As the name implies, the main component in these converters is the matrix of switches, which connects every input phase to every output one. The technique that controls the matrix plays an important role on the final effects and results. Until now the most adopted solutions are the 'Alesina-Venturini' [6][7] technique and the 'Space Vector Modulation'[8][9] (abbreviated to **SVM**).
In this thesis a newly proposed modulation technique is explored. A solution based on $\Sigma\Delta$ [10] converters and their ability to spread quantization noise over a larger band than the signal. The advantages include a more straightforward implementation than SVM and a finer granulation since it can use all the possible combinations. Focus of this work is the realization of the controlling circuitry on an FPGA, in order to exploit its natural parallelism. A bare-metal software and host-software is furnished, to make the board more user-friendly by opening its serial port for desired instructions. Additionally, a stress test is provided for a newborn co-simulation environment able to run concurrently the PS (Programmable System) and the PL (Programmable logic).

Chapter 2

Preliminaries

The project involves AC-AC matrix converters, their piloting strategy, HW-FW-SW co-simulation and hardware design. Before diving into the core, a presentation and description of these matters is presented.

2.1 Direct Matrix Converters

The typical topology of an AC-AC power converter based on a matrix of switches is illustrated in the following figure:

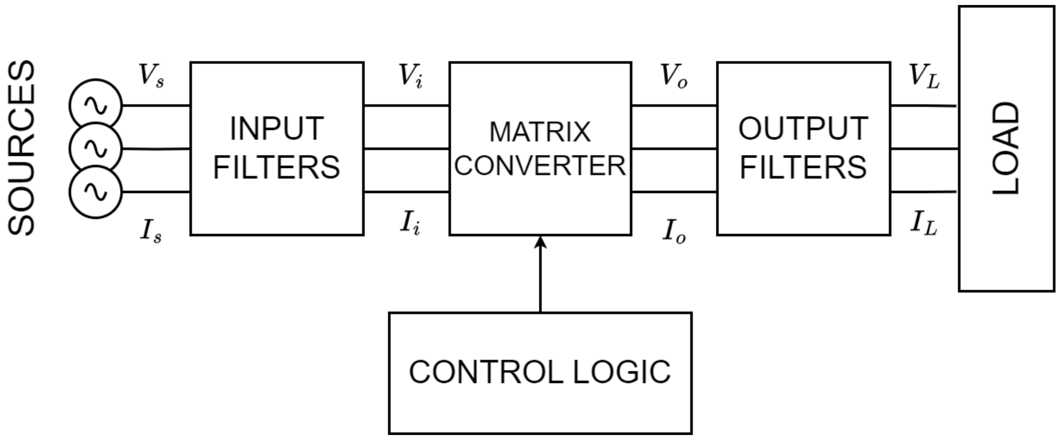


Figure 2.1: Circuit for a 3-phase AC-AC matrix converter.

An input filter is used to screen the supply system from current harmonics generated by the converter, while the output one acts as a low-pass filter to get the average value and eliminate high-frequency components due to switching of the matrix. Expanding on the converter unit, we get to see the matrix:

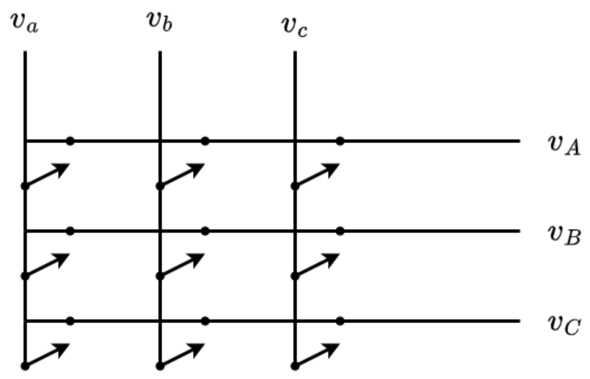


Figure 2.2: Circuit for a 3-phase AC-AC matrix converter.

Where v_a, v_b, v_c are the components of V_i and v_A, v_B, v_C are part of V_o.
From the above figure, it is possible to see that every output phase is connected to each input by a bidirectional switch, that allows flow from sources-to-load and vice-versa. The idea is to connect the output to one of the input voltages in a way that guarantees the desired frequency. This optimal connection should update at a faster rate than input frequency so that the error can exhibit a low magnitude.
The realization of the switches is an important matter too. They must act as a four-quadrant SPST (Single Pole Single Throw), which means they should drive currents of both polarities (when ON), and hold voltages of both polarities (when OFF). A variety of arrangements consisting of a pair of BJTs, or MOSFETs, or even IGBTs can achieve this functionality. The control logic depends on this choice, because drivers should follow a commutation procedure depending on switches' nature to avoid short-circuiting sources. For this project we are provided with nine bidirectional switches, composed by a pair of MOSFETS (IPAW60R600) and a silicon-integrated pair of diodes (fig. 2.3), connected by the drain.

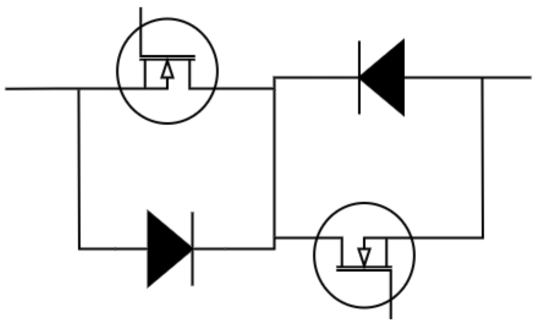
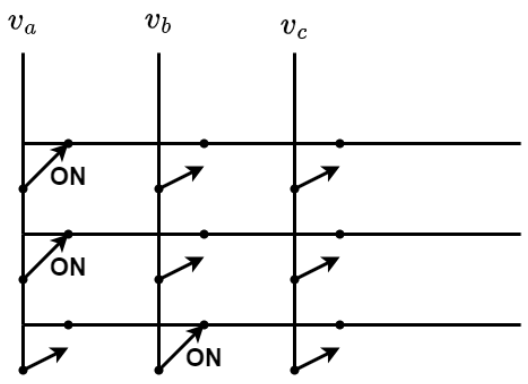


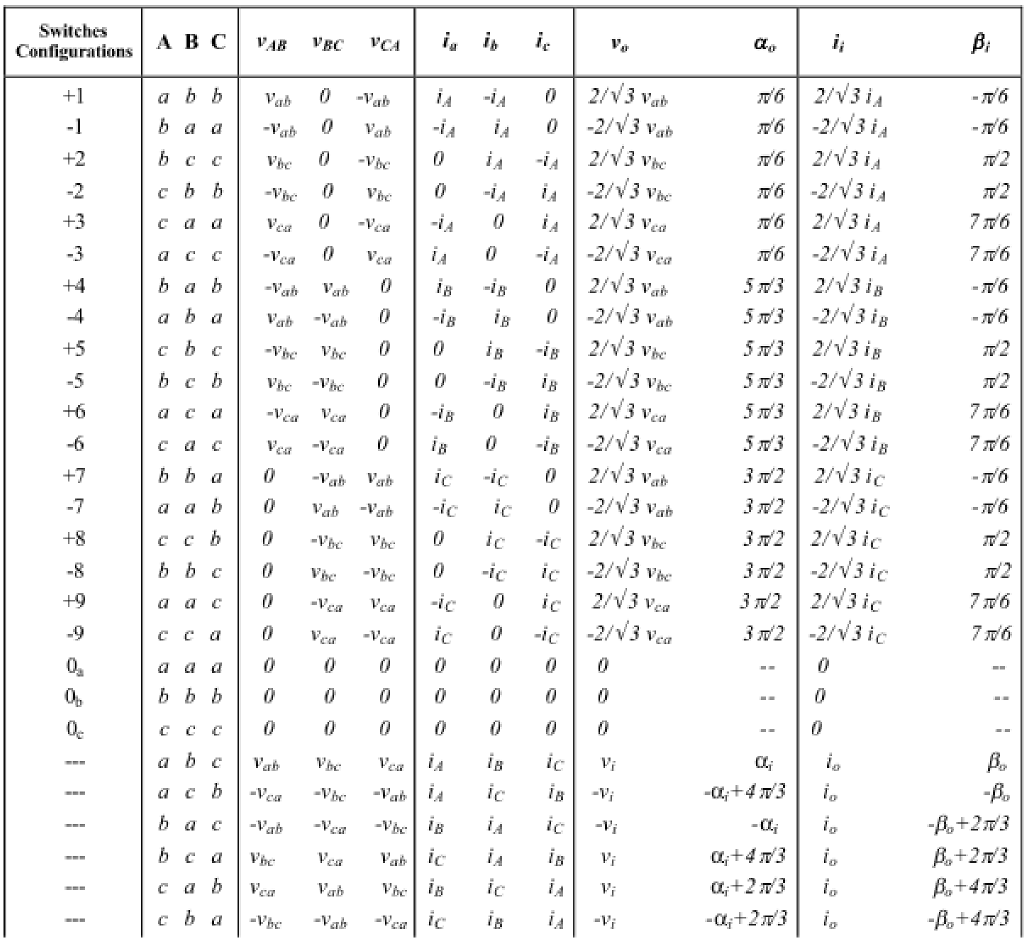
Figure 2.3: Bidirectional switch realization.

The possible switching configurations are 512 (2^9) but, in order to avoid the possibility of short-circuiting the inputs, each output can be linked to only one of the inputs, reducing the total to 27 permitted configurations. Each configuration is characterized by an associated matrix S_k, described in a way so that the presence of a '1' entry indicates the ON state of the switch, and a '0' entry means that the output phase and the input are not linked. For example,



$$\equiv \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.1}$$

The 27 combinations are ordered as in table 2.4, and will be indexed in a similar manner when designing the hardware.



Switches Configurations	A	B	C	v_{AB}	v_{BC}	v_{CA}	i_a	i_b	i_c	v_o	α_o	i_i	β_i
+1	a	b	b	v_{ab}	0	$-v_{ab}$	i_A	$-i_A$	0	$2/\sqrt{3}\, v_{ab}$	$\pi/6$	$2/\sqrt{3}\, i_A$	$-\pi/6$
-1	b	a	a	$-v_{ab}$	0	v_{ab}	$-i_A$	i_A	0	$-2/\sqrt{3}\, v_{ab}$	$\pi/6$	$-2/\sqrt{3}\, i_A$	$-\pi/6$
+2	b	c	c	v_{bc}	0	$-v_{bc}$	0	i_A	$-i_A$	$2/\sqrt{3}\, v_{bc}$	$\pi/6$	$2/\sqrt{3}\, i_A$	$\pi/2$
-2	c	b	b	$-v_{bc}$	0	v_{bc}	0	$-i_A$	i_A	$-2/\sqrt{3}\, v_{bc}$	$\pi/6$	$-2/\sqrt{3}\, i_A$	$\pi/2$
+3	c	a	a	v_{ca}	0	$-v_{ca}$	$-i_A$	0	i_A	$2/\sqrt{3}\, v_{ca}$	$\pi/6$	$2/\sqrt{3}\, i_A$	$7\pi/6$
-3	a	c	c	$-v_{ca}$	0	v_{ca}	i_A	0	$-i_A$	$-2/\sqrt{3}\, v_{ca}$	$\pi/6$	$-2/\sqrt{3}\, i_A$	$7\pi/6$
+4	b	a	b	$-v_{ab}$	v_{ab}	0	i_B	$-i_B$	0	$2/\sqrt{3}\, v_{ab}$	$5\pi/3$	$2/\sqrt{3}\, i_B$	$-\pi/6$
-4	a	b	a	v_{ab}	$-v_{ab}$	0	$-i_B$	i_B	0	$-2/\sqrt{3}\, v_{ab}$	$5\pi/3$	$-2/\sqrt{3}\, i_B$	$-\pi/6$
+5	c	b	c	$-v_{bc}$	v_{bc}	0	0	i_B	$-i_B$	$2/\sqrt{3}\, v_{bc}$	$5\pi/3$	$2/\sqrt{3}\, i_B$	$\pi/2$
-5	b	c	b	v_{bc}	$-v_{bc}$	0	0	$-i_B$	i_B	$-2/\sqrt{3}\, v_{bc}$	$5\pi/3$	$-2/\sqrt{3}\, i_B$	$\pi/2$
+6	a	c	a	$-v_{ca}$	v_{ca}	0	$-i_B$	0	i_B	$2/\sqrt{3}\, v_{ca}$	$5\pi/3$	$2/\sqrt{3}\, i_B$	$7\pi/6$
-6	c	a	c	v_{ca}	$-v_{ca}$	0	i_B	0	$-i_B$	$-2/\sqrt{3}\, v_{ca}$	$5\pi/3$	$-2/\sqrt{3}\, i_B$	$7\pi/6$
+7	b	b	a	0	$-v_{ab}$	v_{ab}	i_C	$-i_C$	0	$2/\sqrt{3}\, v_{ab}$	$3\pi/2$	$2/\sqrt{3}\, i_C$	$-\pi/6$
-7	a	a	b	0	v_{ab}	$-v_{ab}$	$-i_C$	i_C	0	$-2/\sqrt{3}\, v_{ab}$	$3\pi/2$	$-2/\sqrt{3}\, i_C$	$-\pi/6$
+8	c	c	b	0	$-v_{bc}$	v_{bc}	0	i_C	$-i_C$	$2/\sqrt{3}\, v_{bc}$	$3\pi/2$	$2/\sqrt{3}\, i_C$	$\pi/2$
-8	b	b	c	0	v_{bc}	$-v_{bc}$	0	$-i_C$	i_C	$-2/\sqrt{3}\, v_{bc}$	$3\pi/2$	$-2/\sqrt{3}\, i_C$	$\pi/2$
+9	a	a	c	0	$-v_{ca}$	v_{ca}	$-i_C$	0	i_C	$2/\sqrt{3}\, v_{ca}$	$3\pi/2$	$2/\sqrt{3}\, i_C$	$7\pi/6$
-9	c	c	a	0	v_{ca}	$-v_{ca}$	i_C	0	$-i_C$	$-2/\sqrt{3}\, v_{ca}$	$3\pi/2$	$-2/\sqrt{3}\, i_C$	$7\pi/6$
0_a	a	a	a	0	0	0	0	0	0	0	--	0	--
0_b	b	b	b	0	0	0	0	0	0	0	--	0	--
0_c	c	c	c	0	0	0	0	0	0	0	--	0	--
---	a	b	c	v_{ab}	v_{bc}	v_{ca}	i_A	i_B	i_C	v_i	α_i	i_o	β_o
---	a	c	b	$-v_{ca}$	$-v_{bc}$	$-v_{ab}$	i_A	i_C	i_B	$-v_i$	$-\alpha_i+4\pi/3$	i_o	$-\beta_o$
---	b	a	c	$-v_{ab}$	$-v_{ca}$	$-v_{bc}$	i_B	i_A	i_C	$-v_i$	$-\alpha_i$	i_o	$-\beta_o+2\pi/3$
---	b	c	a	v_{bc}	v_{ca}	v_{ab}	i_C	i_A	i_B	v_i	$\alpha_i+4\pi/3$	i_o	$\beta_o+2\pi/3$
---	c	a	b	v_{ca}	v_{ab}	v_{bc}	i_B	i_C	i_A	v_i	$\alpha_i+2\pi/3$	i_o	$\beta_o+4\pi/3$
---	c	b	a	$-v_{bc}$	$-v_{ab}$	$-v_{ca}$	i_C	i_B	i_A	$-v_i$	$-\alpha_i+2\pi/3$	i_o	$-\beta_o+4\pi/3$

Figure 2.4: Available switching configurations and their index. [1]

The figure is from [1], and is adapted to SVM nomenclature; in our case the indexes are 1 instead of +1, 2 in place of -1, 3 instead of +2, and so on until reaching 27. The last nine configuration names in fig. 2.4 assume a particular meaning in SVM: the 0_x are known as 'zero-states', and the remaining ones are referred as 'synchronous' configurations. For $\Sigma\Delta$ piloting, no configuration has a special role.

