

SmartRCP

User Manual



2024.1

pSC
Power Smart Control SL

Contents

1	Document History Revision	7
2	List of acronyms	8
3	Introduction	10
4	Capabilities	11
4.1	Processing power	11
4.2	Extreme tailor-made connectivity	11
4.3	Fully-programmable not black-box system	12
4.4	Networking and paralleling technology	12
4.5	Highlighted use-cases	13
5	Hardware & CarrierBoard details	14
5.1	Mechanical design, dimensions, and weight	14
5.2	Operation requirements	14
5.3	Specification of connectivity and resources	14
5.4	Serial number and HW revision	15
5.5	KRIA K26 description and main features	16
5.6	Connector description: pin-out, function, voltages range and bandwidth . .	17
5.7	Constraints file	25
5.8	Ventilation	26
5.9	Booting modes	26
5.10	USB programming and debugging	26
5.11	In Depth description of functional blocks	26
5.11.1	RS-232	26
5.11.2	RS-485	27
5.11.3	User LEDs	28
5.11.4	Ethernet ring communication	29
5.11.5	Analog output channels	30
5.11.6	Digital inputs	31
5.11.7	Digital outputs	31
5.11.8	PWMs	33
5.11.9	Analog input channels	37
5.12	Connector shielding	39
6	Programming user guide	40
6.1	Development enviroment	40
6.1.1	Introduction to Vitis	40
6.1.2	Vitis Download and Installation Instructions	40
6.2	Vivado software tools	41
6.2.1	How to create a project and the main configuration	41
6.2.2	Add an IP	44
6.2.3	Use the Constraints and create Ports	44
6.2.4	Generate Bitstream	45
6.2.5	How to use SmartRCP_Template provided by PSC	47
6.3	Documentation of Driver IPs included in SmartRCP_Template	50

6.3.1	PSC_CarrierBoard_driv IP	50
6.3.2	IP AXI_PSCoscilloscope	66
6.3.3	Integrated Logic Analyzer (ILA)	70
6.4	Xilinx Vitis software tools	71
6.4.1	Introduction	71
6.4.2	Project creation in Vitis	71
6.4.3	Procedure for loading and debugging the design on SmartRCP . . .	73
6.4.4	How to make multi-processor designs	74
7	SmartRCP Test Mode	77
8	Synchronous Buck Converter demonstrator	78
8.1	Hardware	78
8.2	SmartRCP configuring	82
8.2.1	Introduction	82
8.2.2	Creating C-code based IP cores using Vitis HLS	83
8.2.3	Configuring SmartRCP_Template in Vitis Vivado	89
8.2.4	Configuring processors with Vitis IDE	105
8.2.5	Using the graphical interface	115
9	Terms of Use, Limitations, and Warranty for SmartRCP	117
9.1	Scope of SmartRCP (use, applications, and regulations)	117
9.2	Acceptance of goods (by the buyer)	118
9.3	Limited warranty for SmartRCP	118
9.4	Third parties	119
10	Contact and Support	120
11	Annex A SensingBoard design	122

List of Figures

1	KRIA SOM module [1].	11
2	Networking main topologies.	12
3	SmartRCP main dimensions.	14
4	Serial QR codes and HW revision information.	15
5	KRIA K26 SOM comparison among other Xilinx devices [1].	16
6	KRIA K26 main components and architecture [3].	17
7	RS232 functional block chain (J15).	27
8	RS232 functional block chain (J16).	27
9	RS485 functional block chain (J10).	28
10	RS485 functional block chain (J11).	28
11	SmartRCP user LED position.	29
12	User LEDs functional block chain.	29
13	Ethernet ring functional block chain.	30
14	Analog outputs functional block chain.	30
15	0-24V Digital inputs functional block chain.	31
16	0-24V Digital outputs functional block chain.	32
17	Additional 0-3.3V functional block chain.	32
18	PWM functional block chain (J29 and J31).	33
19	PWM functional block chain (J33 and J35).	33
20	PWM functional block chain (J37 and J39).	34
21	PWM functional block chain (J41).	34
22	PWM functional block chain (J30 and J32).	34
23	PWM functional block chain (J34 and J36).	35
24	PWM functional block chain (J38 and J40).	35
25	PWM functional block chain (J42 and J44).	35
26	PWM functional block chain (J43).	36
27	PWM functional block chain. Enable signal detail.	36
28	PWM channels RJ45 connector pin out.	37
29	Analog input channels front end	37
30	Slow analog input functional block chain.	38
31	Quick analog input functional block chain.	39
32	Vitis installing options.	41
33	Vivado project options.	42
34	Vivado project part definition.	42
35	Vivado project option configurationn	43
36	IP repository in Vivado project.	44
37	Creating an external port.	45
38	Changing name to an external port.	45
39	Creating HDL wrapper.	46
40	Generating output products.	46
41	Generating bitstream.	46
42	Exporting HW.	47
43	Exporting HW with bitstream.	47
44	Opening SmartRCP_Template.xpr.	48
45	IP Integrator view of SmartRCP_Template Vivado project.	49

46	IP Integrator view of SmartRCP_Template Vivado project without TEST blocks.	50
47	Ethernet Ring RJ45 signal wiring.	54
48	Analog outputs connector terminology.	55
49	Digital inputs connector terminology.	57
50	Digital outputs connector terminology (0-24V).	58
51	Digital outputs connector terminology (0-3.3V).	59
52	Pin out of the PWM RJ45 connector.	60
53	AXI_PSCoscilloscope IP and its functional parts.	67
54	Contents of PSCoscilloscope_conf.txt file.	69
55	PSCoscilloscope application.	69
56	Vitis architecture definition.	71
57	Vitis application names and resources.	72
58	Vitis OS or standalone mode selection.	72
59	Creating boot image in Vitis.	73
60	Adding additional application project in Vitis.	74
61	New processor added to the design in Vitis.	74
62	Selecting new processor in Vitis.	75
63	Linker memory positions in Vitis.	75
64	DDR memory available in KRIA K26.	76
65	RCP_Test_Tool Windows application main window.	77
66	Buck Converter topology included in RCPBuckBoard	78
67	RCPBuckBoard.	79
68	Connection among RCPBuckBoard and SmartRCP.	79
69	RCPBuckBoard schematic.	80
70	RCPBuckBoard connection to a desktop oscilloscope.	81
71	RCPBuckBoard connection to a desktop oscilloscope.	81
72	RCPBuckBoard project. Main block diagram and connection.	82
73	Main block diagram of the voltage control loop.	83
74	Moving average filter C code.	84
75	Vitis HLS project settings I.	84
76	Vitis HLS project settings II.	85
77	Vitis HLS project settings. Add sources.	85
78	Vitis HLS project settings. Select top function.	85
79	Vitis HLS project settings. Adding directives.	86
80	Synthesizing the IP in Vitis HLS.	86
81	Implementing the IP in Vitis HLS.	87
82	Result of the IP generated in Vitis HLS.	88
83	IP extra signals created by Vitis HLS.	88
84	PID compensator IP. View of inner algorithm and IP general view.	89
85	PID compensator IP. View of inner algorithm timing and IP general view.	89
86	Vivado IP integrator IP blocks after deleting testing blocks.	90
87	Adding Vitis HLS IP blocks created to Vivado IP repository.	91
88	Adding Vitis HLS IP blocks to Vivado IP integrator.	91
89	Designer Assistance configuration.	92
90	Required connection of signals between IP blocks.	92
91	PSC_CarrierBoard_driv IP connection.	93
92	AXI peripheral creation I.	96

93	AXI peripheral creation II.	97
94	AXI peripheral creation III.	97
95	AXI peripheral creation IV.	97
96	AXI peripheral creation V.	98
97	AXI peripheral creation VI.	98
98	AXI peripheral creation VII.	98
99	AXI peripheral creation VIII.	99
100	AXI peripheral creation IX.	99
101	AXI peripheral creation X.	100
102	AXI peripheral creation XI.	100
103	AXI peripheral creation XII.	101
104	AXI peripheral creation XIII.	101
105	AXI IP real path	101
106	Makefile path	102
107	AXI peripheral creation XIV. Including IP in Vivado.	102
108	Configuration of the AXI IP in Vivado.	103
109	Validate the Vivado design.	103
110	Create the wrapper I.	103
111	Create the wrapper II.	104
112	Export Vivado project I.	104
113	Export Vivado project II.	105
114	Microprocessor resources available in KRIA K26	105
115	Configuring Vitis project I.	106
116	Configuring Vitis project II.	106
117	Configuring Vitis project III.	106
118	Configuring Vitis project IV.	107
119	Configuring Vitis project V.	107
120	Vitis IDE main sections.	108
121	Building in VITIS IDE.	108
122	Checking IP base address in Vivado Adress Editor.	108
123	Update the code in Vitis IDE.	109
124	Open console at Vitis IDE.	110
125	Launching debugger in Vitis IDE I.	110
126	Accessing ILA when debugging.	111
127	Change linker file in ARM A53 when PSCoscilloscope is included.	111
128	Include PSCoscilloscope .elf file when programming or debugging.	112
129	Files required for creating booting files.	112
130	Vitis configuration for creating boot files I.	113
131	Vitis configuration for creating boot files II.	113
132	Vitis configuration for creating boot files III.	113
133	Vitis configuration for creating boot files IV.	114
134	Vitis configuration for creating boot files V.	114
135	RCP_conf.txt file content.	114
136	RCPBuckBoard graphical user interface.	115
137	Compensator parameters.	116
138	NTC temperature sensing analog chain	123
139	NTC temperature sensing schematic.	124
140	LEM current sensing analog chain	125

141 LEM current sensing schematic. 126

142 DC voltage sensing analog chain 126

143 DC voltage sensing schematic. 128

144 AC voltage sensing analog chain 128

145 AC voltage sensing schematic. 130

1 Document History Revision

Date	Person	Revision
2023-08-05	JJG, RVG, JRF	1st version
2024-02-12	RVG, JRF	Revision
2024-04-10	JLL, JRF	Revision
2024-06-03	RVG, JRF	AXI IP makefile – page 90
2024-06-27	JRF	Synchronous Buck Converter demonstrator

2 List of acronyms

ADC – Analog to Digital converter

APP – Application

AXI – Advanced eXtensible Interface

DAC – Digital to Analog Converter

DDR – Double Data Rate

DT – Digital Twin

ENA – Enable

ERR – Error

ETH – Ethernet

FPGA – Field Programmable Gate Arrays

FW – Firmware

HIL – Hardware in The Loop

HW – Hardware

I/O – Input and Output

ILA – Integrated Logic Analyzer

JTAG – Joint Test Action Group

LED – Light Emitting Diode

PCB – Printed Circuit Board

PL – Programmable Logic

PS – Processing System

PSC – Power Smart Control SL

PWM – Pulse Width Modulation

RCP – Rapid Control Prototyping

RTS – Real Time Simulator

SDK – Software Development Kit

SOM – System on Module

SW – Software

UART – Universal Asynchronous Receiver Transmitter

3 Introduction

Rapid Control Prototyping Systems (RCP) is one of the key aspects to accelerate time to market of power electronics converters, especially in the range of several kW and up to MW.

The RCP system developed by Power Smart Control (PSC), SmartRCP, provides the perfect balance between digital processing power and hardware interconnectivity to enable connecting the market most powerful RCP system to the user converter with none or minimal intervention.

SmartRCP system is based in a Xilinx KRIA K26 System on Chip (SoC) module mounted on a high-tech tailor-made carrier board (CarrierBoard) which provides almost infinite power to perform:

- General and specific control algorithmic validation
- Power converter validation
- Control algorithmic pre-certification
- Finite State Machine (FSM) testing
- Digital Twin (DT) and Real Time Simulator (RTS) systems

In PSC we do not believe in black-box systems approximations, our RCP system is fully accessible by the user, users can program the FPGA by using HLS or directly in VHDL or Verilog and, at the same time, program the four available high-specs ARM microprocessors. This system is fully programmable.

Thanks to the massive flexible high-speed sensing stage available in SmartRCP, users can program his own protection logic to avoid over-voltages, over-currents and over-temperature situations in his converter at the same time. All sensors can be connected by means of none or minor adaptations as the analog front-end gain, offset and band-width is fully parametrizable.

Users can also embed a Digital Twin or Real Time Simulator, create a supervisor system or even make automatic testing of different power converter working modes by connecting remotely to SmartRCP system.

As SmartRCP provides a 1GB Ethernet connector and an enormous powerful core it is also suited for IA applications related with converter controlling such as end-of-life estimation, predictive maintenance, etc.

By the use of SmartRCP_Template and tutorials the user has a quite valuable starting point without any limitation neither hardware nor software to perform the controlling and testing he deserves.

4 Capabilities

SmartRCP has been conceived to contain a processing core seamlessly bond to a tailor-made CarrierBoard to provide the user the most powerful solution in the market.

4.1 Processing power

The K26 SOM leverages the XCK26-SFVC784-2LV-C/I, a custom-built Zynq Ultra-Scale+ MPSoC that runs optimally (and exclusively) on the SOM. It provides an embedded processing system (PS) with tightly integrated programmable logic and a rich set of configurable I/O capabilities.

Those capabilities make it possible to obtain PWM channels with less than 4ns time resolution, provides more than 200 user I/Os and integrates 6 high-end ARM microcontrollers.

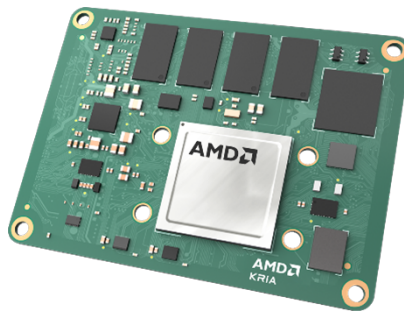


Figure 1: KRIA SOM module [1].

4.2 Extreme tailor-made connectivity

The power provided by KRIA K26 is enhanced employing a tailor-made CarrierBoard which provides all HW building blocks required for an RCP system. It contains:

- 32x 0–24V digital inputs channels
- 32x 0–24V digital outputs channels
- 14x digital output 0–3.3V channels
- 36x PWM channels
- 24x analog input channels
- 16x analog output channels
- 1x GB ETH, 4x USB2.0
- 2x UART 2x RS-485 & 1x microSD

4.3 Fully-programmable not black-box system

SmartRCP systems are programmed and configured by using Xilinx tools and also by several templates and examples provided by PSC. The programming environment is Xilinx Vitis Vivado – including Vitis HLS. PSC provides in addition:

- All drivers and pin assignment already done.
- A test program to check carrier board specs and to use as a base point by the user for new designs.
- All Vivado and Vitis configurations already done.
- The possibility for the user to change those templates or include new ones to fits his needs.
- Several examples already done.
- An embedded oscilloscope and data-tracker.

There is no black-box approach in this RCP, everything is transparent and can be done by the user, both in the FPGA and in the microprocessors.

4.4 Networking and paralleling technology

SmartRCP systems can be set to work in parallel by communicating with a master unit forming a star, with each other forming a ring, or by other configurations. Full freedom is given to the user to create his own protocol and connection scheme. The minimum sampling/clocking time for that protocol is 4ns.

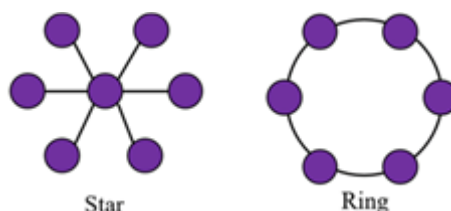


Figure 2: Networking main topologies.

This makes the SmartRCP system fully expandable both in digital resources and also in connecting resources at the time it assures, thanks to high-speed 4ns resolution channels, perfect synchronization of all the devices connected.

SmartRCP system also offers a 1GB ethernet RJ45 connector for networking, this capability enables remote FW upload, remote monitoring and operation, etc., and is fully independent of the networking configuration selected both in HW and SW.

4.5 Highlighted use-cases

The capabilities of SmartRCP make it suitable for an infinite number of applications, however, there are some which must be highlighted:

- Power converter controlling. It can be used by FW department for debugging or while the user Control Unit is being developed.
- It is ideal for testing complicated working scenarios where finite state machine logic is complex or has a high number of variables implicated.
- It can be used to pre-certify control algorithms or even the power converter response to several scenarios like grid deeps, etc.
- Its huge computing power makes it suitable to include supervising systems (Maybe based on IA algorithms) or even Digital Twins or Real Time Simulators.

We have developed SmartRCP to make the possibilities infinite.

5 Hardware & CarrierBoard details

5.1 Mechanical design, dimensions, and weight

SmartRCP comprises a CarrierBoard enclosed in a polymer case that offers access to connections and warrants its secure operation and handling.

Total weight is approximately 0.4kg.

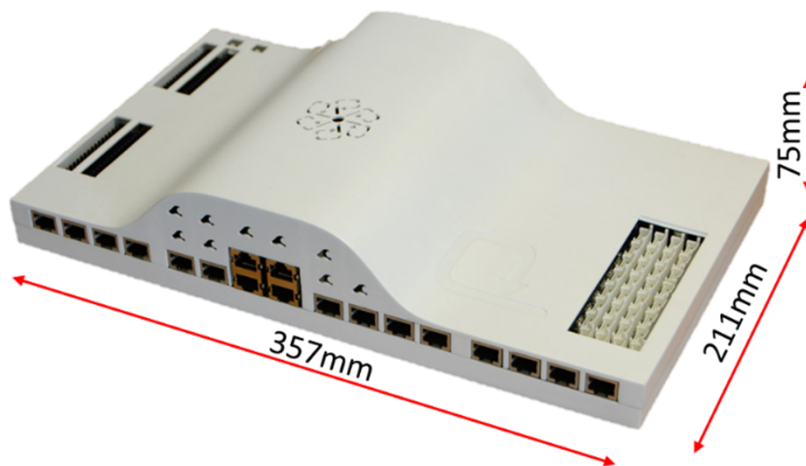


Figure 3: SmartRCP main dimensions.

5.2 Operation requirements

SmartRCP is designed to work from an office to a field environment. To ensure correct care and proper operation the following conditions must be met:

- Electrical connection: DC source from 10 to 35Vdc – nominally it should be connected to 24Vdc. A proper universal AC adapter is included with each unit.
- Power: Maximum power consumption is 60W.
- Storage temperature: From -10 to 60 Celsius degrees.
- Working temperature: From 15 to 40 Celsius degrees.
- Humidity: From 5 to 60
- Salt ambient: Not supported for long periods.

5.3 Specification of connectivity and resources

SmartRCP includes the following resources:

- 32x 0-24V digital inputs – LED indicated

- 32x digital outputs 0-24V – LED indicated
- 14x digital outputs 0-3.3V
- 36x PWM signals – 4ns time resolution, RS-422, RJ45 connector
- 24x analog input channels (12x 500ksamples/s and 12x 3Msamples/s)
- 16x analog output channels (500ksamples/s)
- 1x GB ETH (RJ-45)
- 4x USB2.0
- 2x RS-485 (full duplex)
- 1x microUSB for programming, debugging and serial communication
- 1x microSD card
- 1x Reset button
- 1x Programming button

5.4 Serial number and HW revision

SmartRCP systems contain a QR code that includes:

- Board number and manufacturing data
- Hardware revision

This QR code is placed:

- On the bottom of the polymer cover
- On top of CarrierBoard

To ease the identification of the HW revision, it is also indicated in top of CarrierBoard with letters and numbers.

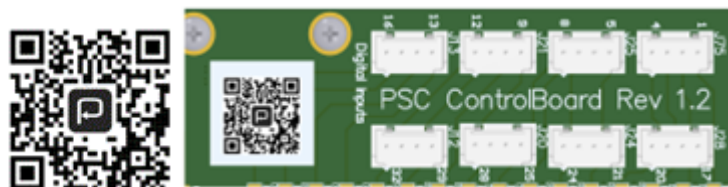


Figure 4: Serial QR codes and HW revision information.

The serial number can be checked when microUSB programming and debugging wire is connected to a computer and Vitis is used.

5.5 KRIA K26 description and main features

Xilinx KRIA K26 SOM leverages the XCK26-SFVC784-2LV-C/I, a custom-built Zynq UltraScale+ MPSoC that runs optimally (and exclusively) on the SOM. It provides an embedded processing system (PS) with tightly integrated programmable logic (PL) and a rich set of configurable I/O capabilities.

The picture below provides a view of the KRIA K26 SOM.

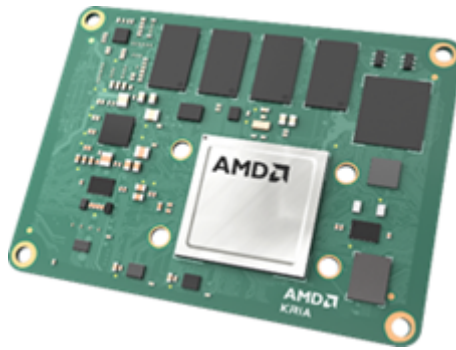


Figure 5: KRIA K26 SOM comparison among other Xilinx devices [1].

KRIA K26 SoC includes:

- Huge highly enriched FPGA

Resource	K26 SOM
System logic cells	256,200
CLB flip-flops	234,240
CLB LUTs	117,120
Distributed RAM (Mb)	3.5
Block RAM	144
Block RAM (Mb)	5.1
UltraRAM blocks	64
DSP slices	1,248
GTH transceivers	4
Video Codec	1
HDIO	69
HPIO	116

[2]

- 2x ARM Cortex R5 real time processors
- 4x ARM Cortex A53 application processors
- 1x ARM Mali dedicated GPU
- 4GB 64bits DDR4 RAM
- Non volatile memories (512 Mb QSPI, 16 GB eMMC & 64 Kb EEPROM)

- Embedded peripheral (I2C, SPI, UART, SD, USB, ETH, DDR controller, RTC, etc.)

The architecture of KRIA K26 makes a seamless union of Programmable Logic (FPGA) and PS (processing system) so users can create their own peripherals in the FPGA and command them with one or several microprocessors. All resources in the SoC are highly interconnected by an ARM proprietary bus (AXI bus). The figure below shows KRIA 26 architecture.

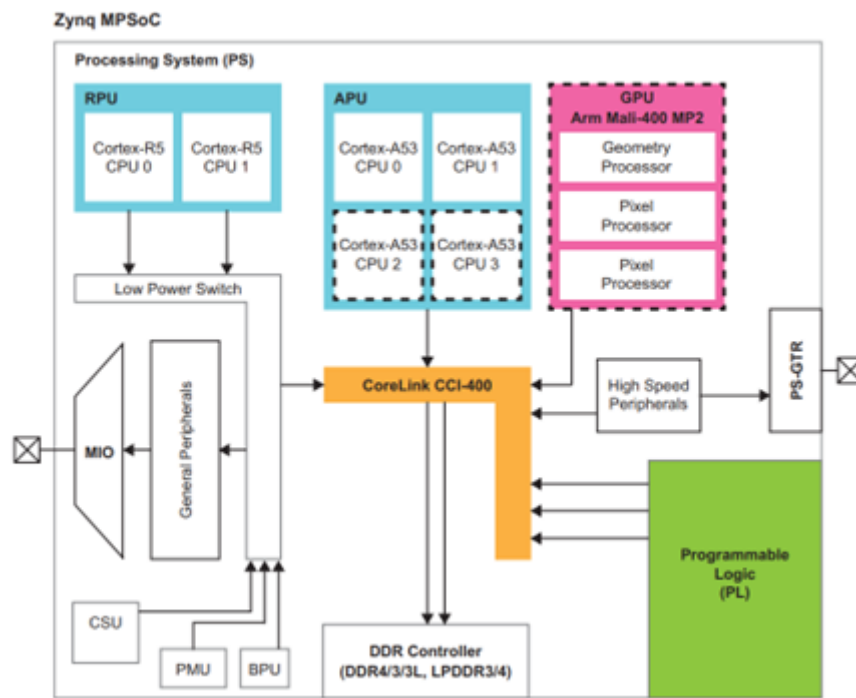


Figure 6: KRIA K26 main components and architecture [3].

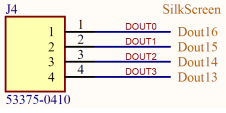
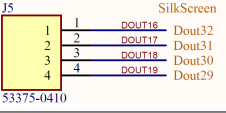
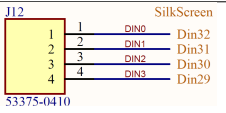
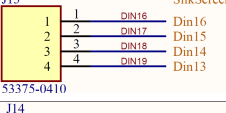
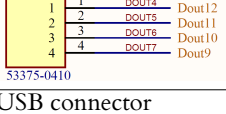
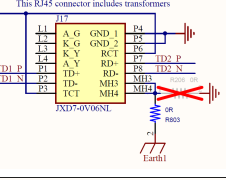
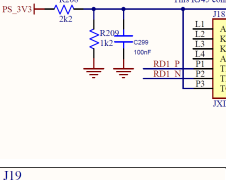
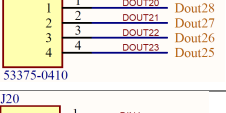
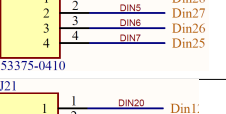
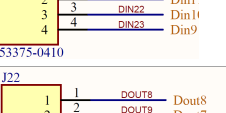
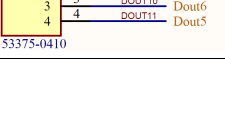
KRIA K26 is the fundamental piece on top of which SmartRCP is built. It provides a digital power not seen before. This power, in combination with the tailor-made CarrierBoard, offers the market the ultimate and most powerful solution for Rapid Control Prototyping.

5.6 Connector description: pin-out, function, voltages range and bandwidth

The following table describes all connectors available in the CarrierBoard, its pin-out, and its bandwidth. This table also includes the voltage range of each pin and connector. Absolute care should be taken to avoid connecting those pins to a higher voltage standard, doing so will inevitably result in hardware damage.

Care should be taken when using power pins not to override the specs of the hardware chain involved. Taking a high current may damage the power supply sub-system.