From figure 2.1, we can retrieve the general equations that describe the system:

$$V_o(t) = S_k V_i(t) \quad I_i(t) = S_k^T I_o(t) \qquad , S_k = \begin{bmatrix} s_{11,k} & s_{12,k} & s_{13,k} \\ s_{21,k} & s_{22,k} & s_{23,k} \\ s_{31,k} & s_{32,k} & s_{33,k} \end{bmatrix} \tag{2.2}$$

$$\sum_{j=1}^{3} s_{ij,k} = 1 \quad , i = 1,2,3 \quad k = 1,2,...,27 \tag{2.3}$$

This last equation is the mathematical description for short-circuit avoidance. Is a way for constraining every row to have just one '1' entry.

2.2 Alesina-Venturni Method

It has been demonstrated that matrix converters can achieve a maximum voltage gain of $\frac{\sqrt{3}}{2}$ ([7]). To reach its peak performance, various strategies have been adopted, one of these is the 'Alesina-Venturini' method.
Basically, three-phase output voltages are generated by sequential piecewise sampling of input waveforms. These output voltages follow a predetermined set of reference or target voltage waveforms by a transfer function approach. The equation for reference voltages is:

$$\begin{bmatrix} v_A \\ v_B \\ v_C \end{bmatrix} = V_{om} \begin{bmatrix} \cos \omega_o t \\ \cos(\omega_o t - 120°) \\ \cos(\omega_o t - 240°) \end{bmatrix} + \frac{v_{im}}{4} \begin{bmatrix} \cos 3\omega_i t \\ \cos 3\omega_i t \\ \cos 3\omega_i t \end{bmatrix} - \frac{V_{om}}{6} \begin{bmatrix} \cos 3\omega_o t \\ \cos 3\omega_o t \\ \cos 3\omega_o t \end{bmatrix} \tag{2.4}$$

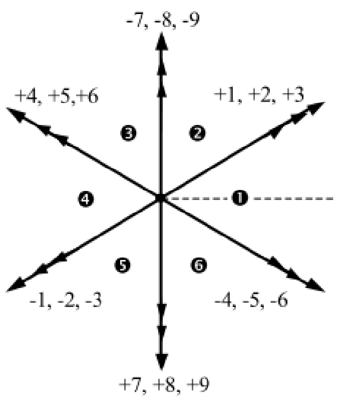
where V_{om} and v_{im} are the output and input magnitudes of the fundamental component, and ω_i, ω_o are the input and output angular frequencies.
It has been derived a general equation ([7]) that describes the optimal switching duty cycle for each switch. Considering that in every row there are three switches, the sequence proposed to achieve maximum performance is a simple 1-2-3, but the duty ratio of each one shall be determined by a precise function.

2.3 SVM Technique

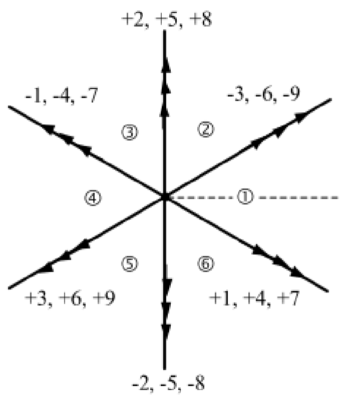
Space Vector Modulation is an established and well-known piloting technique for matrix converters. Based its success on some key aspects like achieving theoretical maximum output gain, a straightforward hardware implementation, a controllable power input factor, flexibility for application scope (reducing common mode voltage, output current ripple, switching losses etc.). Its derivation has roots on a transformation of triphase voltages in a space vector representation, where each sinusoid gets mapped onto a vector in a 2D space. Referring to the circuit topology in fig. 2.2, the mapping corresponds to,

$$\begin{aligned}
\vec{v_i} &= \frac{2}{3}(v_{ab} + \vec{a}v_{bc} + \vec{a}^2 v_{ca}) = v_i(t)e^{j\alpha_i(t)} \\
\vec{v_o} &= \frac{2}{3}(v_{AB} + \vec{a}v_{BC} + \vec{a}^2 v_{CA}) = v_o(t)e^{j\alpha_o(t)} \\
\vec{i_i} &= \frac{2}{3}(i_a + \vec{a}i_b + \vec{a}^2 i_c) = i_i(t)e^{j\beta_i(t)} \\
\vec{i_o} &= \frac{2}{3}(i_A + \vec{a}i_B + \vec{a}^2 i_C) = i_i(t)e^{j\beta_o(t)}
\end{aligned} \tag{2.5}$$

here $\vec{a} = e^{j\frac{2\pi}{3}}$. With equation 2.5, any of the 27 combinations can be associated with a vector. In fact, the rightmost columns in table 2.4 indicate the relative vectors (phase and magnitude). If we draw them on a 2D plane we get the characteristic hexagon,



(a)



(b)

Figure 2.5: Space vector for (a) output voltage and (b) input current. [1]

The SVM strategy requires to represent the desired output vector and project it onto the closest configuration vectors (example in 2.6). Each sector identifies six possible configurations, but input current constraints reduce them to four. By applying these four for a specific duty cycle (just like in Alesina-Venturini), we get a vector whose average value lies on the desired vector. With an additional filter at the output, we get the desired frequency by synthesizing the input ones. If the sum of the four duty cycles is not unitary, a 'zero-state' is applied for the remaining time.

Different variants have foundations on SVM: some utilizes only synchronous and zero configurations, others only synchronous or they can go under different names (like "Indirect SVM") because of a different approach.

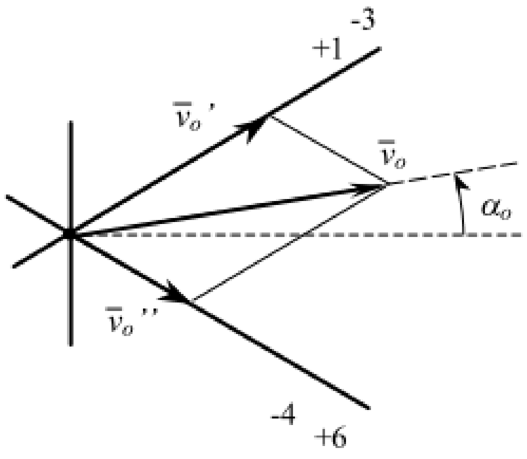


Figure 2.6: Projection of $\vec{v_o}$. [1]

2.4 Sigma-Delta Driving Technique

The focus of this work is to implement a different approach to control the matrix of switches, that relies on $\Sigma\Delta$ filters. This kind of digital filters are highly appreciated and used for the realization of ADCs and DACs, thanks to the exploitation of noise shaping functions and oversampling, but found new life in applications like LED driving or controlling switch-mode power supplies. The general scheme for a first order $\Sigma\Delta$ filter is reported as follows,

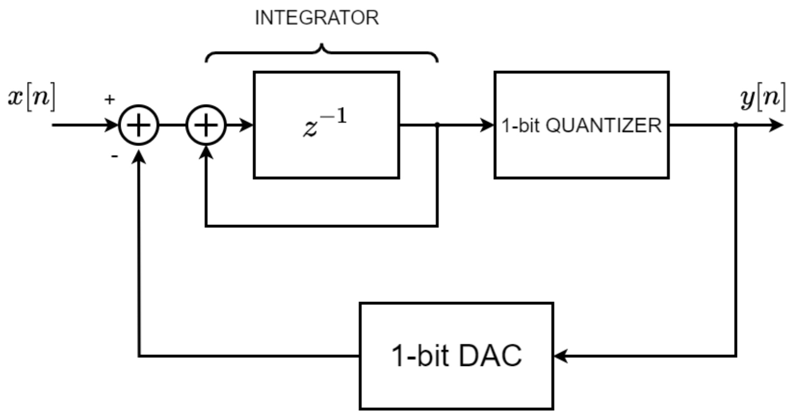


Figure 2.7: First Order Sigma-Delta modulator.

The actual implemented filter falls in a special category of sigma delta modulators, called 'Cascade of Integrators with feed-forward', or **CIFF**. The advantage (as explained in [11]) is making the "fast path" around the first integrator, so that its gain gets large and the noise and distortion added by successive stages are rendered small when compared to the input (figure 2.8).

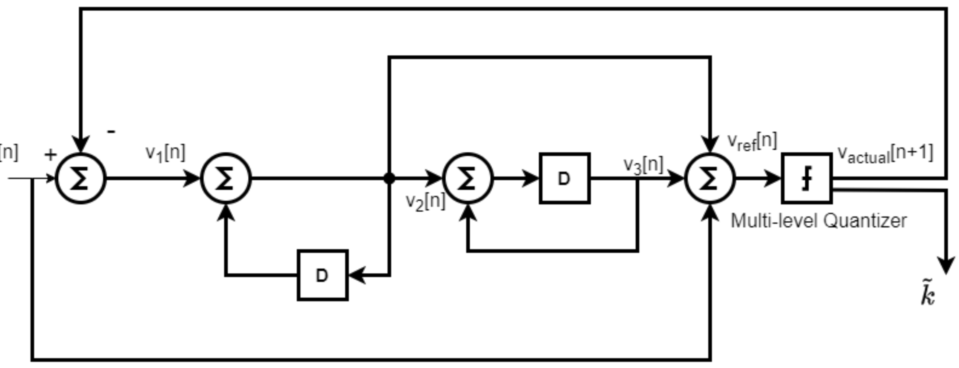


Figure 2.8: Second order CIFF modulator.

An important aspect of ΣΔ modulators is the working frequency, which should be several units higher than the input one. Without a high switching frequency, there can be no noise shaping of the input signal, thus no error mitigation. For the project a working frequency of 100 kHz is chosen, going higher would limit the available time for DSP calculations, making the timing constraints for the design really tight without a sensible performance boost as counterpart.
Speaking about performance, good metrics for comparing AC-AC matrix converters are the total harmonic distortion **THD** and the total harmonic distortion plus noise, **THD+N**. Comparing the ΣΔ approach with the traditional SVM, it is demonstrated ([10]) that utilizing the first one does have better THD results, and this should be expected since the modulator spreads the quantization noise over a large bandwidth. In terms of THD plus noise, both achieve similar results.
The modulation strategy inherits some aspects of a wider family of controlling techniques referred as *Discrete Methods Based on Predictive Model*[12], which revolves around a multi-objective cost function. In this case the cost function is expressed as:

$$C_k[n+1] = (\epsilon_k^v[n+1])^2 + (\epsilon_k^Q[n+1])^2 \tag{2.6}$$

where the k indicates one out of the 27 matrices, and the errors, indicated as ϵ_k^v and ϵ_k^Q, are described by:

$$\epsilon_k^v[n+1] = \frac{\|e_k^v[n+1]\|}{V_{des}+V_s} \quad , \epsilon_k^Q[n+1] = \frac{|e_k^Q[n+1]|}{Q_{des}} \tag{2.7}$$

with,

$$e_k^v[n+1] = V_{ref}[n] - V_o[n+1] \quad , e_k^Q[n+1] = Q_{ref}[n] - Q[n+1] \tag{2.8}$$

These values can be carried out at the current sample, noting that V_{ref} and Q_{ref} are outputs of the sigma delta modulators and $V_o[n+1]$ and $Q[n+1]$ can be referenced to n-th sample. By approximating in such a way,

$$V_o[n+1] = S_k V_i[n+1] \simeq S_k V_s[n+1] \simeq S_k V_s[n] \tag{2.9}$$

this is valid because input filter acts on high-frequency components of current, and $V_s[n+1] \simeq V_s[n]$ since the sampling frequency is at 100 kHz and sources' frequency way below (50 Hz for domestic lines). A similar approach can be taken on the input reactive power Q,

$$Q[n+1] = \Delta \frac{V_i[n+1]}{\sqrt{3}} \bullet I_i[n+1] \tag{2.10}$$

where $\bullet$ stands for scalar product, Δ is the matrix transformation for line-to-line voltages ($\Delta = \begin{bmatrix} 0 & +1 & -1 \\ -1 & 0 & +1 \\ +1 & -1 & 0 \end{bmatrix}$).

Under the hypothesis that $V_i[n] \simeq V_s[n]$ and $I_o[n] \simeq I_L[n]$, we get,

$$Q[n+1] \simeq \Delta \frac{V_s[n+1]}{\sqrt{3}} \bullet S_k^T I_L[n+1] \tag{2.11}$$

Since V_s and I_l are slowly varying compared to switching frequency, we rewrite,

$$Q[n+1] \simeq \Delta \frac{V_s[n]}{\sqrt{3}} \bullet S_k^T I_L[n] \tag{2.12}$$

Then (n+1)-th sample can be calculated at "time" n, making possible to compute the cost function (eq. 2.6).

FPGA's intrinsic parallelism suits the problem very well and can fill the role of matrix controller.