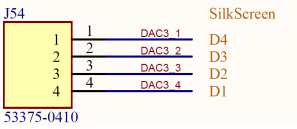
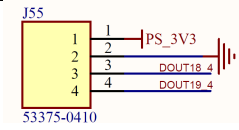
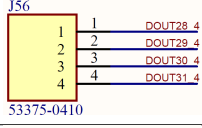
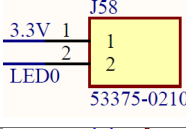
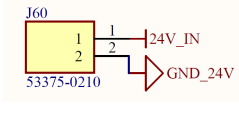
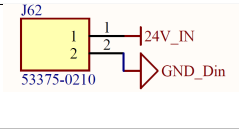
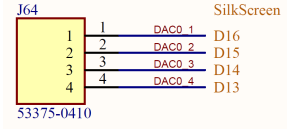
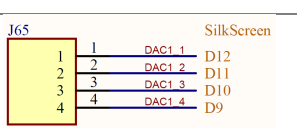
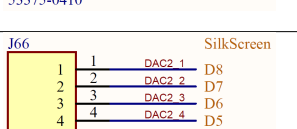
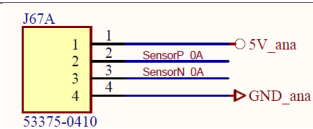
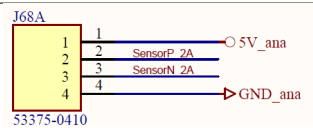
It follows a figure showing the physical placement of each connector inside the card.

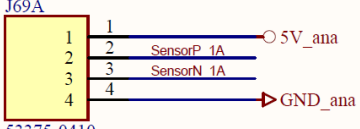
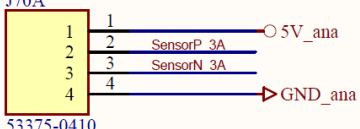
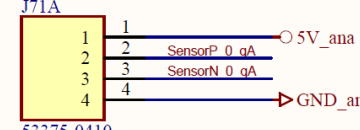
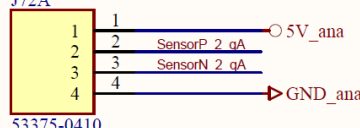
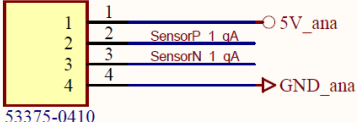
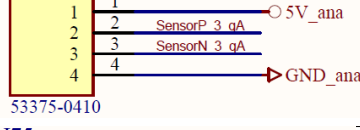
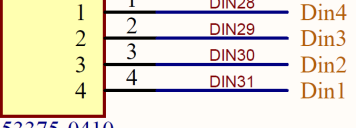
ID	Function	Pin-out	V	BW	Comment
J4	Digital outputs		0-24V	1.5KHz	Multiplexed fast i2c
J5	Digital outputs		0-24V	1.5KHz	Multiplexed fast i2c
J6	Programming & debugging	Female Micro USB connector	0-5V		Connect to a PC
J8	ETH	ETH RJ45 connector	0-3.3V	1GB	Universal use
J9	Micro SD card	Micro SD card connector	0-3.3V		FW storage and disk
J10	RS-485	Female DB9 connector	0-3.3V		
J11	RS-485	Female DB9 connector	0-3.3V		
J12	Digital inputs		0-24V	1.5KHz	Multiplexed fast i2c
J13	Digital inputs		0-24V	1.5KHz	Multiplexed fast i2c
J14	Digital outputs		0-24V	1.5KHz	Multiplexed fast i2c
J15	UART - USB	USB connector	0-5V		Converted to USB
J16	UART - USB	USB connector	0-5V		Converted to USB
J17	ETH Ring				
J18	ETH Ring				
J19	Digital outputs		0-24V	1.5KHz	Multiplexed fast i2c
J20	Digital inputs		0-24V	1.5KHz	Multiplexed fast i2c
J21	Digital inputs		0-24V	1.5KHz	Multiplexed fast i2c
J22	Digital outputs		0-24V	1.5KHz	Multiplexed fast i2c

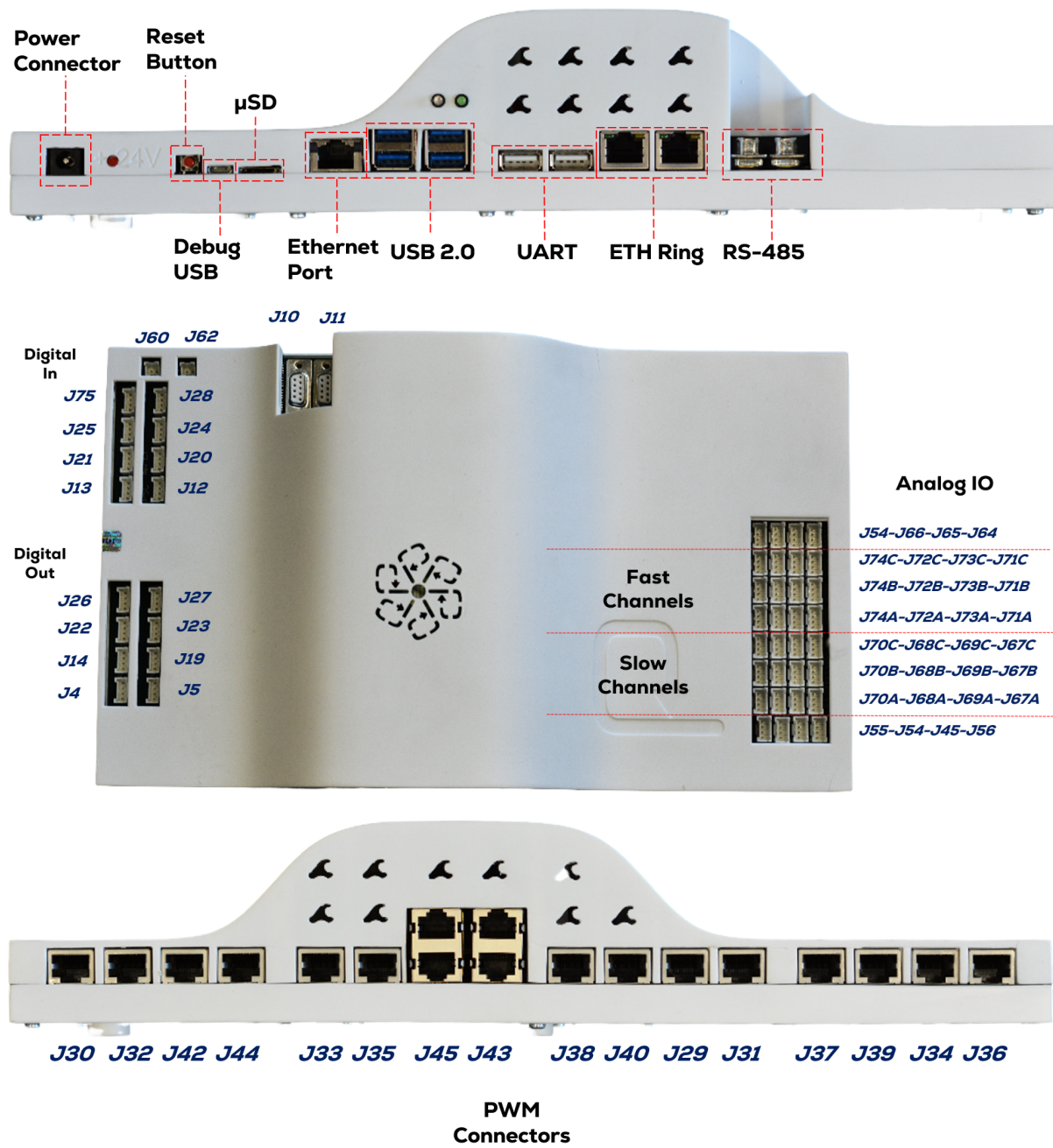
ID	Function	Pin-out	V	BW	Comment
J23	Digital outputs		0-24V	1.5KHz	Multiplexed fast i2c
J24	Digital inputs		0-24V	1.5KHz	Multiplexed fast i2c
J25	Digital inputs		0-24V	1.5KHz	Multiplexed fast i2c
J26	Digital outputs		0-24V	1.5KHz	Multiplexed fast i2c
J27	Digital outputs		0-24V	1.5KHz	Multiplexed fast i2c
J28	Digital inputs		0-24V	1.5KHz	Multiplexed fast i2c
J29	PWM		0-5V	4ns	RS-422 differential
J30	PWM		0-5V	4ns	RS-422 differential
J31	PWM		0-5V	4ns	RS-422 differential
J32	PWM		0-5V	4ns	RS-422 differential
J33	PWM		0-5V	4ns	RS-422 differential

ID	Function	Pin-out	V	BW	Comment
J34	PWM		0-5V	4ns	RS-422 differential
J35	PWM		0-5V	4ns	RS-422 differential
J36	PWM		0-5V	4ns	RS-422 differential
J37	PWM		0-5V	4ns	RS-422 differential
J38	PWM		0-5V	4ns	RS-422 differential
J39	PWM		0-5V	4ns	RS-422 differential
J40	PWM		0-5V	4ns	RS-422 differential

ID	Function	Pin-out	V	BW	Comment
J41	PWM		0-5V	4ns	RS-422 differential
J42	PWM		0-5V	4ns	RS-422 differential
J43	PWM		0-5V	4ns	RS-422 differential
J44	PWM		0-5V	4ns	RS-422 differential
J45	Digital Outputs		0-3.3V	1.5KHz	Multiplexed fast i2c
J50	User LED		0-3.3V		Green panel LED
J52	Digital Outputs		0-3.3V	1.5kHz	Multiplexed fast i2c

ID	Function	Pin-out	V	BW	Comment
J54	DACs		0-3.3V	500 ks/s	Connector pin is the output of DAC IC no analog chain included
J55	Digital Outputs		0-3.3V	1.5kHz	Multiplexed fast i2c
J56	Digital Outputs		0-3.3V	1.5kHz	Multiplexed fast i2c
J58	User LED		0-3.3V		Orange panel LED
J60	DOUT 24V power connector		0-24V	DC	Max total current 10mA per channel
J62	DIN 24V power connector		0-24V	DC	Max total current 10mA per channel
J64	DAC		0-3.3V	500 ks/s	No analog chain included
J65	DAC		0-3.3V	500 ks/s	No analog chain included
J66	DAC		0-3.3V	500 ks/s	No analog chain included
J67ABC	Slow Analog Input		0-5V	500 ks/s	
J68ABC	Slow Analog Input		0-5V	500 ks/s	

ID	Function	Pin-out	V	BW	Comment
J69ABC	Slow Analog Input	 <p>J69A 53375-0410</p>	0-5V	500 ks/s	
J70ABC	Slow Analog Input	 <p>J70A 53375-0410</p>	0-5V	500 ks/s	
J71ABC	Quick Analog Input	 <p>J71A 53375-0410</p>	0-5V	3 ks/s	
J72ABC	Quick Analog Input	 <p>J72A 53375-0410</p>	0-5V	3 ks/s	
J73ABC	Quick Analog Input	 <p>J73A 53375-0410</p>	0-5V	3 ks/s	
J74ABC	Quick Analog Input	 <p>J74A 53375-0410</p>	0-5V	3 ks/s	
J75	Digital inputs	 <p>J75 53375-0410</p>	0-24V	1.5KHz	Multiplexed fast i2c
U36	2x USB 2.0		0-5V		
U38	2x USB 2.0		0-5V		
SW2	Reset	Reset pusher			
XSW1/ XSW2	Power connector	Power connector barrel style (Inner contact is positive terminal)			24V / 60Wmax



5.7 Constraints file

When programming a system that combines programmable devices it is always required to link the programming description of the input/output to a certain pin. Xilinx Vivado allows the definition of connectors in the IP Integrator and its connection with Xilinx-made or custom-made IP blocks. Constraints file is the file in which the connections between the IP integrator ports and real pins are made.

This file has been fully configured and is provided by PSC so the user can directly use it. However, it is provided in an open format in case advanced users want to modify the pin-out or access extra resources available in the SOM by re-routing them to existing pins.

This file also defines the voltage range of each pin or bank. Extreme care should be taken to ensure each pin is configured with the proper voltage standard. Not doing so can damage the SOM I/O bank.

The syntax of the file is the following:

1. `## SOM BANK 1-B - 3.3V`
2. `set_property PACKAGE_PIN J10 [get_ports PMOD1_1_1_2];`
3. `set_property IOSTANDARD LVCMOS33 [get_ports PMOD1_1_1_P2];`
4. `set_property SLEW SLOW [get_ports PMOD1_1_1_P2];`
5. `set_property DRIVE 4 [get_ports PMOD1_1_1_P2];`
6. `# Pin som240_1_b16 - Bank 45 VCCO - som240_1_b13 - IO_L1N_AD15N_45`

The meaning of each line is:

1. Comment for user identification of the bank voltage.
2. Pin identification – use the same label used in Vivado IP integrator.
3. Voltage standard of the pin – USE CARE WHEN MODIFYING THIS FIELD.
4. Buffer special requirements.
5. Buffer special requirements.
6. PSC comment to identify the pin in the SOM high density connector.