2.5 FPGA DEV Tools

When developing an FPGA, a design flow must be followed in order to deliver a functional product. To do so the FPGA producers make available a set of software tools to enable developers' skill and save them many profitable hours. An example of this care comes from Xilinx and its software toolset which includes Vivado and Vitis. The first one is dedicated to hardware designs and follows the workflow in every process: from writing RTL sources to instantiating IP modules, from behavioural simulation to post-synthesis/post-implementation simulation, from synthesis and routing/floor-planning to physical and timing constraints. If the intended design is an hardware-only one, then Vivado is more than a complete toolbox for the purposes. But in most occasions the design will have a bare-metal software which can enable more complex algorithms and functionalities. This is what Vitis was planned for. In fact, a design that includes a microprocessor, like the Microblaze or the Zynq, needs a software that can run with such hardware, and this can be written by the aid of Vitis which includes a text editor, C/C++ compilers, C/C++ linkers and more. The Vitis environment includes other stand-alone softwares like Vitis HLS, or Vitis Model Composer, that enables more rich features like High-Level-Synthesis or Simulink modeling, but for the scope of this thesis these are not used nor investigated.

Utilizing the Vivado or Vitis can be highly profitable, but at the same time prohibitive. Supposing the designer/engineer is familiar with the environment (which is not trivial), the amount of computations and data that can be driven by a simulation process or by a synthesis of a medium to large design, can sensibly reduce the productivity and bring stalling conditions (especially if the machine is not particularly powerful). The adopted solution for such matter of issues, was to use a cosimulator platform, open-source and free-to-use, developed by my relator Giorgio Biagetti [13]. In the next subsection the co-simulator is explained further.

2.5.1 The COSIM Platform

The platform serves for testing the cosimulation behaviour of the expected digital design, where real life aspects like synthesis/implementation constraints or RTL inference can be neglected to focus on simulation and functional verification. The platform is the result of a precise interconnection/intercommunication of available open-source tools, in particular QEMU and GHDL. The first one has the role to emulate the PS (programmable system) portion of the design and the latter to reproduce the PL (programmable logic). Technically speaking, the QEMU emulates an ARMv7 processor that runs the firmware, and attached to it, a QEMU module called **RTLbridge** exposes a user-configurable memory-mapped I/O region to the HW simulator and routes IRQs back from hardware to the CPU. In addition, an ad-hoc IPC (inter-process protocol) keeps in contact QEMU and GHDL to synchronize these two elements. The platform can also emulate the interaction with an external host, which runs a software with the ability to establish a serial communication. In this way cosimulation of HW-FW-SW is achieved and a functional prototype can be delivered. The downside to this approach, resides in the fact that a design working on the platform, cannot be "copy-pasted" but has do go under additional revisions to be compliant with the Vivado. Some of these revisions include adapting to the Xilinx RTL coding guidelines, directories re-configuration, rewriting the testbench and writing a new module at the highest hierarchy for connecting the VHDL entities with the IP allocated ones. Since a good portion of the thesis required working with the cosimulator, in the next chapters there will be clear references and results obtained by this platform.

Chapter 3

System Architecture

The project is built on three pillars: HW, FW and SW. The HW, on the digital part, makes advantage of the FPGA parallelism to run DSP calculations and generate control for the switches, while, on the analog part, gathers data from the sensors and filters high-frequency components. The FW portion setups the HW and opens a serial communication to accept instructions from hosts. The SW is the element which interacts with the final user and provides a simple CLI for debugging and managing hardware operations by opening the other end of the serial. In the next sections each one of these areas will be analysed.

3.1 Hardware Architecture

The hardware design for the COSIM environment differs from the one for Vivado, this is due coding changes (as explained in sec. 2.5.1) and IPs' encryption. The next subsections illustrates the design in COSIM, and in section 3.1.2 we illustrate the differences.

3.1.1 COSIM

The following figure shows the top-level diagram for the proposed architecture:

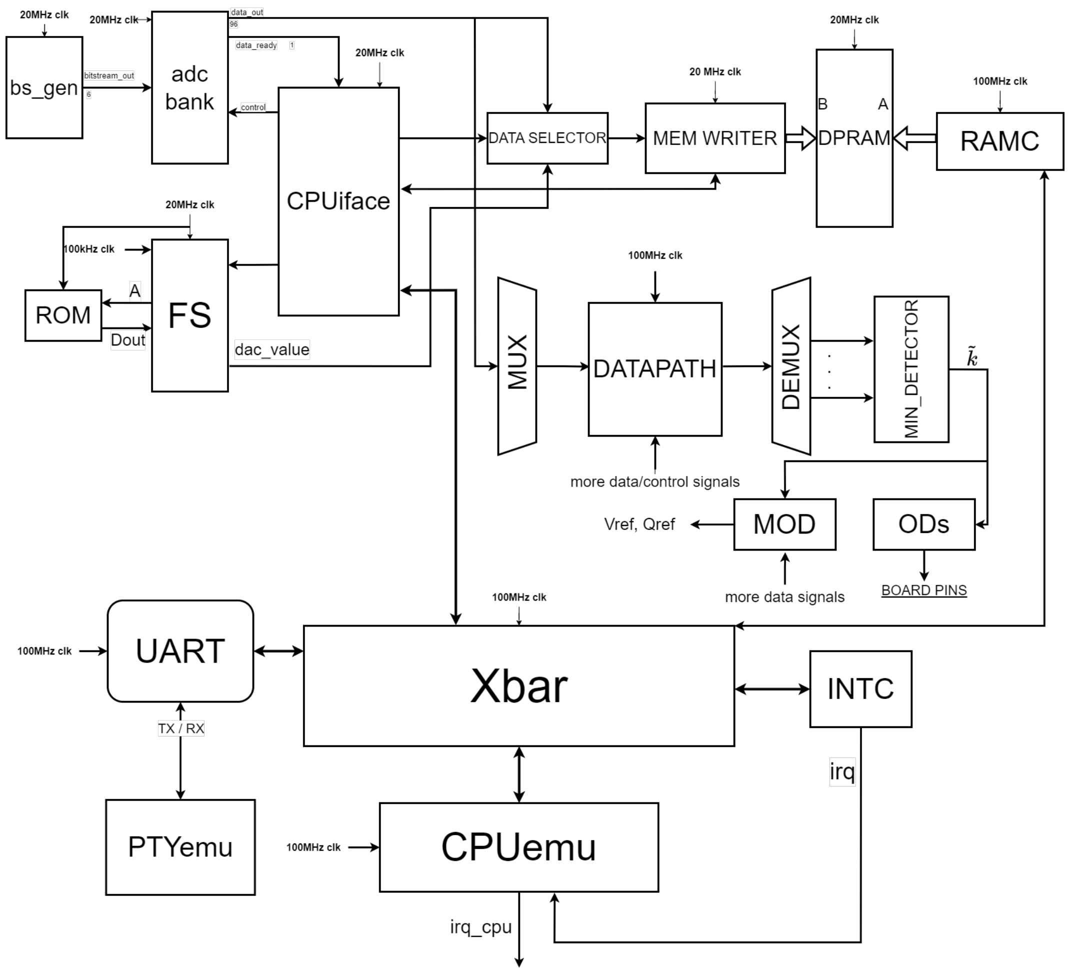


Figure 3.1: Top-level diagram of SoC.

Starting from clocks, four clock sources are generated from the system clock:

1. 100 MHz clock; fastest frequency to feed CPU, the Datapath, Minimum Detector and UART
2. 20 MHz clock; source for many blocks comprising ADC bank, MOD bank, CPUiface and more
3. 60 MHz clock; SPI interface

Some hardware uses more than one clock signal, introducing **Crossing Clock Domains**. They typically are a threat for the design, since they introduce the chance of metastability. Typical solutions are (ordered by complexity) sequences of flip-flops, FIFOs or handshake mechanisms. But considering that in Vivado the 100 MHz, becomes a 120 MHz, it is evident that each clock is a submultiple of this one, so that one crossing involving whichever clock, is always in phase, limiting the metastability chances.
The choice of creating clock toggles (like in ADC bank or multiplexers) rather than

generating an additional clock resource, is done for avoiding the design of a SoC with a numerous number of clocks and clock regions to manage, including their crossings.

A brief description of each module will help in understanding the signal flow.

The **ADC BANKS** is the interface to board sensors. Each one generates a bitstream at 20 Mbps which gets demodulated by a bank of $sinc^3$ filters, with the job to generate 16-bits words and decimate the input frequency to 100 kHz. Since on the COSIM platform the analog part can't be simulated, a simulation of bitstreams is reproduced with C++ code. It mimics the output of the real ADCs and writes the bitstream on a text file which gets treated as input for a VHDL module ('bs_gen') which reads the content and generates the bitstream. The input voltage ADCs can be emulated fairly easy (triphase sinusoids), but the output currents are not realistic, because their value depends on load and output filters, and so the analog part.

The **FS** and **ROM** realize the frequency synthesizer. To reproduce the required sinusoids, a *Direct Digital Synthesizer* (or DDS) uses the main clock at 100 MHz as a reference and utilizes the ROM as a look-up-table for sine values. The generic scheme for a DDS is reported in fig. 3.2.

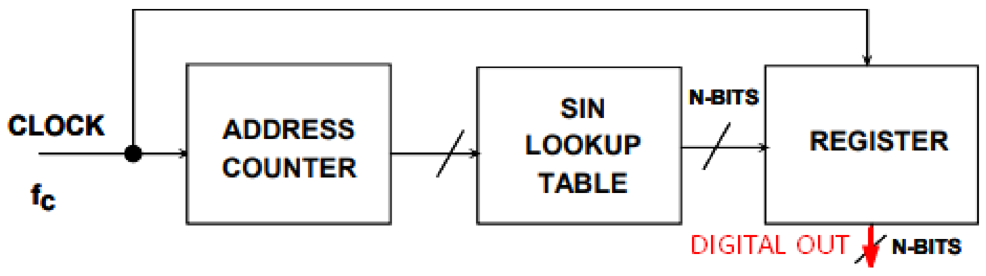


Figure 3.2: General scheme of a DDS. [2]

The **CPUiface** can be seen as a proxy for AXI addressing mechanism. Since a high amount of AXI slaves is not supportable and menaces severe performance degradation, the CPUiface embraces several AXI-addressable registers to redirect itself the content to the right modules. It also has the role to raise IRQs as a consequence of memory filling. Speaking of memory, the DPRAM can function in two modes, in a way called "COCO" (Continuous Conversion) where it gets overwritten by new data continuously and raises IRQs when half of its size is filled, or in a mode called "Snapshot" where it gets filled from the first to the last address, and upon completion raises an IRQ. The CPUiface has access to the highest bits and is acknowledged of the memory status as the system is running.

The **MODULATOR BANKS** are the sigma-delta modulators proposed for the new switching strategy. A total of four modulators are allocated: one for the input reactive power, and three for the input triphase voltages. The structure is that showed in 2.8, moreover includes an overflow compensation by saturation. Specifically, the sine waveforms in their peak reach a word value of ±31250, this means that $v_1 = v_{des} - v_{actual}$ can potentially reach double that value. In this sense a saturation at ±62500 limits the value span of integrator's output.

The **DATA SELECTOR** formats the data destined to the DPRAM. The packet generated assumes two forms depending on a user-configurable signal:

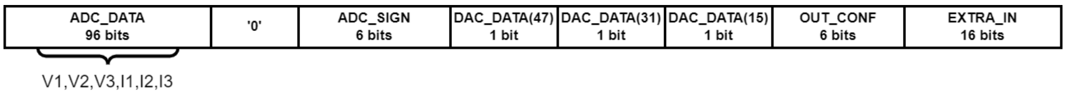


Figure 3.3: Format 1 (*selector='0'*).

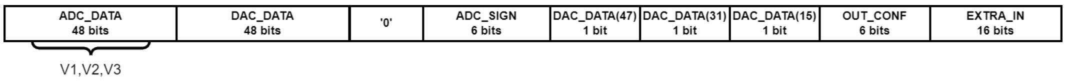


Figure 3.4: Format 2 (*selector='1'*).

The **out_conf** bits contain the actual voltages applied to the matrix; for example if the $\tilde{k}$ chosen is 4 then, referring to 2.4, the 6 bits that will be written on memory are "111010", corresponding to the encoding c="11", b="10" and a="01". This mapping is used all along the project.
The **extra_in** bits contain the 'pointed' memory location of the DPRAM.

In such a way, the memory can be consulted as a 'log' to follow, and eventually debug, the operations running on the SoC. In fact, memory data can be requested by the user that issues a specific command (more on 3.3).

The **MEMORY WRITER** is the module that actually writes the data that has been wrapped. The packets are 128 bits long, so write operation concludes after 4 cycles (32-bits writing/reading in DPRAM are allowed).

The **RAM CONTROLLER** is the access to the other port of the DPRAM. The FW, and the MicroBlaze in particular, can access logged data through this port. About the **DUAL PORT RAM**: nothing more than a memory, accessible from the SoC via *memory writer* and from the MicroBlaze via *ram controller*.

The **MINIMUM DETECTOR** outputs the configuration index, that will be associated to the corresponding matrix S_k, which is the minimum among the 27 possibilities. It works in a merge-sort fashion to locate the minimum among the outputs from the demultiplexers, with an associated time complexity of $\log_2 N$. In our case $N = 27$, so the module will have a latency of 5 clock cycles (fig. 3.5). Once the index has been found, the actual output voltage at the DMC terminals can be determined and so the input reactive power, making possible, for the $\Sigma\Delta$ modulators, to calculate their next output.

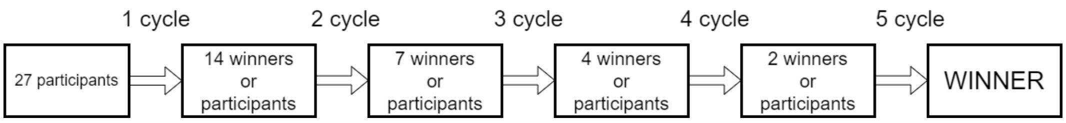

1 cycle
2 cycle
3 cycle
4 cycle
5 cycle
27 participants
14 winners or participants
7 winners or participants
4 winners or participants
2 winners or participants
WINNER

Figure 3.5: Merge sort to find minimum.

Looking again at the scheme of a traditional $\Sigma\Delta$ modulator (2.7), the role of the detector is the exact same of the 1-bit DAC. Thanks to this conclusion, it is absolutely right to name the detector as a "multilevel quantizer".