It shall be noted that in SmartRCP_Template of Constraints file the linking between the SOC high-density connector and the SmartRCP connector pin has been already made. Doing so requires a deep understanding of the schematic, the analog stage in between, and the Vivado syntax. Users are advised to take care of this file.

All Constraints file port names are the same as the schematic nets and the same as the Vivado IP integrator ports created in the SmartRCP_Template provided. All of them contain the name of the port and the name of the connector involved.

5.8 Ventilation

KRIA K26 is a powerful 10W device so it requires a constant fan to keep it cooled. Two possible fan power connectors are supplied within the CarrierBoard: for 5V and 24V.

By default, PSC delivers all systems with a 5V low-noise high-life fan connected. This fan is mounted on top of KRIA SOM, screwed to its local heatsink.

This fan should not be disconnected, removed, or stopped. Doing so will raise the KRIA's temperature and its thermal cut-off will actuate.

5.9 Booting modes

KRIA K26 SOC supports several booting methods like JTAG, SPI, SD card, USB, etc.

By default, PSC will supply SmartRCP so that it boots from a microSD card directly. Other configurations can be delivered under special request.

5.10 USB programming and debugging

For custom programming and debugging the SmartRCP system, it is required to have the Xilinx Vitis tool installed on a computer connected to SmartRCP microUSB (J6) and NO microSD card inserted in J9.

SmartRCP does not need the use of JTAG, the very same functionality is provided through the microUSB connector (J6). This configuration makes it unnecessary for the user to acquire any extra HW apart from a simple microUSB cable.

The KRIA SOC inside SmartRCP shall be detected by Vitis and its identifier will correspond to its serial number in hexadecimal format.

Programming of SmartRCP will not differ from programming any other Xilinx SOC.

5.11 In Depth description of functional blocks

5.11.1 RS-232

UART3. This is a UART intended to be connected to a computer as it is converted to USB 0-5V standard, its detailed connection and signals can be seen in the figure 7:

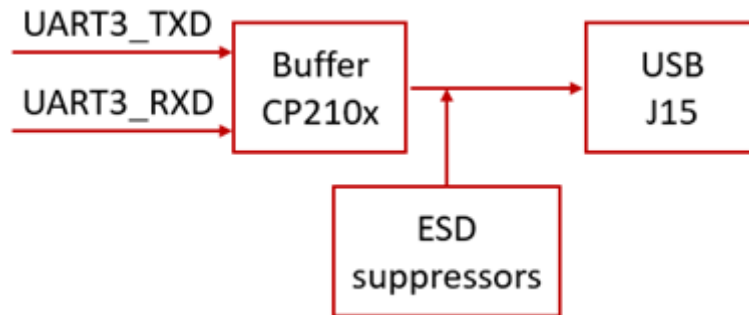


Figure 7: RS232 functional block chain (J15).

UART4. This is a UART intended to be connected to a computer as it is converted to USB **0-5V standard**, its detailed connection and signals can be seen in the figure 8:

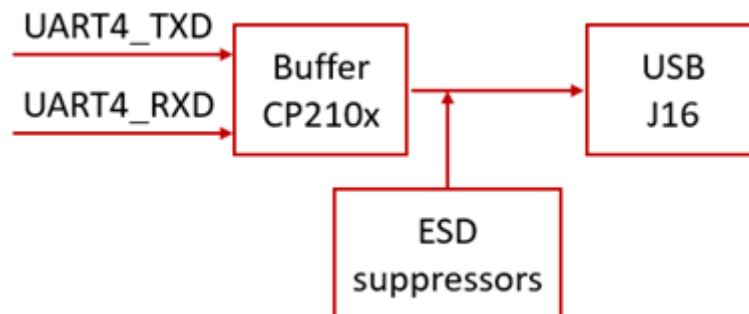


Figure 8: RS232 functional block chain (J16).

5.11.2 RS-485

Two RS-485 ports are connected to KRIA K26 PL pins and the port is not limited in speed (bauds). There is a transceiver that converts and protects those KRIA signals into a full RS485 port.

Jumpers JP1 and JP2, not accessible with the cover, allow the selection of the RS-485 working mode: full duplex or half duplex. By default, PSC will supply SmartRCP with a Full Duplex RS485 configuration in both connectors. Other configurations can be delivered under special request.

RS-485 ports are compatible with **3.3V standard**.

The next figures (9,10) provide all the required information about the chain:

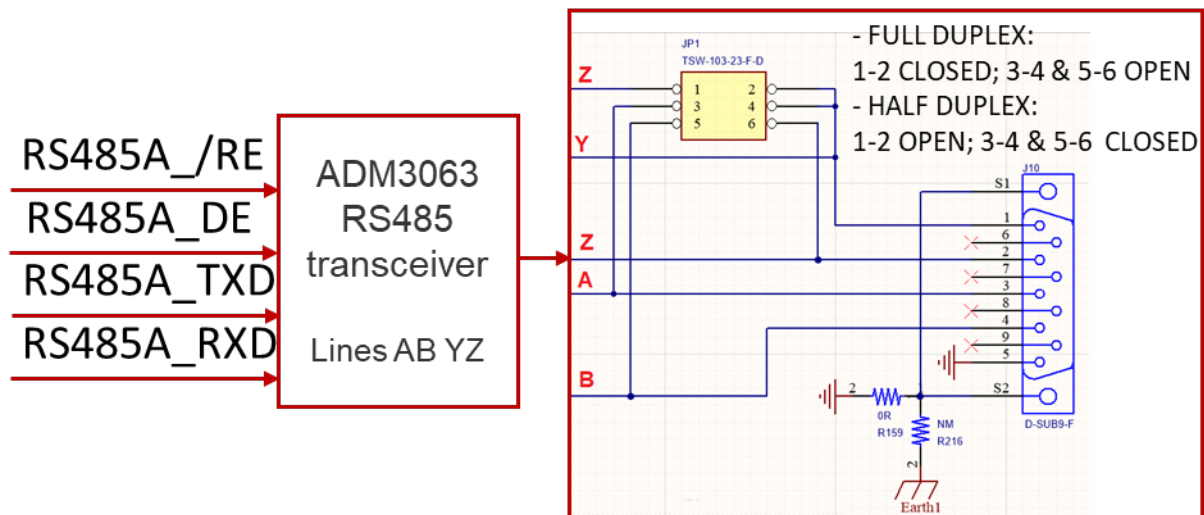


Figure 9: RS485 functional block chain (J10).

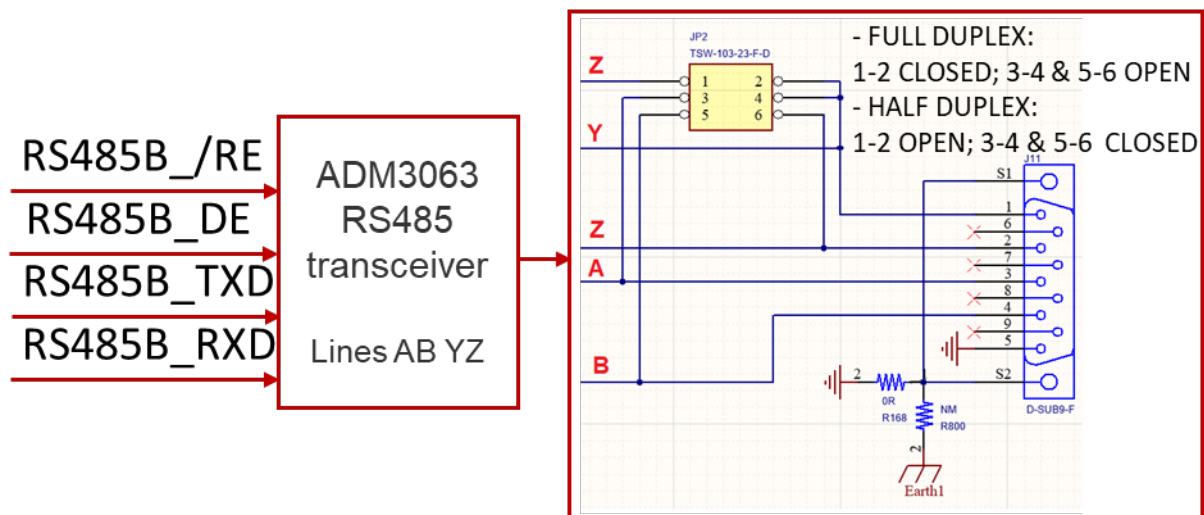


Figure 10: RS485 functional block chain(J11).

5.11.3 User LEDs

The cover has two LEDs available for the user. LEDs connected are LED0 (J58) in orange color and LED1 (J50) in green color. See figure 11:

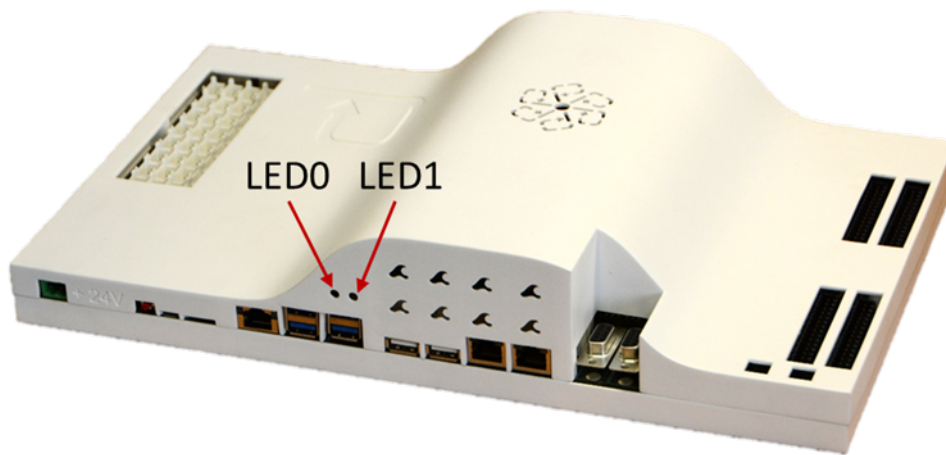


Figure 11: SmartRCP user LED position.

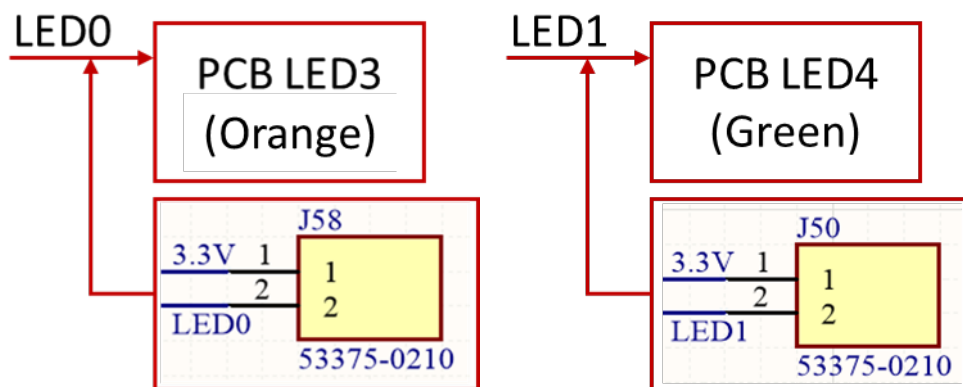


Figure 12: User LEDs functional block chain.

5.11.4 Ethernet ring communication

Ethernet ring communication consists of two different RJ45 connectors galvanically isolated which are connected to LVDS KRIA K26 PL pins by means of a transceiver. These connectors are **3.3V standard compatible**.

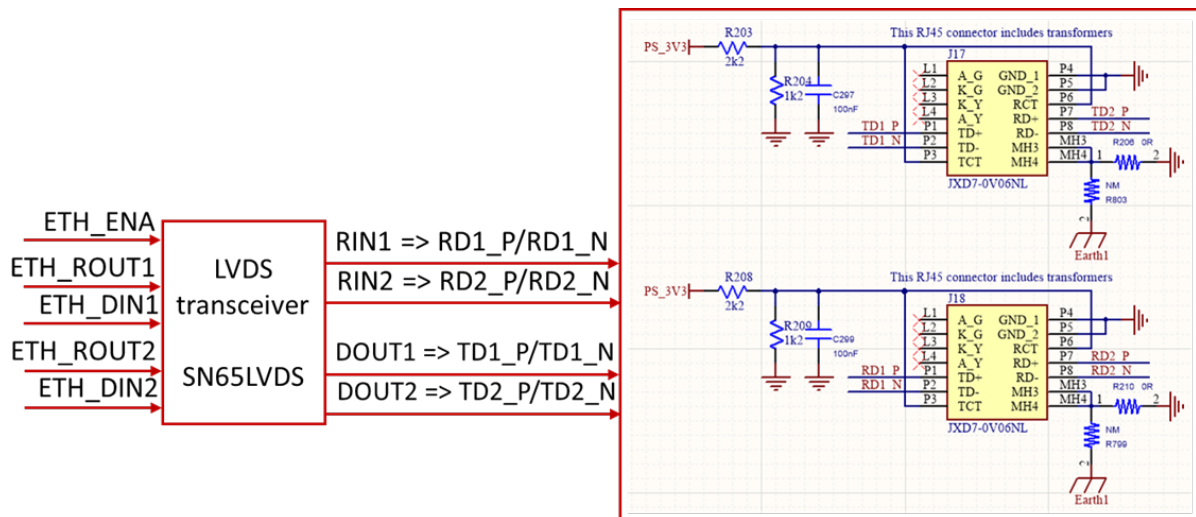


Figure 13: Ethernet ring functional block chain.

5.11.5 Analog output channels

Four quad-channel 16bit 500ks/s **0-5V compatible** DAC converters are provided and accessible with the cover. Output signals are referred to as analog ground.

Figure 14 provides more information about the signals involved and the structure used.

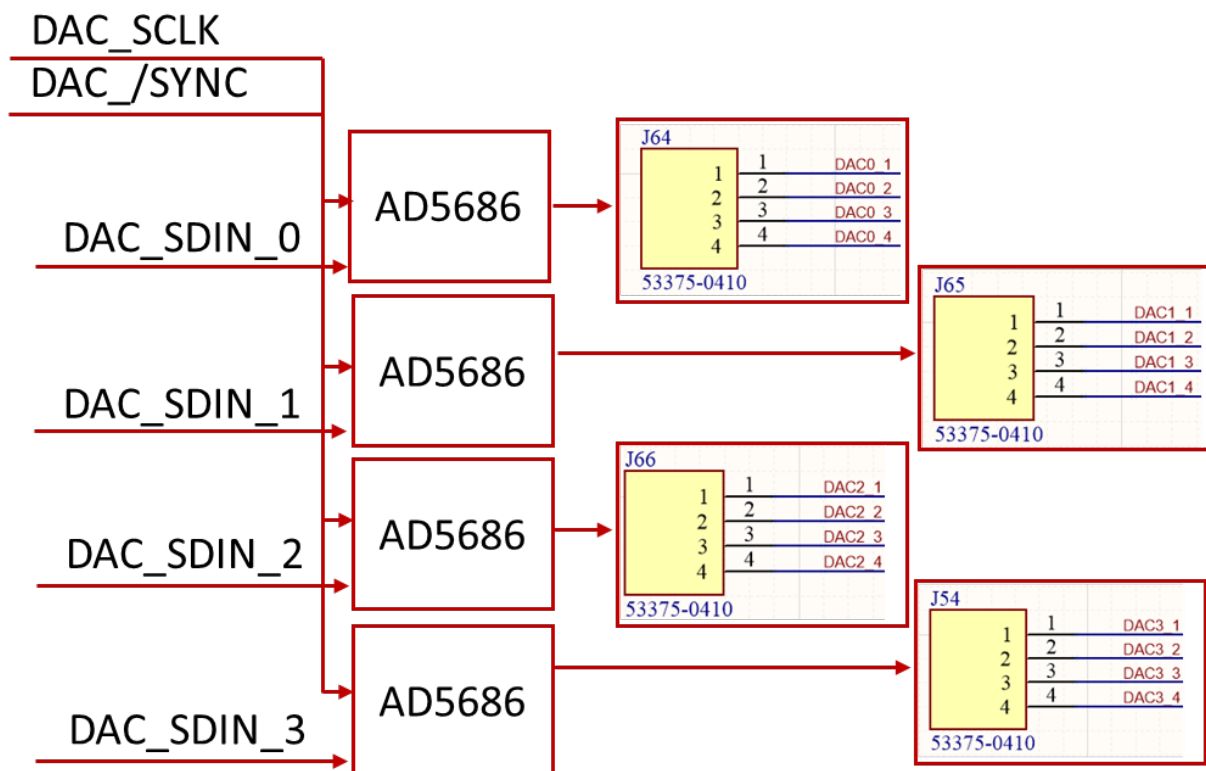


Figure 14: Analog outputs functional block chain.

5.11.6 Digital inputs

32 channels of 0-24V digital inputs are provided which are connected to four 8-bit i2c digital expanders. Each signal is connected to a green LED that will glow when +24V is connected to it, an input 1kHz low pass filter (1st order), and a TVS diode. Up to 10mA can be drained by internal circuitry in each channel.

J62 connector is provided to supply 24Vdc to the user, its ground pin can also be used as an electrical reference.

More details can be found in the figure 15.

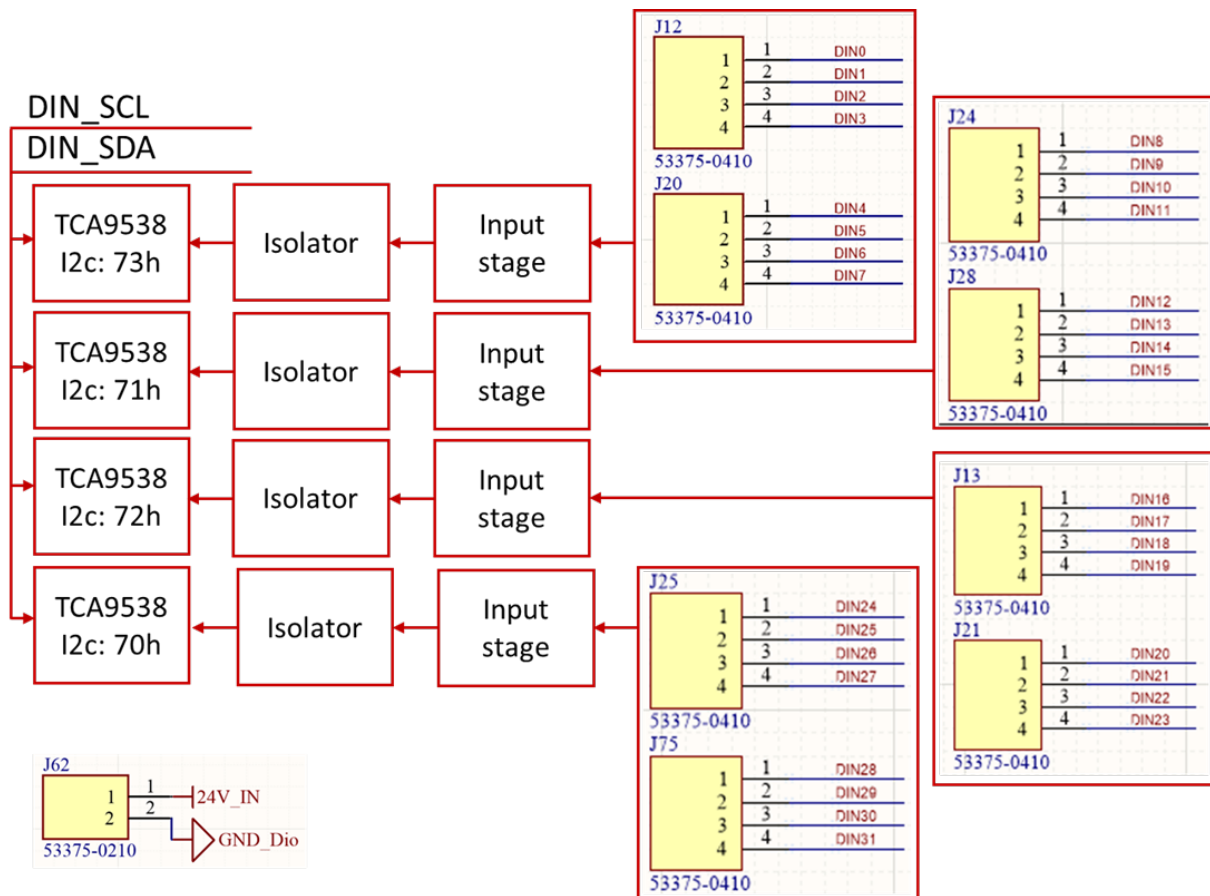


Figure 15: 0-24V Digital inputs functional block chain.

5.11.7 Digital outputs

32 channels of 0-24V digital outputs are provided which are connected to four 8-bit i2c digital expanders. Each signal is connected to a green LED that will glow when +24V is generated by it, each channel is protected by a series PTC.

J60 connector is provided to supply 24Vdc to the user, its ground pin can also be used for referencing the signals.

More details can be found in the figure 16.

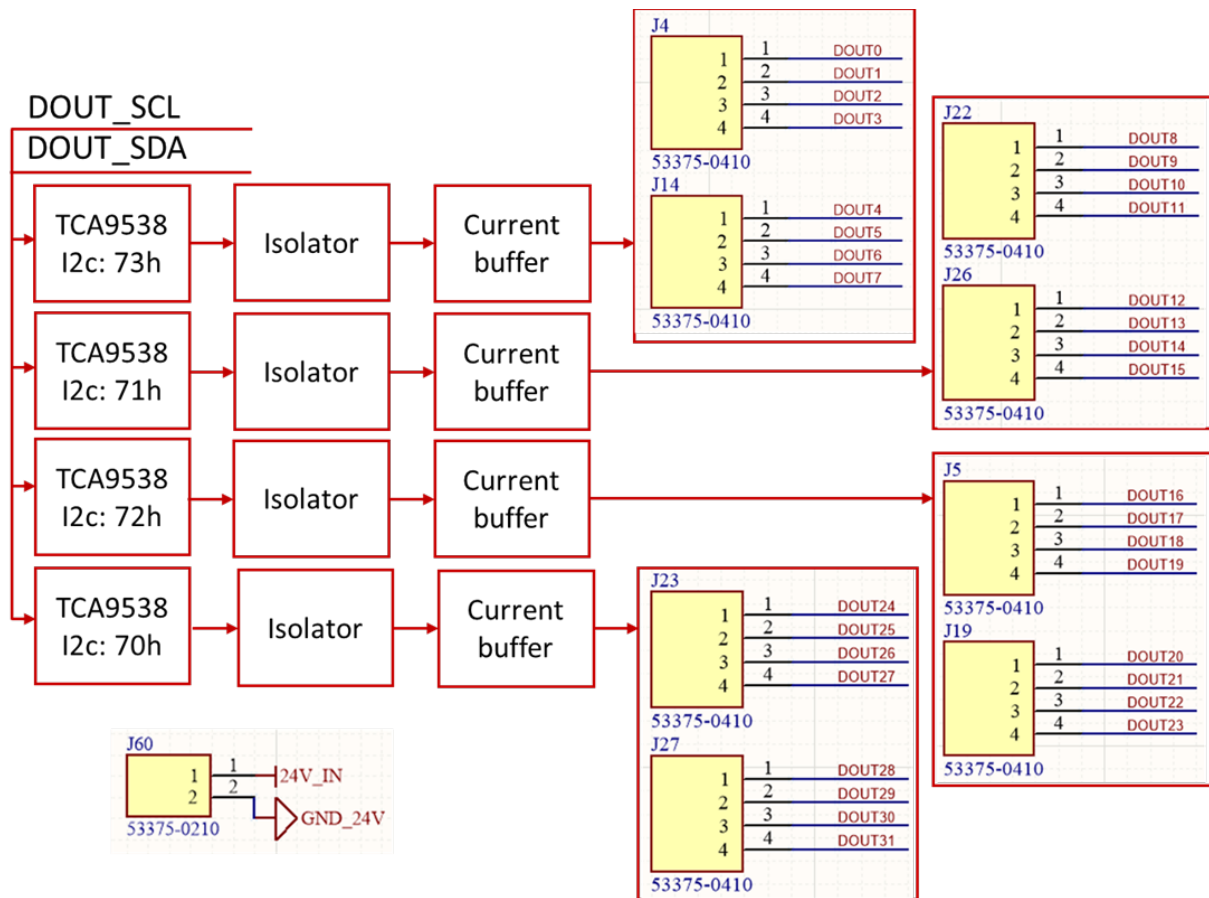


Figure 16: 0-24V Digital outputs functional block chain.

There are 14 additional channels of output signals with a **0-3.3V standard**. These signals have no output analog chain to protect or to limit them. Extreme care shall be taken when using those pins as they have no short-circuit protection.

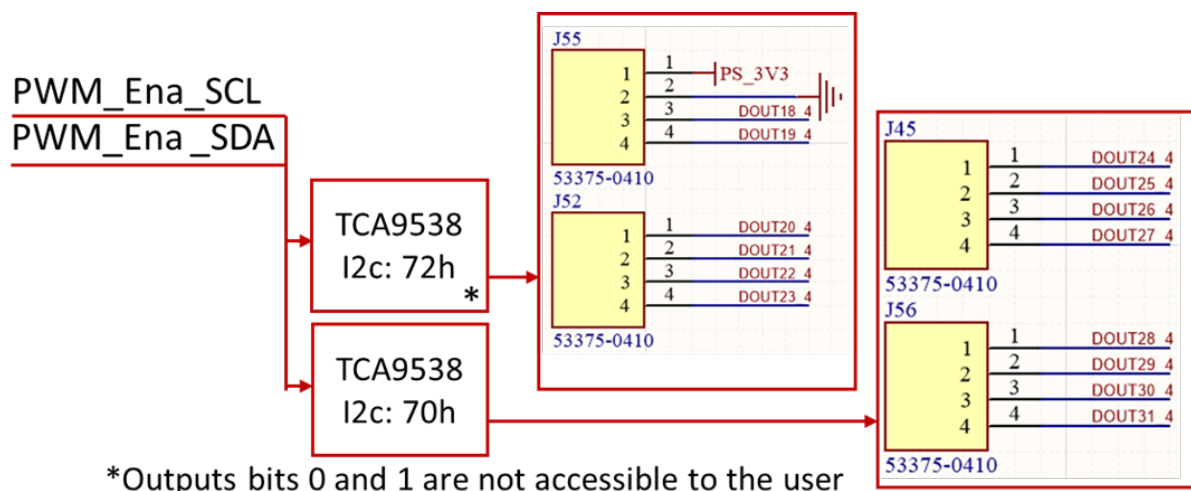


Figure 17: Additional 0-3.3V functional block chain.