Strictly related to the detector, there is the **OUTPUT DRIVERS** which generate the control for the MOSFETS, with a precise commutation protocol to avoid possibilities of short-circuitation (fig. 3.6). More on the commutation problem and how to address it a good solution.
When switching from a phase to a new one there is the chance either to short-circuit the inputs (3.7a) or momentary open the load terminal (3.7b). Since both these situations must be prevented, a commutation sequence must be implemented. As [12] suggests, two strategies can be adopted, either *Output Current Direction Based Commutation* or *Input Voltage Magnitude Based Commutation*. The choice depends on what can be determined easily. For this case we opt for the current one, since it's sufficient to check the sign of the data-word generated by the ADCs.

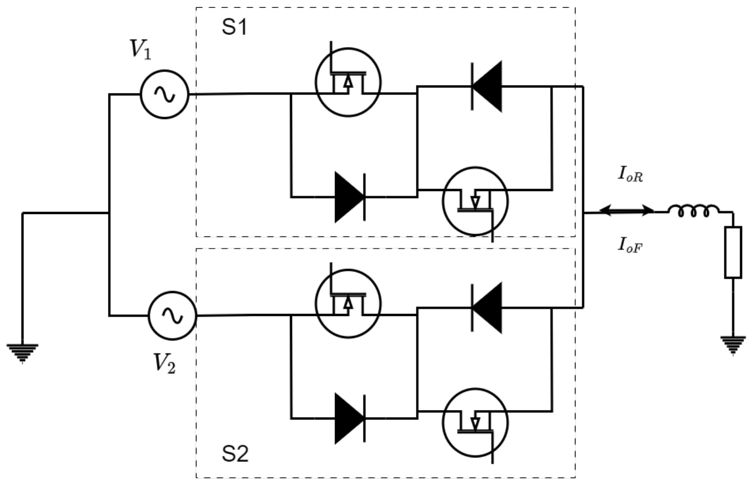


Figure 3.6: Example of connection for 2ϕ-1ϕ.

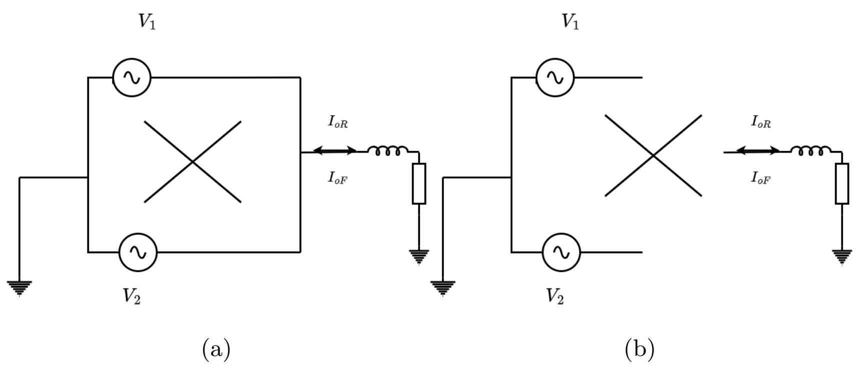


Figure 3.7: a. Short-circuitation and b. open terminals due to bad commutation.

Supposing we are switching from phase 1 to 2 in fig. 3.6, the protocol looks like this:

1. in S1, turn off the MOSFET which is not conducting
2. in S2, turn on the MOSFET that can sustain I_o
3. in S1, the other switch is turned off
4. in S2, turn on the other MOSFET

In this way, a safe switching can be achieved in 4 steps.

AXI Protocol

The inter-chip communication chosen is **AXI4Lite** part of the AMBA Standards, developed by ARM. The AMBA AXI has been designed for purposes of high-performance, high-frequency and high-speed. The advantages for adopting this standard are various:

- Separate address, control and data phases
- Support for unaligned data transfers
- Burst transactions
- Separate read and write data channels
- Easy addition of register stages to meet timing requirements

In addition, a bunch of interfaces' implementations are defined as Master-Slave, Master-Interconnect and Interconnect-Slave, which will be extensively used in current design. While going for the full feature AXI4 would have result in a higher complexity and usage of resources, the AXI4LITE seems a fine alternative. Trading a bit of performance for lightweight interfaces and communications, can be accepted since the design doesn't have a compromising frequency. In fact the weak point in terms of time is the DSP unit, but it has a working frequency way below the declared maximum.

Design Frequency	Maximum Frequency
120 MHz (cpu clock)	$363.77MHz < F_{max} < 628.93MHz$[DS181]

In the design an AXI Interconnect (or **AXI Crossbar**) is included to connect several slaves to a unique master, the MicroBlaze, which initiates a data transfer for reading or writing depending on the context. The AXI4Lite has 5 different channels: three for writing operations, two for reading ones. Each one implements signals for handshake and payload, and optionally a signal for data protection (not implemented for this design). The table shows all signals for an AXI4LITE interface:

Global	ACLK ARESETn	
Write Address Channel	AWVALID AWREADY AWADDR AWPROT	Master->Slave Slave->Master Master->Slave Master->Slave
Write Data Channel	WVALID WREADY WDATA WSTRB	Master->Slave Slave->Master Master->Slave Master->Slave
Write Response Channel	BVALID BREADY BRESP	Slave->Master Master->Slave Slave->Master
Read Address Channel	ARVALID ARREADY ARADDR ARPROT	Master->Slave Slave->Master Master->Slave Master->Slave
Read Data Channel	RVALID RREADY RRESP RDATA	Slave->Master Master->Slave Slave->Master Slave->Master

Table 3.1: Signals in AXI4Lite protocol.

The global signals should be coherent with master and slaves, and dictates the frequency and resets an interface (note the reset is active low). The signals indicated as xREADY and xVALID are the handshake signals (fig. 3.8), the xADDR or xDATA are the payload (for address phase or data phase), the xPROT are for protection, the xRESP is a feedback message which can indicate a successful or unsuccessful transaction.

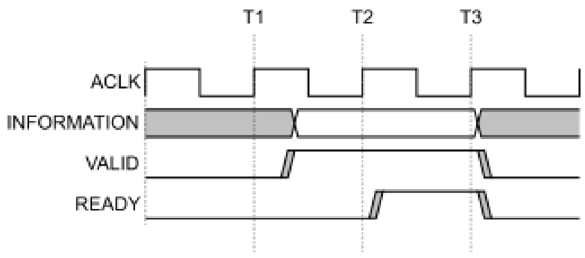


Figure 3.8: AXI handshake mechanism.

When utilizing an AXI Interconnect is mandatory to specify the address map. In practice, a master that initiates a communication looks up for an available device among those connected, and refers to it by its unique address region. This mapping is known both to the MicroBlaze and the Interconnect. In COSIM environment the mapping is listed below:

AXI Slave	OFFSET
UART data register	0x0000
UART control register	0x0004
INTC	0x1000
CPUiface	0x2000
RAMC controller	0x10000

Table 3.2: AXI address map.

The *OFFSET* is a constant value, specifically of 0xE0000000, dependent on the COSIM setup. In fact the addressable custom I/O can be accessed from this specific address.

More modules

For interrupt handling, an **INTERRUPT CONTROLLER** is provided, that can manage the interrupt sources and raise an IRQ flag to the MicroBlaze. In this design, two elements can ask for interruption of program flow: the UART, to communicate the state of the internal FIFOs (half, empty or available), and the CPUiface for memory status in *COCO* or *Snapshot* Mode. The MicroBlaze necessarily has to acknwoledge the IRQ, by clearing the flag bit and entering the equivalent *Interrupt Service Routine* (ISR). Once the exception gets resolved, the firmware continues from the interrupted point with its normal flow.

The **PTYemu** and **CPUemu**, as stated in sec. 2.5.1, are modules that make possible the inter-process-communication between QEMU and GHDL.

The **UART** provides serial communication. In particular will serve as an entry for user instructions typed on the CLI application. On Digilent board is present a USB-UART bridge (FTDI FT2232HQ) in order to facilitate communication with PC applications using standard Windows COM port commands. Xilinx provides an IP for implementing a simple UART, but its speed is limited to 115.200 kBd. Since the FT2232HQ can work at rates up to 12 MBd, an UART module has been implemented to reach such speed.

As last, the **DSP** completes the SoC. This is a module that works in between the $\Sigma - \Delta$ modulator and the quantizer/minimum detector. For each of the 27 configurations, the associated error shall be calculated (eq. 2.8), in doing so ADC data undergoes integer multiplication, sum and floating-point arithmetic. Before deep diving into the algorithm mapped on hardware, some considerations should be tackled. The design of this portion follows a particular paradigm called **time multiplexing** or **resource sharing**. Since the 27 configurations require the same equation, hence same hardware, there is no sense in allocating different modules that do the exact same thing (an exception are those designs were resource usage is not a problem and time is tightly constrained). But still we want to make use of parallelism, so the solution is in between. Three identical modules have been allocated, each of these distributes over time 9 of the 27 possible solutions (fig. 3.9).

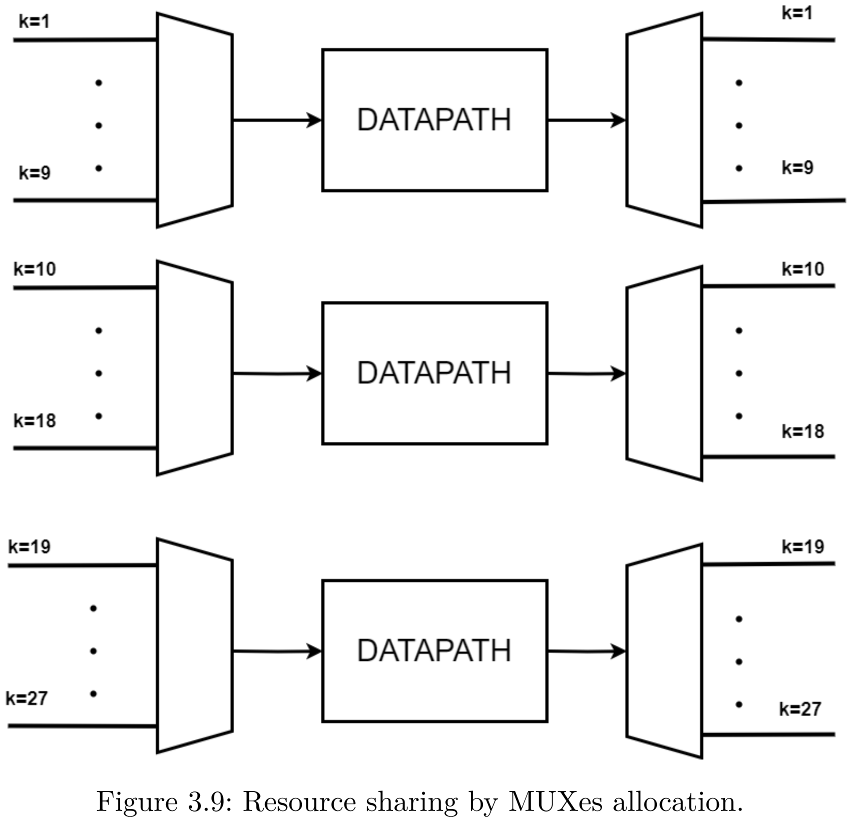


Figure 3.9: Resource sharing by MUXes allocation.

Their input comes from multiplexers with an internal toggle at 2 MHz, which rules the frequency for switching from a certain configuration index to the next one (fig. 3.10). The demultiplexers have an identical mechanism.

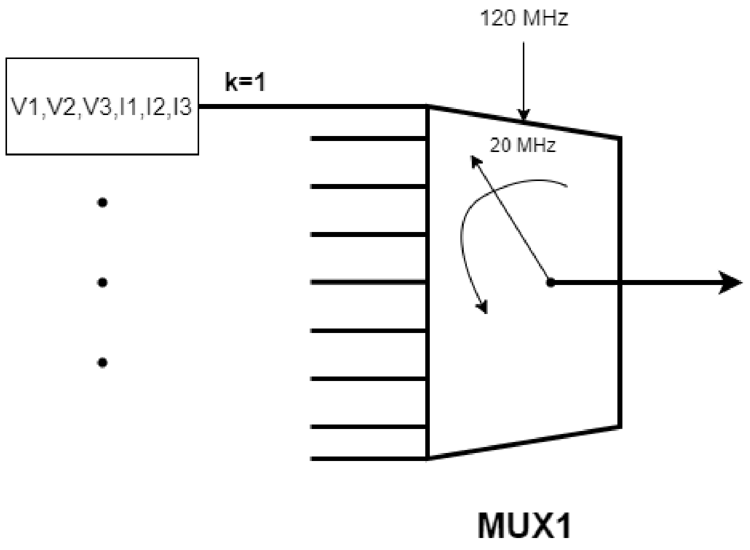


Figure 3.10: MUX and its toggled switching.

The choice for a 2 MHz switching frequency originates from a tradeoff conclusion. By inspecting the waveform analyser can be estimated how much time the DSP has left for solving the computations, and this value is around 8250 ns. With the displayed time-multiplexing mechanism in mind, the following equation sets the lower bound for the switching frequency:

$$\frac{9}{F_{clk\_mux}} < 8250ns \quad or \quad F_{clk\_mux} > 1.0909MHz \tag{3.1}$$

Rounding to the closest integer, we gain 2 MHz. Going at a faster speed not only is not necessary, but also can implicate a timing failure. In fact the datapath unit works at 100 MHz, this means that has 50 clock cycles to complete operations in case of 2 MHz switching, and 10 cycles for a 10 MHz. The choice of 2 MHz has another advantage, provides additional time for quantizer operation and output driving. The left time can be calculated:

$$8250ns - 9T_{clk\_mux} = 3750ns \tag{3.2}$$

The multiplexed input has to undergo data manipulation. The algorithm mapped is shown next,

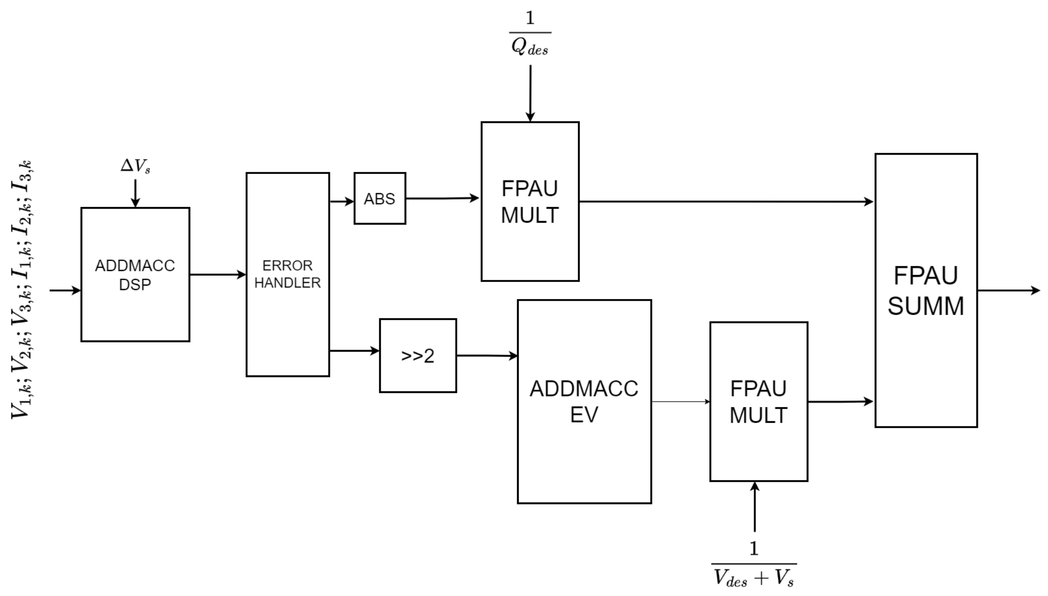


Figure 3.11: Datapath insight.

Some modifications can be made, for example removing the scaling factor of $\sqrt{3}$ won't affect algorithm results, the matrix multiplication can be rewritten and so the input currents which depend on output ones. This gets us to,

$$Q[n+1] \simeq \begin{bmatrix} v_{s2} - v_{s3} \\ v_{s3} - v_{s1} \\ v_{s1} - v_{s2} \end{bmatrix} [n] \bullet S_k^T i_L[n] \tag{3.3}$$

The module that does this computation is the **ADDMACC DSP**. A further note on the functioning of the datapath: the data travels accompanied by an *enable* impulsive signal and a *done* one (fig. 3.12). The first one informs the entity that data arrived and has to be processes, the latter one signals the availability of the result and is connected to the enable of the next block expected on signal flow. This paradigm also enables a pipelined approach, that has not been exploited in the project, since the timing is achieved.

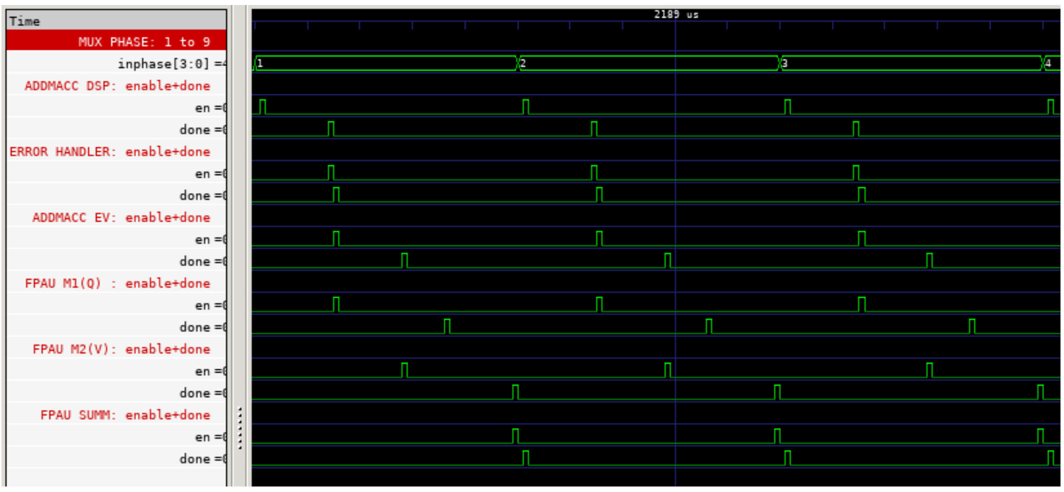


Figure 3.12: Chain of control signals in Datapath.

The *ADDMACC DSP* result is passed to **ERROR HANDLER** which generates the errors between ΣΔ modulators and the quantized result,

$$e_k^Q[n+1] = Q_{ref} - Q[n+1] \quad , \mathbf{e}_k^v[n+1] = \mathbf{V}_{ref} - \mathbf{V}[n+1] \tag{3.4}$$

(The boldness indicates vectors.)

The reactive power error goes into the Floating Point Arithmetic Unit (**FPAU**) for multiplication by the scaling factor (normalized error),

$$\epsilon_k^Q[n+1] = \frac{|e_k^Q[n+1]|}{Q_{des}} \tag{3.5}$$

while the voltage errors are used for calculating their magnitude (inside the **ADDMACC EV**) and, successively, scaled for normalization,

$$\epsilon_k^v[n+1] = \frac{\|\mathbf{e}_k^v[n+1]\|}{V_{des} + V_s} \tag{3.6}$$

The final step implements a sum for floating points, which will be the input for the quantizer.

Those blocks with *ADDMACC* as prefix allocates one DSP48E1 unit in an accumulator functionality. In fact Vivado offers the possibility to use the so-called **Templates**, which enable the direct instantiation of primitive elements (like the DSP) optimized for a particular macro/functionality (fig. 3.13)[UG479]. This grants higher control on resources and on the inference at synthesis-level.

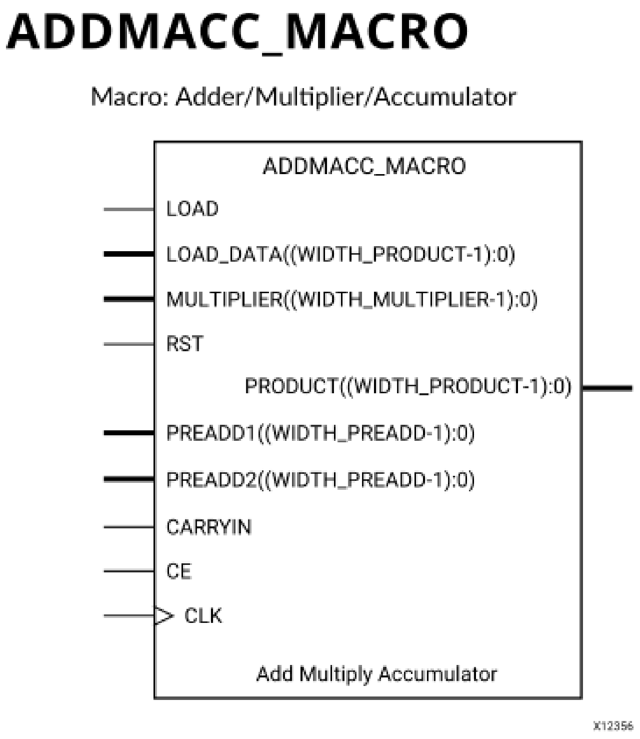

ADDMACC_MACRO
Macro: Adder/Multiplier/Accumulator
ADDMACC_MACRO
LOAD
LOAD_DATA((WIDTH_PRODUCT-1):0)
MULTIPLIER((WIDTH_MULTIPLIER-1):0)
RST
PRODUCT((WIDTH_PRODUCT-1):0)
PREADD1((WIDTH_PREADD-1):0)
PREADD2((WIDTH_PREADD-1):0)
CARRYIN
CE
CLK
Add Multiply Accumulator
X12356

Figure 3.13: DSP48E1 Template. [UG479]

Floating Point Arithmetic Unit

Also the FPAU uses a DSP in such a way. In fact for floating point multiplication a wide-multiplicator (32x32 bit) is required, and the DSP48E1 is not sufficient (15x28 bit implemented), but still can be used in *ADD MACC* mode by employing the Karatsuba-Ofman algorithm [14] and some basics arithmetic. To prove this the diagram for multiplication algorithm is illustrated in fig. 3.14.

The floating point values mentioned refer to single-precision (32 bit) IEEE-754 standard format. This design lacks overflow/underflow detection that can happen when multiplying/adding two floating points. But we will demonstrate that operands magnitude is not subject to this error, still can be material for future work.
In order to justify the overflow/underflow chance, a short discourse on word-widths has to be made. In the following table, the bit size of most important quantities is displayed.

SIGNAL	q_{adc}	q_{act}	e_q	e_{vx}	$\|e_v\|$
PIN	OUT:ADDMACC_DSP	OUT:ADDMACC_DSP	OUT:ERR_H	OUT:ERR_H	OUT:ADDMACC_EV
WIDTH	signed 35bit	signed 16bit	signed 20bit	signed 20bit	signed 37bit

The size depends upon the equation in which each variable is involved. For example, the q_{adc} for k=1 is,

$$q_{adc,1} = (v_b - v_c)(i_A) + (v_c - v_a)(i_B + i_C) \tag{3.7}$$

where, $v_b - v_c$ is a 17 bit signed and i_A or $i_B + i_C$ are 17 bits signed. So the size of q_{adc} is justified, but at this point a clarification can be made. Since the input reactive power is one of the inputs for the $\Sigma\Delta$ modulators, it has to be resized to a dimension of 16 bits with sign. To do so the 16 MSBs are taken into account and this takes us to q_{act}. The e_{vx} go under a similar manipulation. Since the DSP48E1, inside the 'ADDMACC_EV', provides 18x25 multiplications, the e_{vx} gets right-shifted to a size of 18 bits.

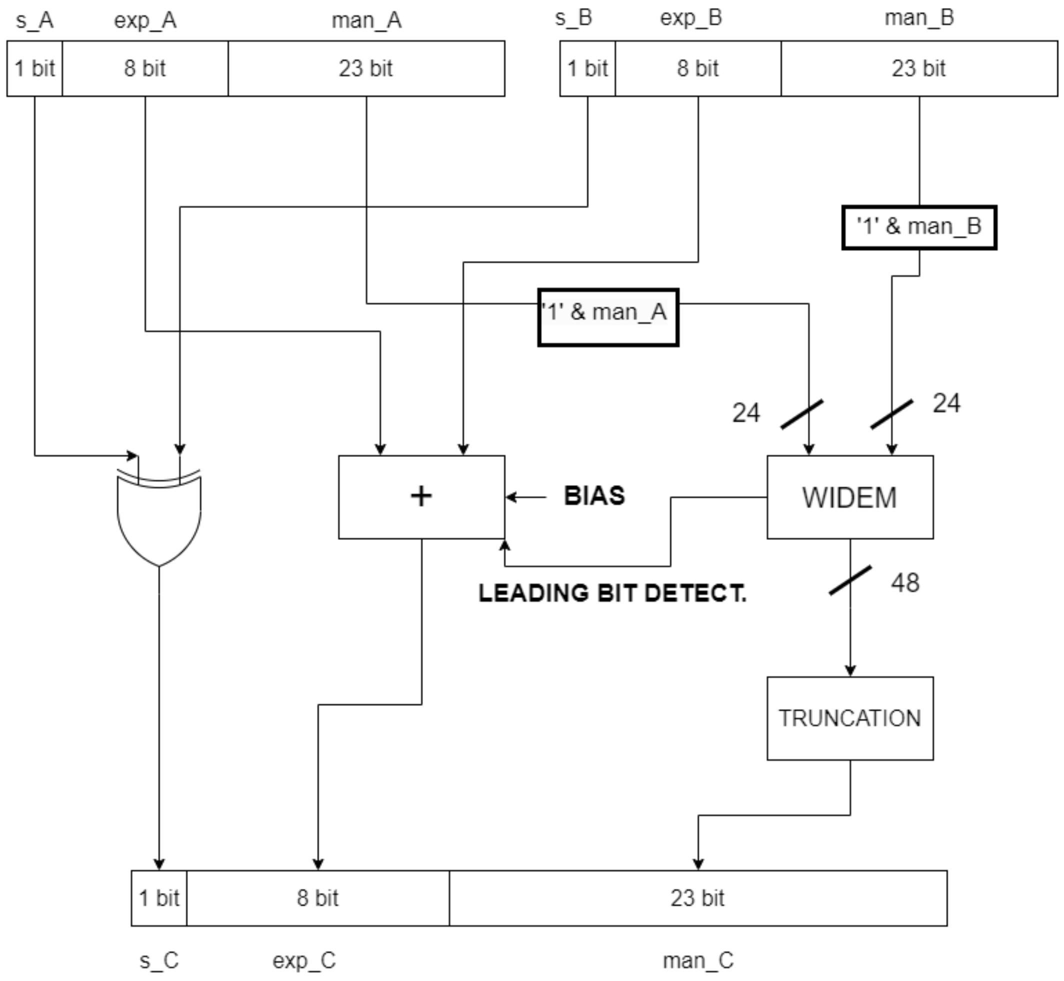


Figure 3.14: Floating point multiplication algorithm.

These clarifications were necessary to understand the magnitudes that come into play with the FPAUs. For example, the multiplier with inputs $\frac{1}{Q_{des}}$ and e_q won't cause an overflow/underflow. In fact, the desired input reactive power is obtained from,

$$Q_{des} = 2\pi f_s 3V_s^2 C \tag{3.8}$$

if we want to deny that generated by input filters. Supposing an input frequency of 50 Hz, a source amplitude in the range of 1-200 V, and an input filter capacitance of 13.2µF [10]; then, considering that e_q has a maximum absolute value of 2^{19}, we get,

$$2.01e^{-3} < \frac{1}{Q_{des}} < 80.422 \tag{3.9}$$

and thus,

$$0 < |e_q \frac{1}{Q_{des}}| < 84.3285e^{6} \tag{3.10}$$

which is fully contained in IEEE754 single-precision range. A similar approach can be showed for the other multiplier.

Turning back to the FP multiplication algorithm. The 24x24 multiplication involves only one DSP48E1. In fact, considering 2k-bits inputs, say X and Y, their value can be decomposed as,

$$X = 2^k X_1 + X_0 \quad , Y = 2^k Y_1 + Y_0 \tag{3.11}$$

where X_1 and Y_1 are the most significant bits, and X_0 and Y_0 the least significant ones. Then the product can be written as,

$$XY = (2^k X_1 + X_0)(2^k Y_1 + Y_0) = 2^{2k} X_1 Y_1 + 2^k (X_1 Y_0 + Y_1 X_0) + X_0 Y_0 \tag{3.12}$$

This already shows that a wide-multiplication can be decomposed in steps of 'narrow' ones. A further modification if we use Karatsuba-Ofman algorithm,

$$X_1 Y_0 + Y_1 X_0 = X_1 Y_1 + X_0 Y_0 - D_X D_Y \tag{3.13}$$

where $D_X = X_1 - X_0$ and $D_Y = Y_1 - Y_0$. This algorithm made possible to rewrite the middle term from 3.12 in a way that suits the *ADD MACC* functionality of the primitive unit (tab. 3.3 shows performance).