5.11.8 PWMs

There are 36 PWM differential channels directly connected to KRIA K26 PL pins. Those signals are RS422 differential and fully isolated.

RJ45 connectors **without** transformers of 2 or 4 PWM channels are placed. These connectors include error (ERRx – CarrierBoard input) and enable (ENAx – CarrierBoard output) signals. Enable signals are multiplexed while error signals are directly connected to KRIA K26 PL pins.

The next figures (18, 19, 20, 21, 22, 23, 24, 25, 26, and 27) provide more info about the different signals included.

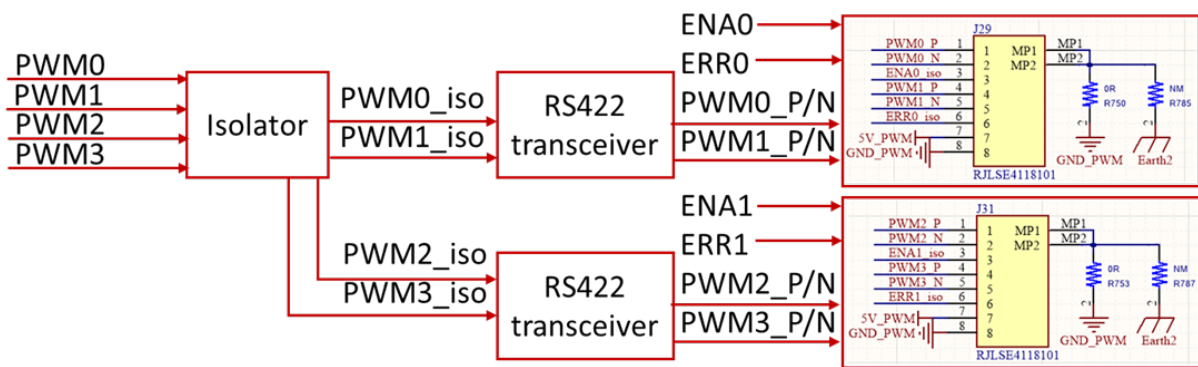


Figure 18: PWM functional block chain (J29 and J31).

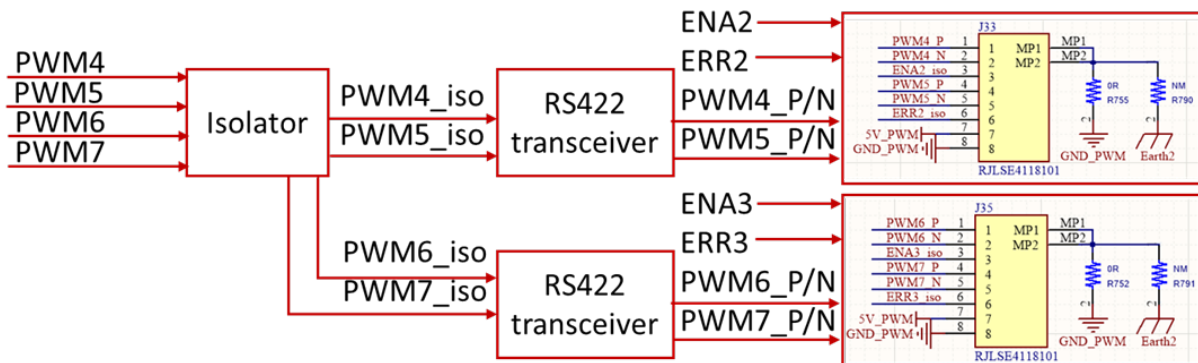


Figure 19: PWM functional block chain (J33 and J35).

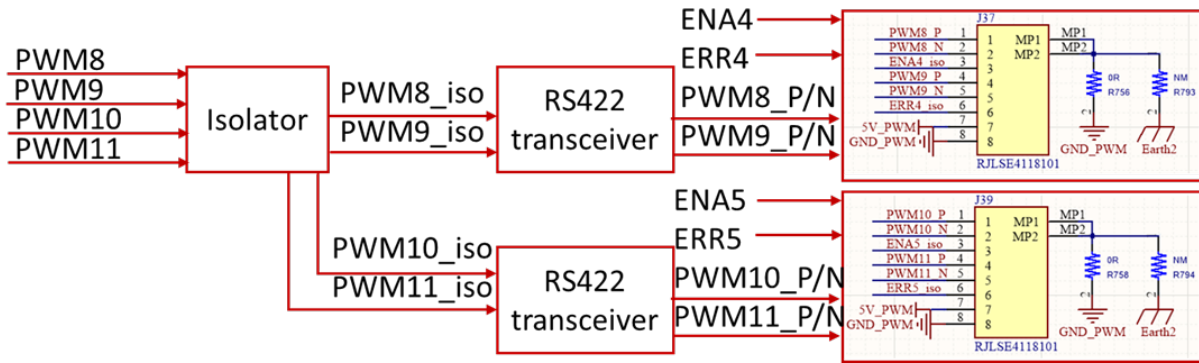


Figure 20: PWM functional block chain (J37 and J39).

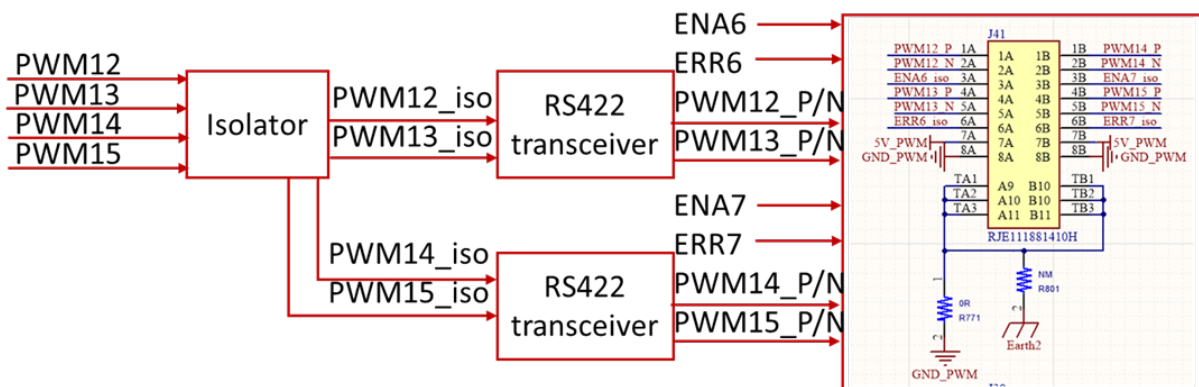


Figure 21: PWM functional block chain (J41).

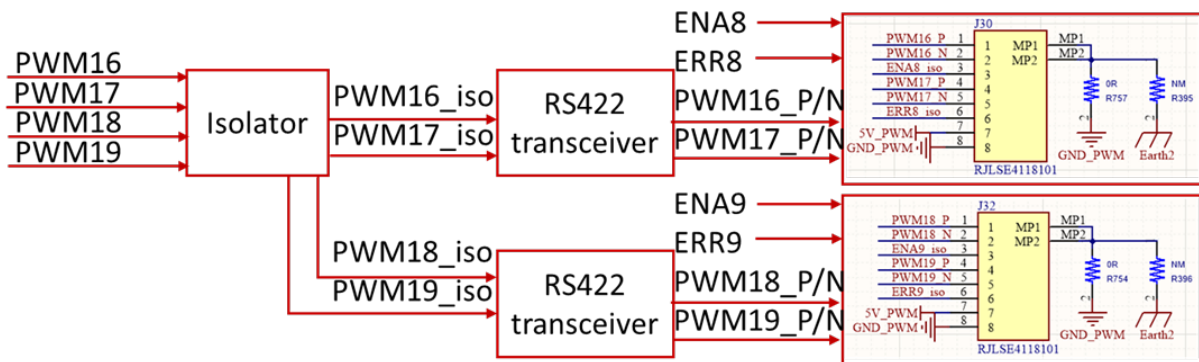


Figure 22: PWM functional block chain (J30 and J32).

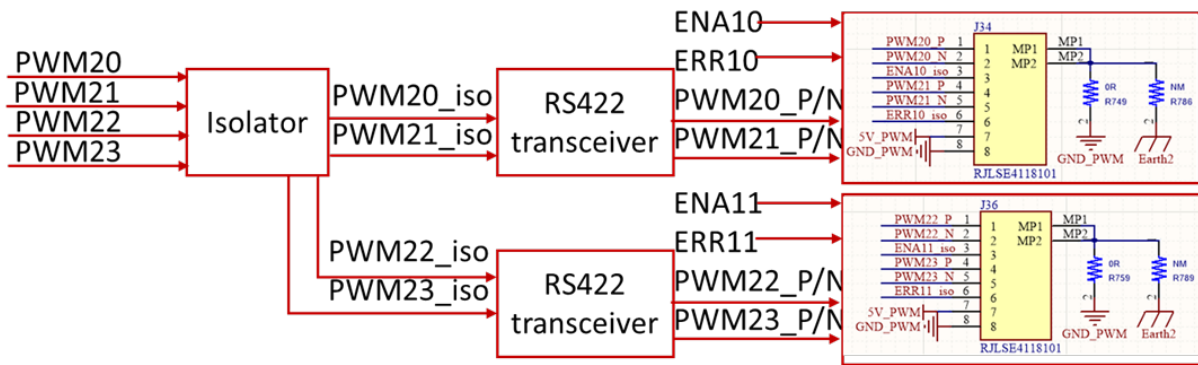


Figure 23: PWM functional block chain (J34 and J36).

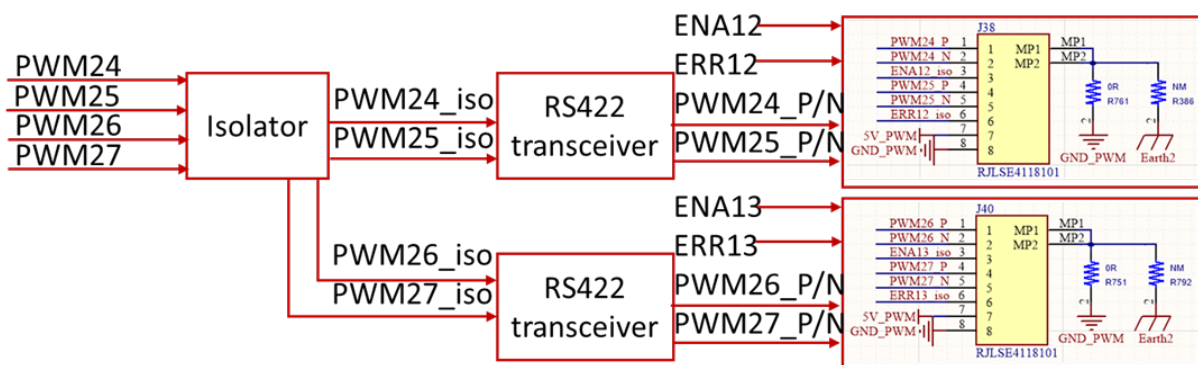


Figure 24: PWM functional block chain (J38 and J40).

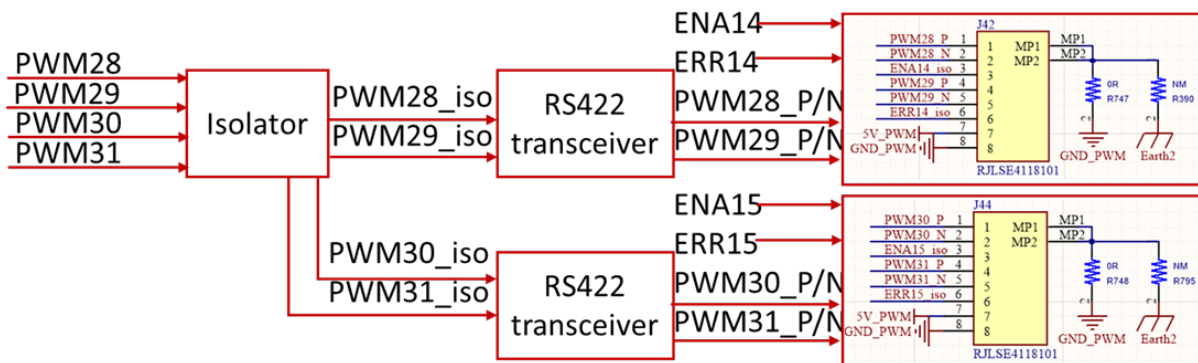


Figure 25: PWM functional block chain (J42 and J44).

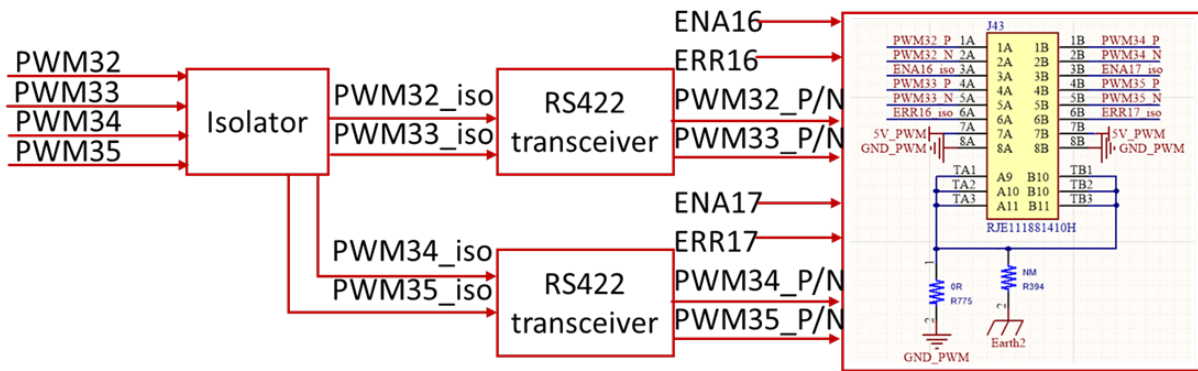


Figure 26: PWM functional block chain (J43).

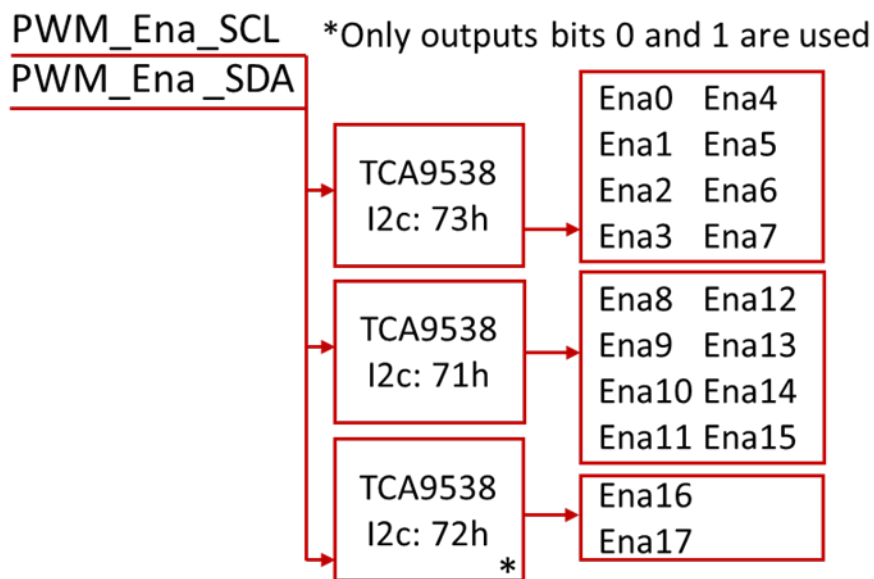


Figure 27: PWM functional block chain. Enable signal detail.

ERRx or PWM error signals are directly connected to KRIA K26 PL pins, those signals are not multiplexed as ENAx (PWM enable) signals.

The connection wire of PWM channels and pin-out can be seen in the figure 28:

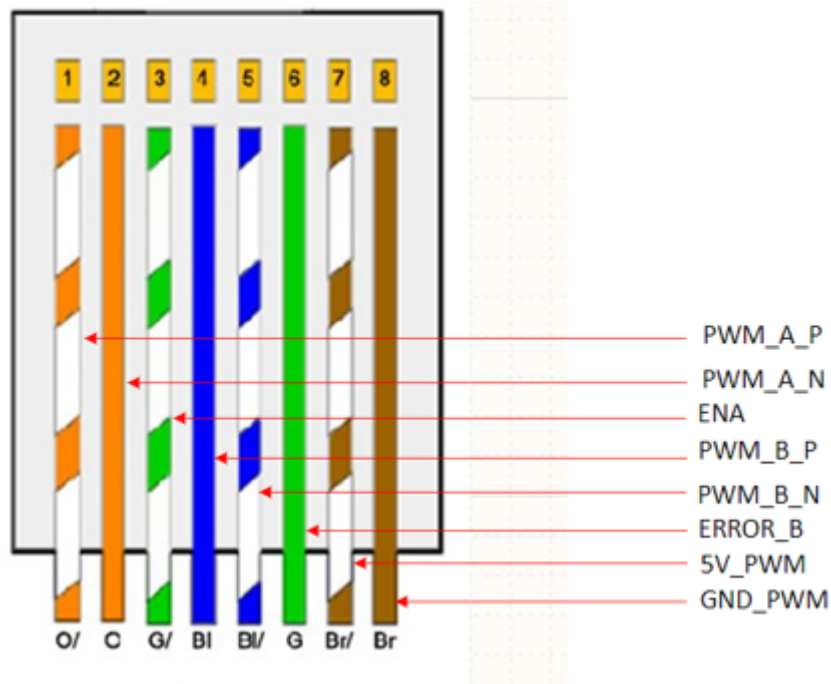


Figure 28: PWM channels RJ45 connector pin out.

5.11.9 Analog input channels

There are 24 analog inputs with two different types of ADC channels: quick (x12 channels at 3Ms/s) and slow (x12 channels at 500ks/s), however, the analog input stage is common to both types. The next figure shows the input analog chain placed in all channels.

The frequency response of the chain is determined by C1, C3, and C4; the gain of the output of the system can be calculated by the next formula:

$$V_{out} = \frac{R_b}{2 \cdot R_a} (V_{in}^+ - V_{in}^-) + V_{offset} \quad (1)$$

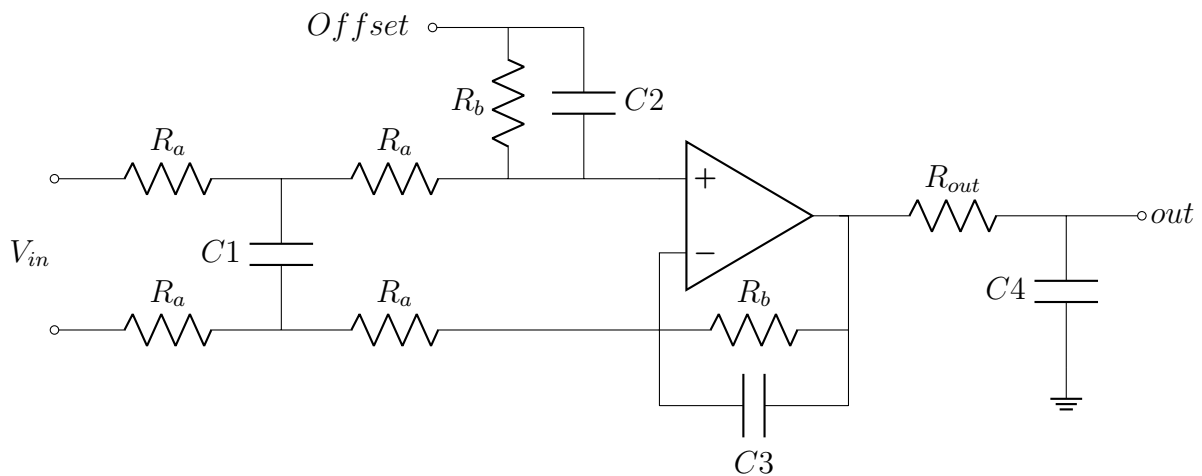


Figure 29: Analog input channels front end

Slow analog channels.

Those channels have the following details:

- Differential high impedance input
- Analog input voltage spam: 0-5V
- Total gain of the analog chain: 1V/V ($5V/5V=1V/V$)
- Value of the components: $R_a=5k\Omega$, $R_b=10k\Omega$, $C_1=56pF$, $C_2=56pF$, $C_3=56pF$, $R_{out}=33\Omega$, $C_4=10nF$, $V_{offset}=0V$ (under special request can be connected to 2.5V)
- ADC features (ADC124S051): 12bits per channel, 4 channels, 500ks/s per channel, SPI
- ADC analog input spam: 0-5V
- Maximum bandwidth of the analog input slow channel (-3dB): 200kHz (limited by C1-C4 capacitors, this BW can be modified by PSC under special request)

Figure 30 provides more details about signals:

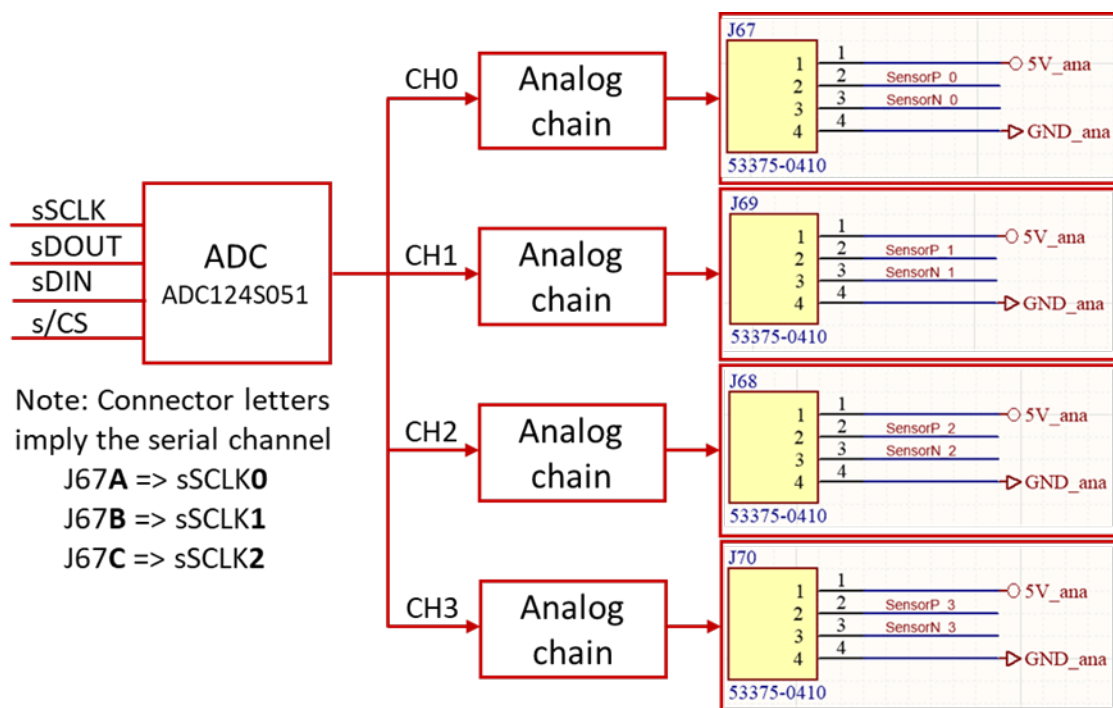


Figure 30: Slow analog input functional block chain.

Quick analog channels.