HW usage	**Latency**	**Frequency**
1xDSP48E1, 1xSplitter, 1x32bit Right Shift, 1x16bit Right Shift, 2x32bit Adder	14	100 MHz (120 for Vivado)

Table 3.3: FP Multiplication performance.

The FPAU adder design can be found in fig. 3.15. The algorithm is easier than multiplication and requires less units.

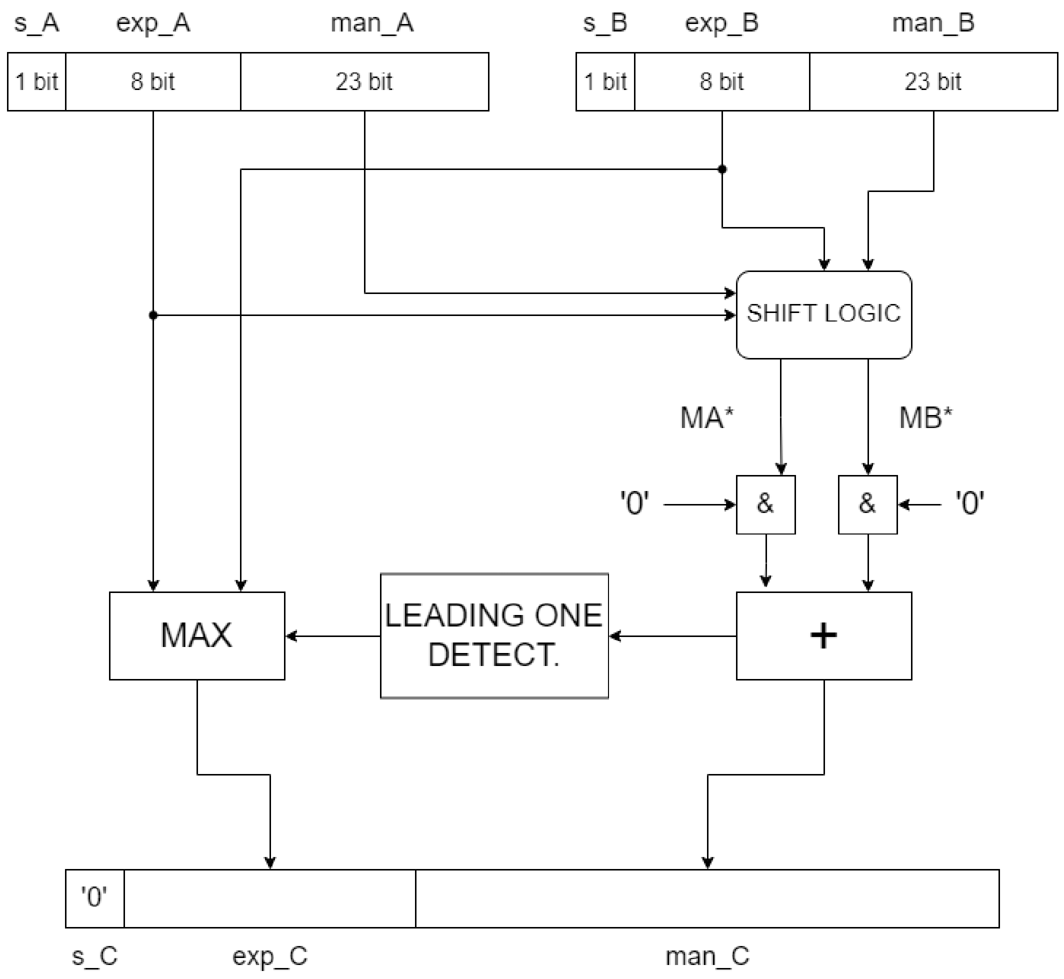


Figure 3.15: Floating point sum algorithm.

HW usage	Latency	Frequency
1x32bit Right Shift, 2x32bit Adder, 1x8bit Comparator	2	100 MHz (120 for Vivado)

Table 3.4: FP Sum performance.

An additional clarification has to be detailed. The FPAUs accept as operand two **std_logic_vector** of 32 bits each, but while the operands coming from the MicroBlaze, the $\frac{1}{Q_{des}}$ and $\frac{1}{V_{des}+V_s}$, are represented in IEEE-754 format, the other operands have to undergo a reformatting. Specifically these two are signed numbers in a 2's complement representation, and have to be 'casted' in their single-float equivalent. The IEEE library, *float_pkg*, contains a method for this casting, but doesn't work at the speed of 120 MHz. So an additional module has been designed for this purpose. The circuit that helps fix the issue, is a **LEADING ONE DETECTOR**, which

output is used to shift appropriately the bits into mantissa. The detector works in a pipelined manner and its design is reported in fig. 3.16. The 32-bit version here implemented allocates multiple times an elementary unit that works with 4-bit inputs (fig. 3.17). The output will be a *std_logic_vector* filled with zeroes, except for the position of the leading one bit. An additional multiplexer, which adds another cycle latency, generates the corresponding integer that informs the FPAU on the number of bits that have to be included in mantissa.
The pipeline stages are there to guarantee timing closure and establish a determined latency of 4 cycles (3 for pipelined LOD and 1 for multiplexing) that will be acknowledged by subsequent modules, for a coherent functioning. This latency has to be added for floating pint multiplier and adder.

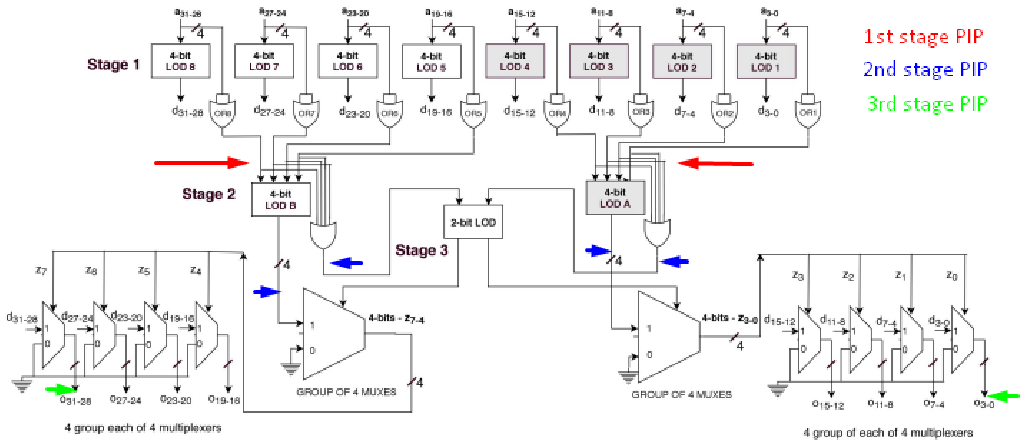


Figure 3.16: 32-bit Leading One Detector. [3]

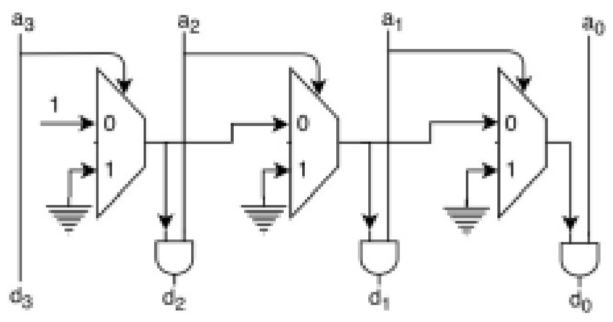


Figure 3.17: 4-bit LOD, basic block.

3.1.2 Vivado

As explained before, the Vivado comes with a bunch of hardware blocks ready to use and relatively easy to implement through a grapical interface. For example when designing memory units, is good practice to allocate them with the relative IP, "Block

Memory Generator", which is platform-wise, a better solution than generating VHDL. As for memory, other blocks get substituted in such manner. To be thorough, the following list details the changes:

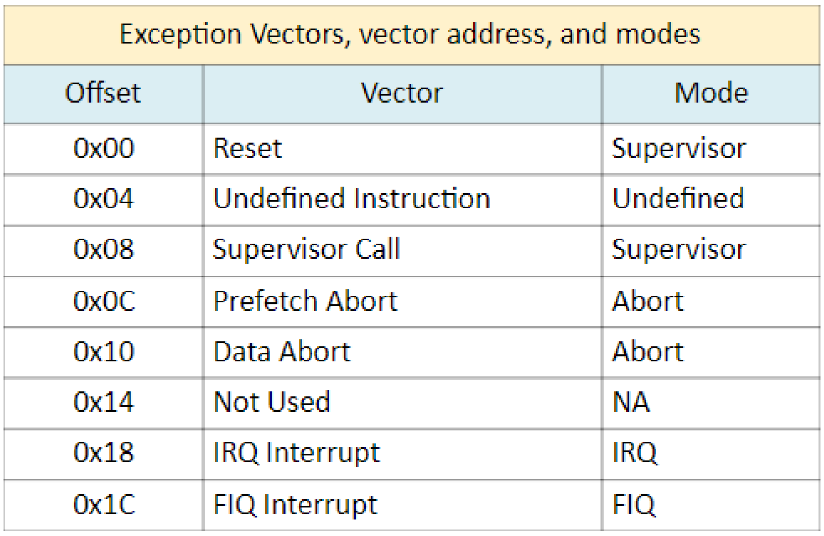
COSIM	VIVADO
testbench and QEMU clock	Clocking Wizard
testbench and QEMU reset	Processor System Reset
Dual Port RAM	Block Memory Generator
ROM (sine LUT)	Block Memory Generator
AXI Xbar	AXI Interconnect
intc	AXI Interrupt Controller
ARMv7	MicroBlaze

The biggest change regards the microprocessor. In COSIM environment, the QEMU simulates an ARM Cortex-A9, while on Vivado the processor is a MicroBlaze, which is a RISCV architecture. Some blocks are present in both designs and get re-interpreted, other are added only in Vivado. Blocks like the SPI, the I2C, the local memory buses and the GPIO are present only in Vivado's design since implementing such modules on the COSIM would have been unnecessary. The Vivado is the closest step to FPGA programming, this motivates the addition of modules with IO connections to communicate with other integrated circuits present on the Digilent Board. To integrate the IP portion of the design and the VHDL described one, Vivado allows a feature called 'HDL Wrapper' which -as the name states- makes the proprietary blocks enclosed in a VHDL entity. In such a way is possible connect the various modules, reaching the final schematic.

3.2 Firmware Development

The finished hardware would be useless without some bare-metal software, so a firmware has been coded for the MicroBlaze. This IP element is referred also as *Soft Processor*, to underline difference with *Hard Processors* that are physically realized and not hardware-programmable. It is a 32-bit processor based on **RISC** architecture with a bunch of interfaces, like UART, SPI, I2C or GPIO, and an interrupt controller. It comes with a bunch of advantages like flexibility, low-power design, low resource usage (close to 1% for Artix 7) and royalty-free material. It can be used in three application-specific configurations, specifically as microcontroller, real-time processor or application processor. For our case the first mode is more than enough for hosting bare-metal software. Can be designed with Vitis, part of the Vivado toolset, which comes with an important feature that allows to import the designed hardware and write firmware ad-hoc with the generated peripherals and interconnections. As for the hardware, firmware written for COSIM or for Vivado differs by some changes. In particular transition from RISCV ISA to ARM is required.
The job of firmware is to setup the PS, open the UART end for serial communication

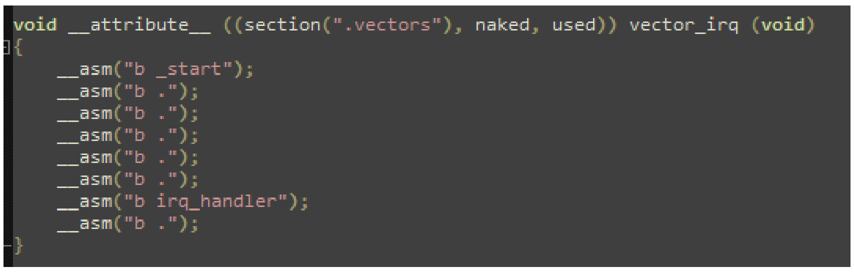
and reply to user instructions appropriately. This can be achieved with AXI4Lite transactions and memory-mapped slaves. Additional headers to main program are provided for register mapping, irq handlers and UART management. Talking about IRQs, below is showed the ARMv7 exception vector:



Exception Vectors, vector address, and modes		
Offset	Vector	Mode
0x00	Reset	Supervisor
0x04	Undefined Instruction	Undefined
0x08	Supervisor Call	Supervisor
0x0C	Prefetch Abort	Abort
0x10	Data Abort	Abort
0x14	Not Used	NA
0x18	IRQ Interrupt	IRQ
0x1C	FIQ Interrupt	FIQ

Figure 3.18: Exception vector for ARMv7.

This information should be included in main code, so that it knows where to start and how to handle exception calls. This is done by writing some assembly code inside a key function (fig. 3.19). Thanks to compiler-specific attributes, this function can be placed in a well defined section rather than .text .
So when an IRQ, the function redirects control to a generic ISR, that checks the raised flag and branches to correspondent routine.



```
void __attribute__ ((section(".vectors"), naked, used)) vector_irq (void)
{
    __asm("b _start");
    __asm("b .");
    __asm("b .");
    __asm("b .");
    __asm("b .");
    __asm("b .");
    __asm("b irq_handler");
    __asm("b .");
}
```

Figure 3.19: Key function for exception handling.

3.3 Software for User Interface

User can interact with SoC by a command-line interface written in C. In COSIM, software and hardware can communicate through the PTYemu, which is a module that creates a 'pty' file that can record UART transactions. An illustration of this mechanism is showed:

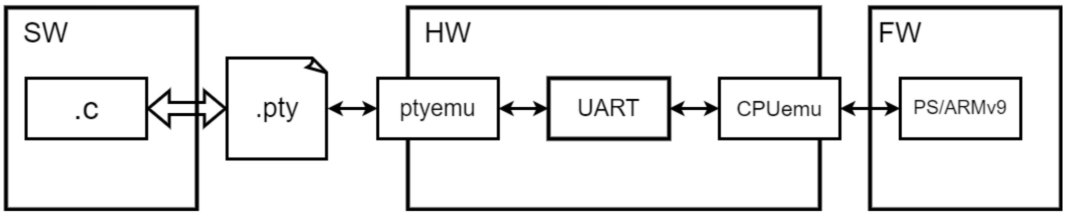


Figure 3.20: Communication protocol in COSIM.

The C code writes/reads on/from ’pty’ as any normal file. The communication is initiated by software which issues a command to firmware, and it replies on the basis of the raised command. A list of available instructions and relative options is provided.

COMMAND	OPTIONS	DESCRIPTION
help	//	lists the available commands
register status <REG>	REG=ADC,DAC,FREQ	displays content of internal registers
memory	[-n <NUM_BYTES>] [-o <OFFSET>]	shows a chunk of DPRAM from OFFSET address
write	-f <FREQ_DES> or -q <Q_DES> or -a <AMPL_DES>	changes the desired sinewave or Q_{des}

Every command gets pre-processed in SW, in order not to involve the processor at every passage. For example the **help** command can be executed without sending UART data, and can be elaborated locally to the host.
The messages exchanged are encapsulated in a packet which helps avoiding spurious transaction due to FIFO buffers, or noise sources. The sent messages (SW->FW) are started and ended by a characteristic sequence (0x01), while the responses (SW<-FW) are started by the same character but have no end. This difference happens because the SW side knows how many bytes/characters to expect from the PS, so has to detect the initial one and start counting from there. This solution does not solve unexpected communications started by 0x01, its success is statistics-dependent.

Chapter 4

FPGA Design

4.1 UltraFast Methodology Approach

The adopted FPGA development strategy is a crucial point for the realization of a valid product. For this project, the **Ultrafast Methodology Approach** proposed by Xilinx has been followed [UG949]. What's appealing about it is the highly particularized chain of development and the time-optimized cycle. It is nothing more than a collection of good practices derived from the collective engineering experience, which can be used as a good representation of the big picture in FPGA development. It consists of 5 milestone:

1. Board and Device Planning
2. Design Creation with RTL
3. Design Constraints
4. Design Implementation
5. Design Closure

Since an already available board has been employed for this project, the "Board and Device Planning" step can be skipped, but not neglected since this will affect upcoming design choices.

4.2 Board and Planned I/O

The board containing the FPGA and additional ICs, is part of a PCB realized by Allegrezza Giuseppe [15], in his thesis work. The prototype takes care of distributing supply voltages, enable protection for high power and hosts the six ADCs (AMC1306M25DWVR), the FPGA board (CMOD A7-35T) and the nine MOSFETS (IPAW60R600) with corresponding drivers (STGAP2HD).
In particular, on the board, the FPGA can communicate with the following modules:

- Flash Memory via Quad-SPI, for FPGA initial configuration (further details in next lines)

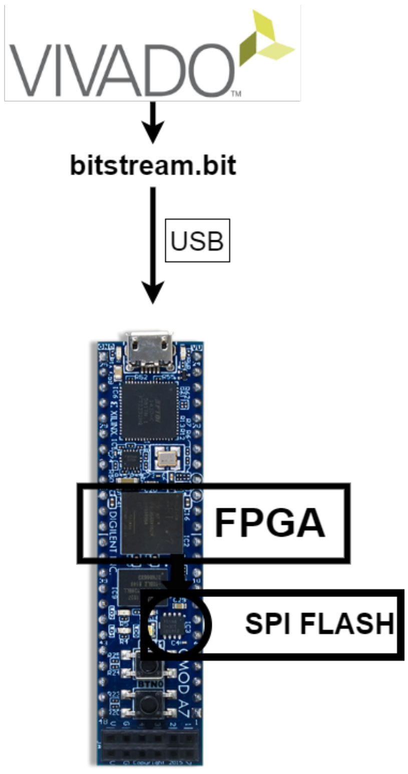


Figure 4.1: Indirect programming the FPGA.

- 512 KB SRAM for additional memory with a maximum bandwidth of 125 MBps
- Oscillator module to generate FPGA input clock at 12 MHz
- USB-UART bridge for serial communications; FW will communicate with the CLI tool on the user side via this interface
- GPIO comprising an RGB LED, 2 push buttons and 2 individual leds; mainly used for debug and generating processor system reset
- XADC, the ADC module present on 7 series FPGAs (not used)

The FPGA can be configured in two ways: by USB-JTAG circuitry, or by accessing a file stored in the flash memory. The second possibility has been implemented. In fact the bitstream will be generated in Vivado tool and stored inside the memory thanks to a method called *indirect programming* (fig. 4.1).

The SRAM can be accessed by an 'External Memory Controller' which has been integrated with its IP. Although has not been utilized in the current project, it can be taken in consideration for future upgrades.
The board generates a 12 MHz input clock for the FPGA, that feeds the internal

Clock Modifying Blocks (**CMB**), in particular the MMCM (a primitive clocking resource), which gets instantiated by the 'Clocking Wizard' and generates the derived clocks for all internal operations.

The GPIO buttons are used for resetting the system to a known state, included the MicroBlaze, whilst the LEDs are used for debugging purposes.
Obviously part of the pins on the FPGA are destined to the nine MOSFET drivers and to the six sensors. During this step the PCB designer supplies not only the schematics, but also the **XDC** file. This element is going to be used by the Synthesis and Implementation processes, and contains information about the physical constraints tied to the board IO placement (more in 4.4).

4.3 RTL Development

Once the IO is planned, is time for designing the RTL sources. Design choices made at this point influence the next steps. At this phase is necessary to:

- Plan the hierarchy of the design
- Identify the IP cores to use and customize in the design
- Create the custom RTL for interconnect logic, when a suitable IP is not available
- Create timing, power, and physical constraints
- Specify additional constraints, attributes, and other elements used during synthesis and implementation

The proposed hierarchy is illustrated as follows:

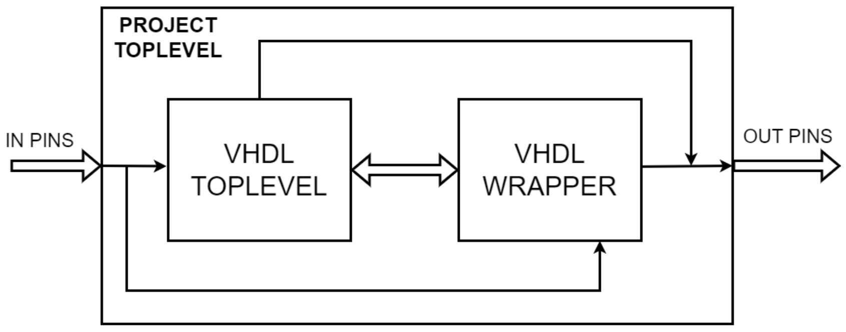


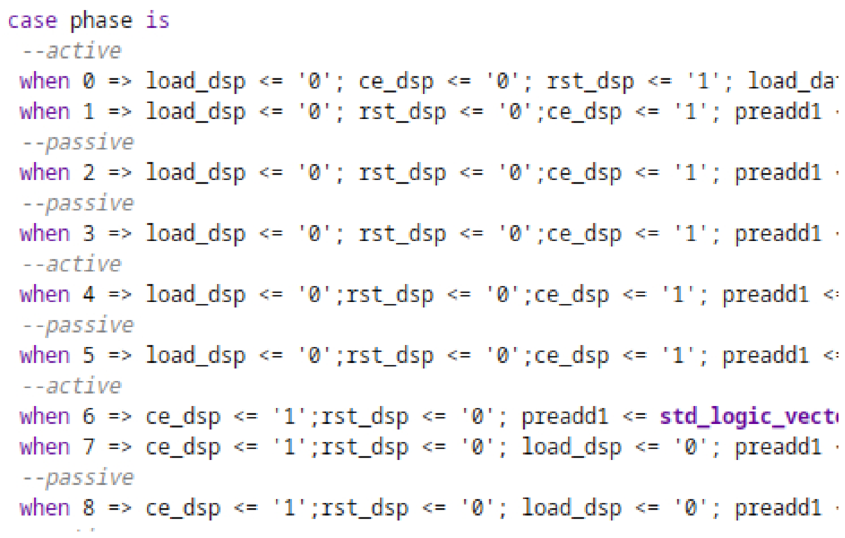
Figure 4.2: Design hierarchy.

Two major blocks can be distinguished: the "VHDL Toplevel" which allocates RTL sources designed with VHDL, and "VHDL Wrapper" which instantiates block IPs. Since the coded sources are written in VHDL-2008, we can't benefit of Vivado GUI for connections, so we have to handle them manually by writing additional code.

The ports which have to be routed to FPGA pins, are exposed in ’Project Toplevel’ and constrained by XDC file.
Some of the design rules adopted from the ’RTL Coding Guidelines’ are:

- Using Vivado HDL Templates; that’s the case for DSP48E1 instantiations with the ’ADDMACRO’ template
- Restructuring long datapaths with pipelining; the transition from ’to_float()’ VHDL method to LOD module is done with this perspective
- Synchronous vs. Asynchronous resets; majority of designed resets and enables are synchronous, for example the impulsive control signals in DSP datapath

Other suggestions include: knowing when inferring RAM or ROM, coding shift registers and delay lines, cosing styles to improve either frequency or power, and more.



```
case phase is
 --active
 when 0 => load_dsp <= '0'; ce_dsp <= '0'; rst_dsp <= '1'; load_da
 when 1 => load_dsp <= '0'; rst_dsp <= '0';ce_dsp <= '1'; preadd1
 --passive
 when 2 => load_dsp <= '0'; rst_dsp <= '0';ce_dsp <= '1'; preadd1
 --passive
 when 3 => load_dsp <= '0'; rst_dsp <= '0';ce_dsp <= '1'; preadd1
 --active
 when 4 => load_dsp <= '0';rst_dsp <= '0';ce_dsp <= '1'; preadd1 <
 --passive
 when 5 => load_dsp <= '0';rst_dsp <= '0';ce_dsp <= '1'; preadd1 <
 --active
 when 6 => ce_dsp <= '1';rst_dsp <= '0'; preadd1 <= std_logic_vect
 when 7 => ce_dsp <= '1';rst_dsp <= '0'; load_dsp <= '0'; preadd1
 --passive
 when 8 => ce_dsp <= '1';rst_dsp <= '0'; load_dsp <= '0'; preadd1
```

Figure 4.3: Succession of active and passive steps.

A useful tool in this phase, provided by the Vivado toolbox, is **LINTER**. It is a pre-synthesis program that can generate a general view of how the design modules are connected and can issue warnings if a particular line of code could be miss-interpreted by the synthesizer. A recurrent warning encountered during this phase, is the ’INFERRED LATCHES’ option. These elements are usually not recommended for FPGA designs because can easily become a point of failure, since they were inferred and not coded-defined. This warns about a bad code construct with conditional statements like if/else or case. To tackle the issue, the most delicate elemts have been redesigned, in particular the modules containing the DSP primitive, as the ’FPAU multiplie’, the ’ADDMACC EV’ or the ’ADDMACC DSP’. In fact for these blocks a big case statement is declared for controlling the state of the DSP at each clock phase. To fix the missing cases, additional phases have been added, in a way that doesn't hurt the functionality of the block (fig. 4.3). As a consequence, in the case statement, some conditions are considered as 'active', meaning that have a function to solve, and others as 'passive', referring to their neutral role.

4.4 Constraints

In pre-synthesis, the VHDL sources must be provided and so the constraints file. Designing good limitations is crucial for the success of the prototype, since they influence the synthesis tool (timing constraints) and the implementation tool (physical constraints). Over-constraining or under-constraining will make timing closure difficult.
For defining good limitations, we use ,as an entry-point, the XDC file provided by the board vendor (Digilent). XDCs derive from SDC (**S**ynopsys **D**esign **C**onstraints) and are an adaptation to AMD devices. It uses a file syntax based on TCL (**T**ool **C**ommand **L**anguage) and the SDC commands are similar to TCL ones. In a XDC file several type of constraints can be declared [16]:

- Wire load models; provide a statistical estimate of the wire-lengths
- System interface; provide guidance on the assumptions design needs to make about blocks it will be connected to or interacting with in a subsystem or chip or SoC
- Design rule constraints; requirements of the target technology
- Timing constraints; provide guidance on design parameters that affect operational frequency
- Timing exceptions; commands that help designer relax the requirements set forth by other commands thereby providing scope for leniency
- Area and power constraints; provide guidance on the area a design must fit within and power requirements for optimization

For this project, we focus on timing aspects. In particular the synthesis tool gets instructed on the main clock (12 MHz), while the generated ones (the 20 MHz, 120 MHz,...) are managed by 'Clocking Wizard' IP. In fact when adding an IP module in the design, not only the RTL sources are produced, but also specific constraints that the tools should be aware of.
After taking care of clocks, it's the turn for I/Os. The ADCs, the UART, the LEDs, the I2C, the SPI and the Gate Drivers are connected to specific FPGA pins. The ADCs input receive a special treatment in this sense, in fact we set an *input delay* condition and a *pulldown* propriety. The first one disables the ideal assumption that ports have zero delay, by specifying minimum and maximum delays that the combinational logic should have into account. The latter one is a protection

mechanism to avoid the chance of having floating ports, by inserting a pulldown resistor we assure the FPGA that when inputs are not driven, then they will get applied a weak logic low level.
Just to be complete, when synthesis fails the designer can opt either to restructure the RTL (changing code) or to loosen/tighten the constraints. This project is not an exception, and the final constraints are a result of this refining procedure.

4.5 Synthesis

The synthesis step has the role of converting source code (IPs and custom + constraints) into an optimized netlist. This is the element that interprets the code (VHDL-2008 in this case) and, depending how and if certain constructs are implemented, allocates FPGA primitives. Not all the VHDL-2008 procedures and methods are synthesizable, pointing out the difference between ’Designing’ and ’Designing for Synthesis’.
At process’ end, a final report and schematic is provided, with an addition of various post-synthesis tools, like DRC (Design Rule Checks) report, methodology report and a QOR (Quality of Results) report.

4.6 Implementation

The implementation tools runs the ’Routing’ and ’Floor-planning’ algorithms. ’Routing’, or better ’Routing and Placement’ establishes the optimal topology on the FPGA that can achieve the specified constraints. Several optimization criterion can be required, for example ’Power optimization’ that optimizes dynamic power using clock gating, or ’Logic optimization’ that ensures the most efficient logic design before attempting placement, or even creating a new and custom implementation strategy. For the project, no optimization technique is preferred to another, so it is kept at its default value.
A successful implementation will generate the bitstream that will program the FPGA with the procedure described in section 4.2.

Chapter 5

Results and Analysis

In this chapter the COSIM and Vivado results are illustrated. First the design is simulated in COSIM environment to validate its functionality, then the synthesis and implementation solution is showed. Finally a latency breakdown concludes the analysis.

5.1 COSIM Results

In COSIM the design can be tested to validate the functioning of the 'Frequency Synthesizer', the correctness of ADC inputs, the validity of configuration switching and more. Let's start from the simulated sources. The ADCs on the board are $\Sigma\Delta$ modulators working at 20 Mbps, since they cannot be integrated in COSIM, a C++ code simulates their behaviour. From this file it's possible to fake the real input bitstream for a sinusoid with a configurable amplitude and frequency. The V_{des} is custom too and is adjustable in FW or via CLI instruction.

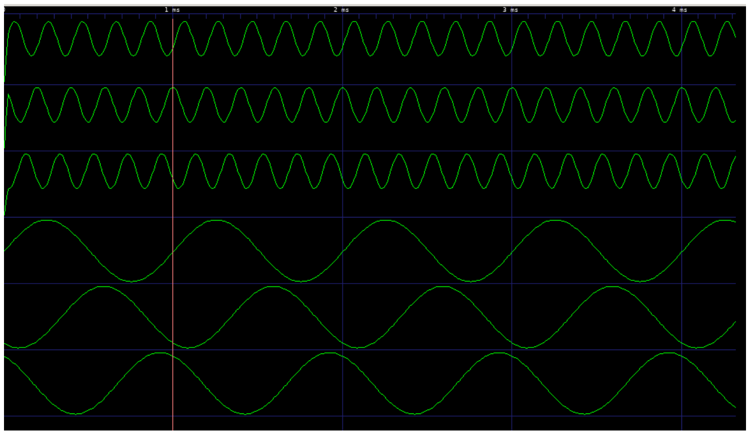


Figure 5.1: Simulated ADCs and Frequency Synthesizer.

The top 3 sinewaves are the output from the ADCs, and are simulated with a frequency of 5 kHz, while the bottom ones are synthesized by a frequency of 1 kHz (note the timeline at the top).
A clear example of HW-FW-SW interaction can be seen with the next figure:

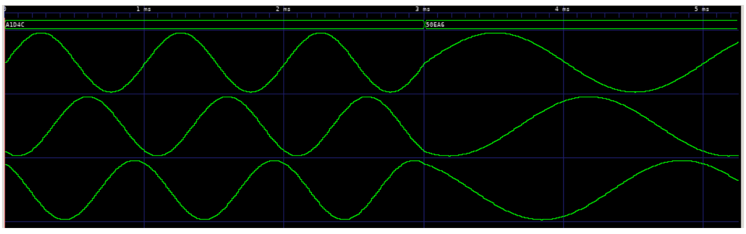


Figure 5.2: On-line frequency change.

where is showed, in response to the CLI command, the switch in frequency (by a 0.5 factor).
An UART transaction is reported in next figures. The transmission SW->FW/HW in fig. 5.3, and the reception SW<-FW/HW in fig. 5.4. Notice the wrapping protocol as stated in section 3.3.

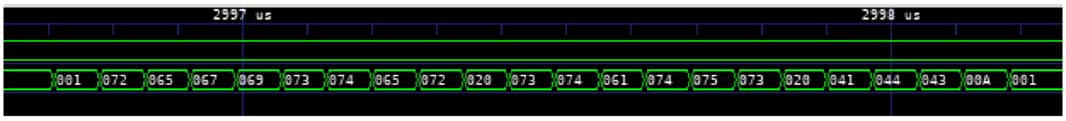

2997 us
2998 us
001 072 065 067 069 073 074 065 072 020 073 074 061 074 075 073 020 041 044 043 00A 001

Figure 5.3: Serial communication for CLI instruction. (TX)

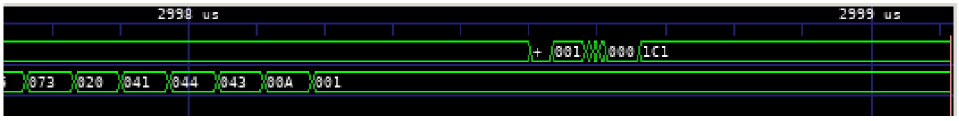


Figure 5.4: Serial communication for CLI instruction. (RX)

Finally the quantization action. In fig. 5.5, the top signal shows the chosen $\tilde{k}$ out of the 27 configurations at each time sample, the *min_p_x* signals indicate which phase should be linked to output voltages ('11'=v_c, '10'=v_b, '01'=v_a, '00' not allowed), and the *vx_actual* are the output at DMC terminals. Since the load current waveforms are not available, the filtering won't reproduce the desired frequency, but we can appreciate the multilevel quantization doing its job.

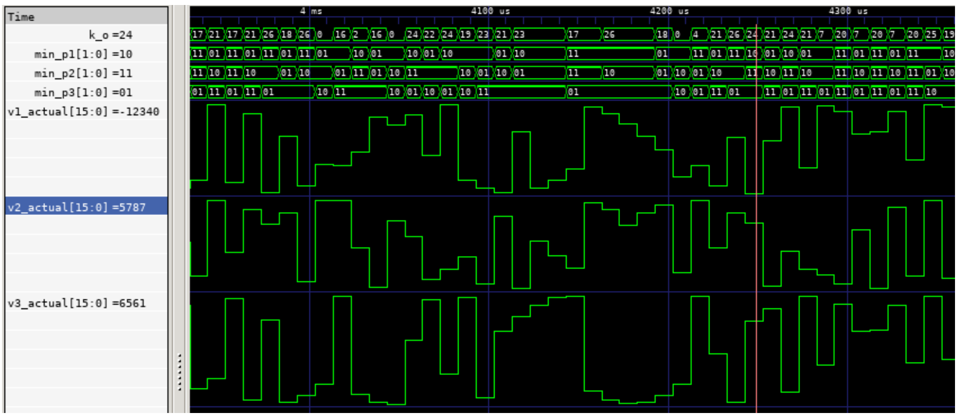
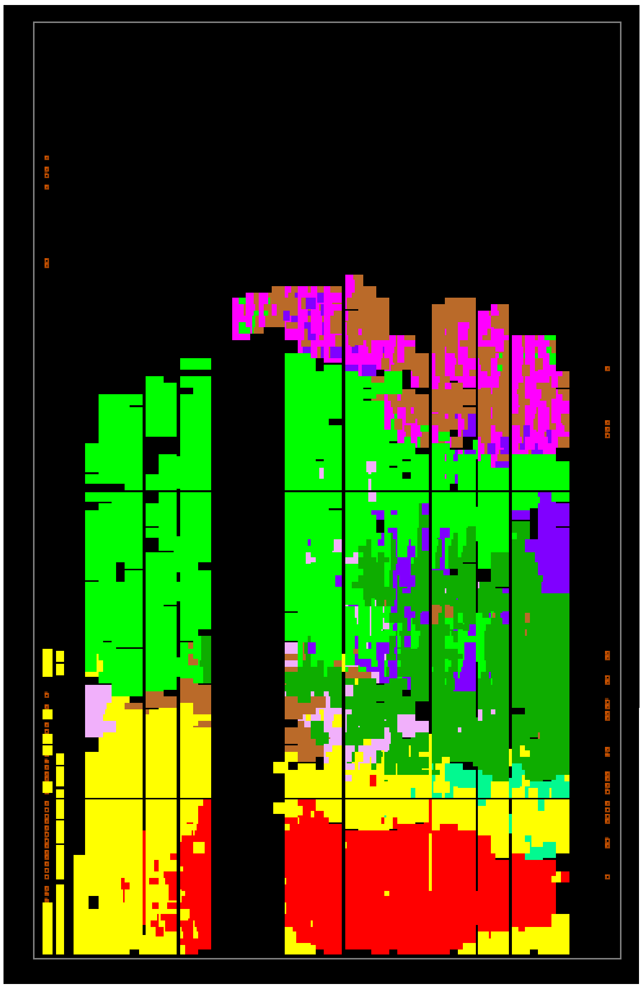


Figure 5.5: DMC output voltages.

5.2 Layout Results

The following figure illustrates the programmed FPGA:



(a)

⬛	Blackground
🟥	MicroBlaze
🟨	VHDL Wrapper (IP design)
🟪	Demultiplexers
🟥	Multiplexers
🟩	Datapath
🟫	Minimum Detector
🟦	UART
🟪	CPUiface
🟩	remaining modules (adc bank, mod bank,...)

(b)

Figure 5.6: (a) Amoeba view and (b) its color legend.

The figure 5.7 shows the cells utilization on the device. IO Banks and Clocking regions are also included.

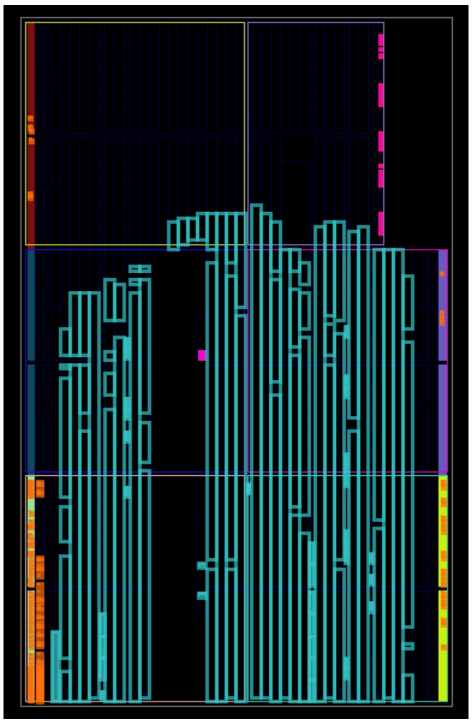


Figure 5.7: Cell view.

A final summary after Placement and Routing of resource usage:

	Slice Regs		LUTs		DSP		BRAM		I/O	
TOTAL	12519	30.09%	13331	64.09%	17	18.89%	12	24%	73	68.87%
ADC BANK	1484	3.57%	1386	6.66%	0	0%	0	0%	0	0%
DATAPATH	1258	3.02%	1215	5.84%	4	4.44%	0	0%	0	0%
MIN DETECTOR	944	2.27%	846	4.07%	0	0%	0	0%	0	0%
IPs	3826	9.20%	4588	22.06%	2	2.22%	11	22%	0	0%
MICROBLAZE	1737	4.18%	2177	10.47%	2	2.22%	6	12%	0	0%
CPUIFACE	252	0.61%	161	0.77%	0	0%	0	0%	0	0%

Chapter 6

Conclusions

The work touched many critical aspects of a digital design: FPGA programming, mixed-signal sides, FW coding, pipelined architectures, inter-chip communications, waveform analysis, resource-optimized techniques and cosimulation HW-FW-SW. Learning the tools like Linux command line, VHDL's rigorous syntax, the lightweight COSIM, and the not so lightweight Xilinx's suite, has been an exciting path and a first approach to the fascinating world of VLSI chips and its industry standards.

6.1 Future Work

For possible upgrades and completion of the project, here it is a list for next steps:

- Test board with programmed FPGA, in an hypothetical scenario
- Upgrade FPAUs with overflow/underflow flags
- Find optimal coefficient values for CIFF modulators
- Refine CIFF modulator's saturation point for input reactive power

Bibliography

[1] Marco Matteini. Control techniques for matrix converter adjustable speed drives. *Diss. Ph. D. Dissertation, Dept. Electrical. Eng., Univ. Bologna, Bologna, Italy*, 2001.

[2] ANALOG DEVICES. Fundamentals of direct digital synthesis (dds), 2008.

[3] Shyama Gandhi, Mohammad Saeed Ansari, Bruce F Cockburn, and Jie Han. Approximate leading one detector design for a hardware-efficient mitchell multiplier. In *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, pages 1–4. IEEE, 2019.

[4] Muhammad H Rashid. *Power electronics handbook*. Butterworth-heinemann, 2017.

[5] Jiayang Wu, Albert T. L. Lee, Siew-Chong Tan, and S. Y. Ron Hui. A bridgeless single-stage single-inductor multiple-output (simo) ac-ac converter for wireless power transfer applications. In *2023 IEEE Wireless Power Technology Conference and Expo (WPTCE)*, pages 1–5, 2023.

[6] Marco Venturini and Alberto Alesina. The generalised transformer: A new bidirectional, sinusoidal waveform frequency converter with continuously adjustable input power factor. In *1980 IEEE Power Electronics Specialists Conference*, pages 242–252. IEEE, 1980.

[7] Alberto Alesina and Marco GB Venturini. Analysis and design of optimum-amplitude nine-switch direct ac-ac converters. *IEEE transactions on power electronics*, 4(1):101–112, 1989.

[8] Laszlo Huber and Dusan Borojevic. Space vector modulated three-phase to three-phase matrix converter with input power factor correction. *IEEE transactions on industry applications*, 31(6):1234–1246, 1995.

[9] Laszlo Huber, Nándor Burány, and Dušan Borojević. Analysis, design and implementation of the space-vector modulator for forced-commutated cycloconvertors. In *IEE Proceedings B (Electric Power Applications)*, pages 103–113. IET, 1992.

[10] Simone Orcioni, Giorgio Biagetti, Paolo Crippa, and Laura Falaschetti. A driving technique for ac-ac direct matrix converters based on sigma-delta modulation. *Energies*, 12(6):1103, 2019.

[11] Shanthi Pavan, Richard Schreier, and Gabor C Temes. *Understanding delta-sigma data converters.* John Wiley & Sons, 2017.

[12] Anindya Dasgupta and Parthasarathi Sensarma. *Design and Control of Matrix Converters.* Springer, 2017.

[13] Giorgio Biagetti, Laura Falaschetti, Paolo Crippa, Michele Alessandrini, and Claudio Turchetti. Open-source hw/sw co-simulation using qemu and ghdl for vhdl-based soc design. *Electronics*, 12(18):3986, 2023.

[14] Florent De Dinechin and Bogdan Pasca. Large multipliers with less dsp blocks. In *Field Programmable Logic and Applications.* IEEE, 2009.

[15] GIUSEPPE ALLEGREZZA. Sviluppo di un prototipo basato su fpga per il pilotaggio in modulazione sigma-delta di convertitori di potenza a matrice. 2021.

[16] Sridhar Gangadharan and Sanjay Churiwala. *Constraining Designs for Synthesis and Timing Analysis.* Springer, 2013.