Those channels have the following details:

- Differential high impedance input
- Analog input voltage spam: 0-5V

- Total gain of the analog chain: $0.66V/V$ ($3.3V/5V=0.66V/V$)
- Value of the components: $R_a=7.5k\Omega$, $R_b=10k\Omega$, $C_1=10pF$, $C_2=10pF$, $C_3=10pF$, $R_{out}=33\Omega$, $C_4=1nF$, $V_{offset}=0V$ (selectable among $0V$ and $1.65V$)
- ADC features (AD7276A): 12bits per channel, 4 channels, 3Ms/s per channel, SPI
- ADC analog input spam: $0-3.3V$
- Maximum bandwidth of the analog input quick channel ($-3dB$): $1MHz$ (limited by C_1-C_4 capacitors this BW can be modified by PSC under special request)

Figure 31 provides more detail about signals:

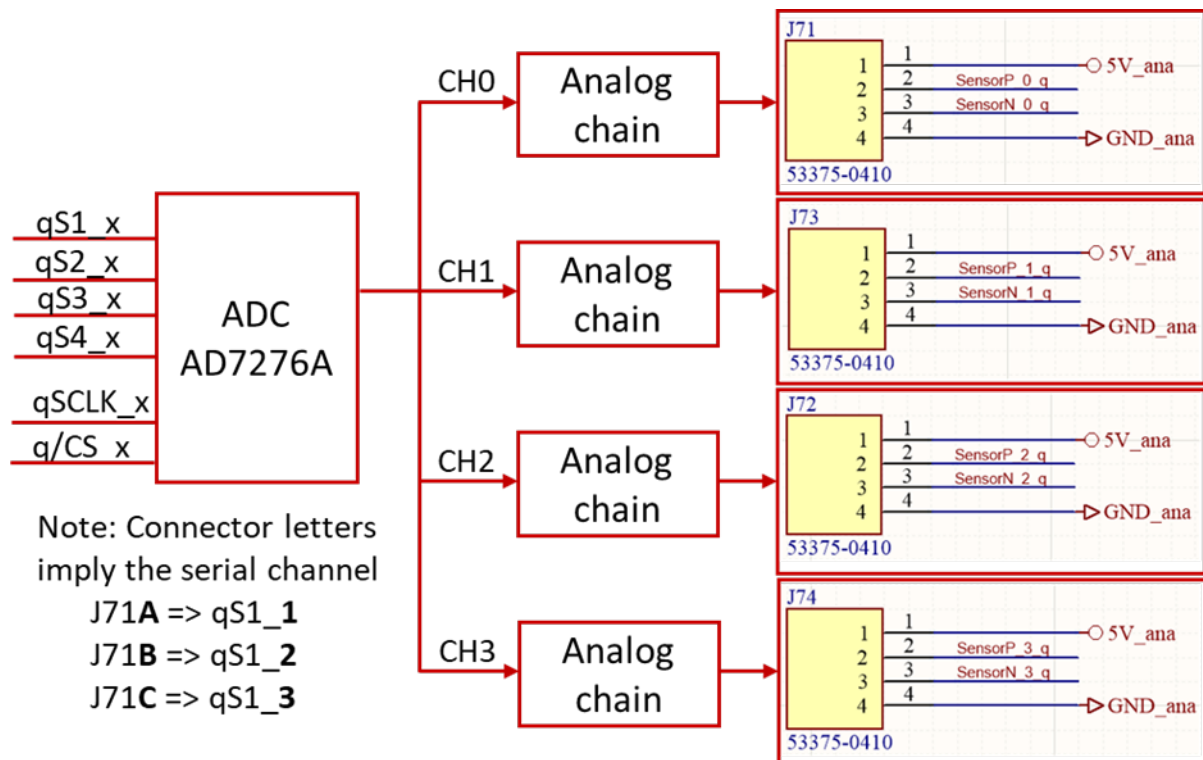


Figure 31: Quick analog input functional block chain.

5.12 Connector shielding

All connectors applicable include the possibility to connect its metallic cover (shield) to an additional EARTH plane accessible through CarrierBoard mounting screw holes. This detail is of paramount importance in the case of controlling high-power converters where EMI can interfere or get introduced in wires or connections not shielded.

Each connector shield is, by default, connected to a regular “gnd”. Connection to EARTH can be done by PSC under special request.

6 Programming user guide

6.1 Development enviroment

To work with the CarrierBoard it is necessary to use Vitis, an ecosystem provided by Xilinx to facilitate the use of its products. This manual provides some basic information about the programs of the Vitis ecosystem. It will also address the use of the template project that PSC provides to ease the use of SmartRCP.

6.1.1 Introduction to Vitis

The software tools required to work with SmartRCP are:

1. **Vivado:** For the development of all the digital designs used in the FPGA, as well as for the basic configuration of the board's peripherals: JTAG, Ethernet, SD, etc.
2. **Vitis:** Development of the hardware part of the design: microprocessor programming, memory management, debugging, etc.
3. **Vitis HLS:** This tool allows high-level synthesis of C++ code and generates digital designs for implementation on FPGA. This tool is optional but PSC recommends its use.

6.1.2 Vitis Download and Installation Instructions

Before starting with the download and installation it is important to note that both the provided template project and the instructions for use are made with Vitis version 2022.1.

1. Click on the link below <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2022-1.html>, select the Web Installer option that corresponds to your operating system (An AMD account must be created before you can install).

 [Xilinx Unified Installer 2022.1: Windows Self Extracting Web Installer \(EXE - 205.92 MB\)](#)

2. Execute the downloaded file, a welcome screen will appear, and click next. Login with the same account. Select "Download and install now", and click next.
3. Select Vitis and click next.
4. Select the following options and click next.

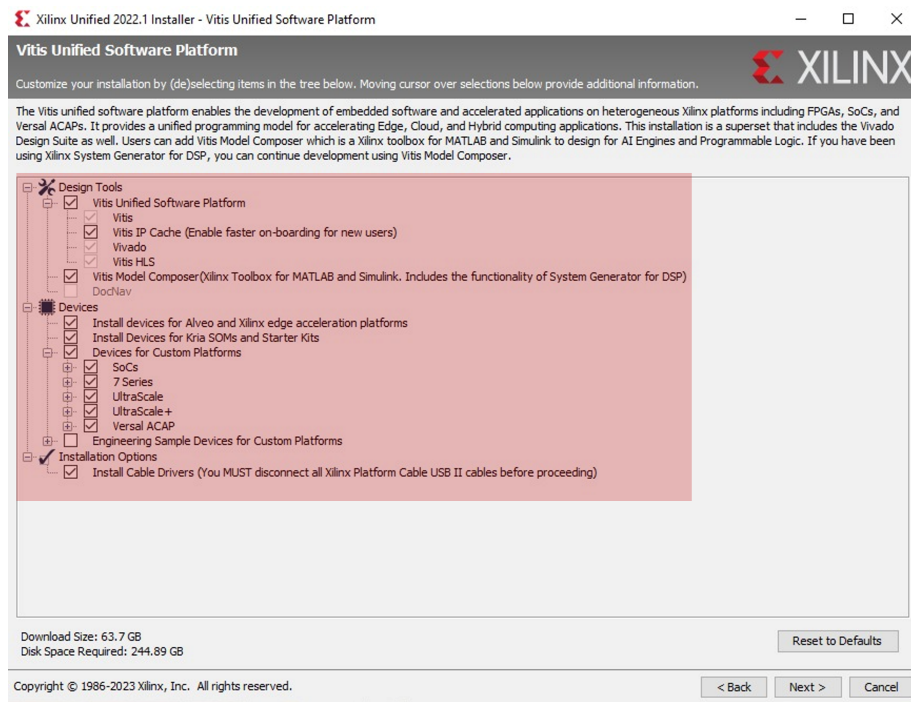


Figure 32: Vitis installing options.

5. Accept all the conditions and click next in all the following windows.
6. Finally, an installation summary tab will appear. Click on install, this process takes some time.

6.2 Vivado software tools

This section defines how to use Vivado to:

- Create a project and the main configuration
- Add an IP
- Use the Constraints file and create Ports
- Generate Bitstream
- Use the SmartRCP_Template

6.2.1 How to create a project and the main configuration

This section contains the basic steps to create a new Vivado project together with the necessary configurations to be made.

1. Open Vivado software, in the main screen you will see several options, go to "Quick Start" and select the option "Create Project".

2. In the first window that pops up click next. In the next window define the name of your project and the location where it will be saved; Leave the option "Create project sub-directory" checked. It is recommended that the location where the project is saved should not be too long, as this can sometimes cause problems. Click next.
3. Leave the following options checked and click on next.

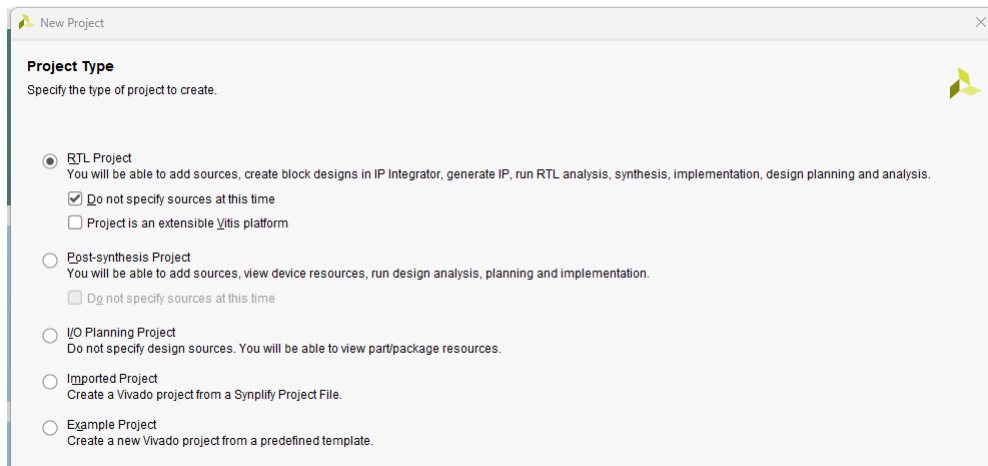


Figure 33: Vivado project options.

4. In the new screen, go to Boards and select "Kria k26i SOM". See figure 34.

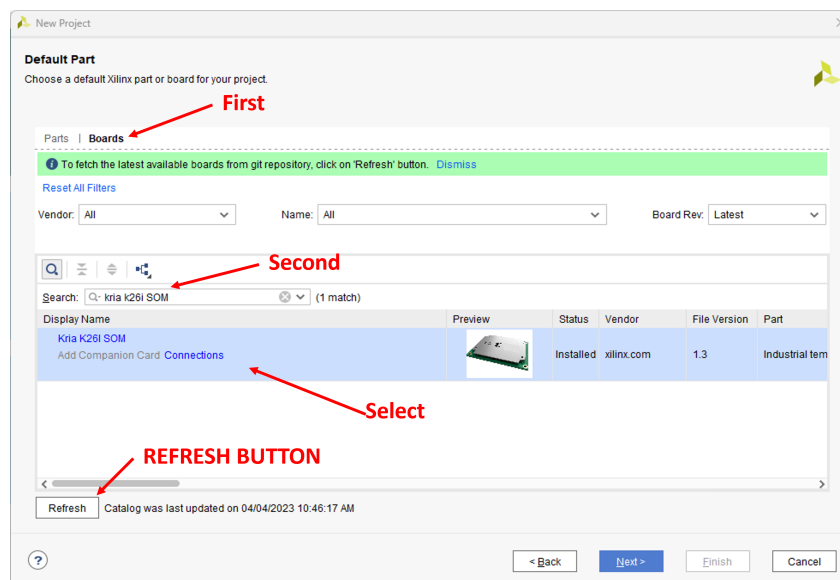


Figure 34: Vivado project part definition.

5. Finally, a summary window appears, click on Finish.
6. The project has already been created. Next is to perform some basic configurations, to do so, go to the project manager area and click on Settings.

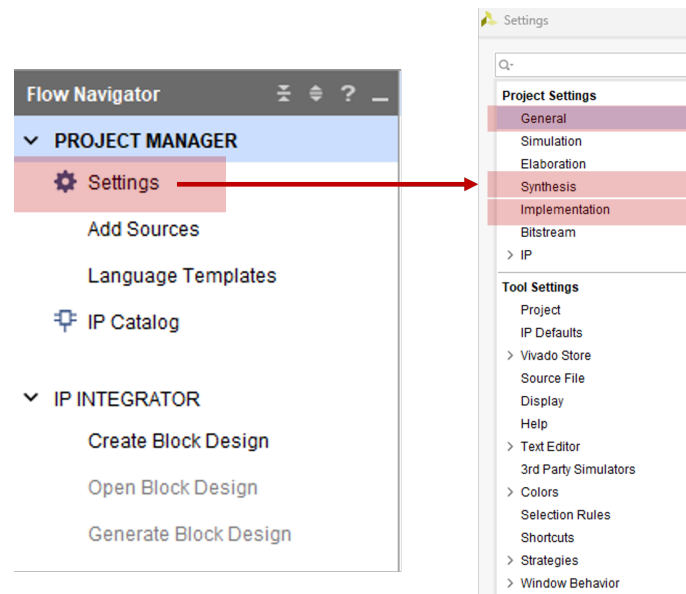
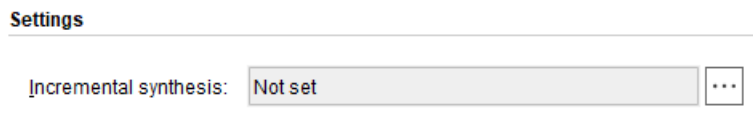
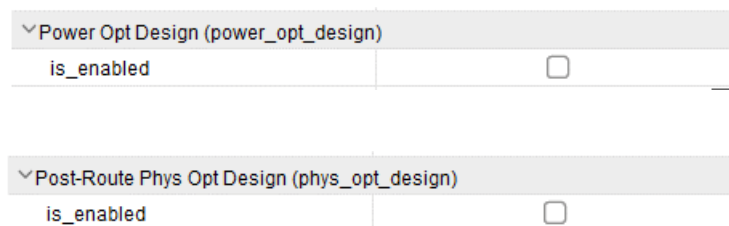


Figure 35: Vivado project option configuration

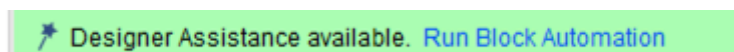
- a. In **“general”**, in the "Target language" area define the digital design language. Select VHDL in this option.
- b. In **“Synthesis”**, click on the three dots next to "Incremental Synthesis" and select the option "disable incremental synthesis" and click OK.



- c. In **“Implementation”** go to settings and disable all optimization options.



7. In "IP INTEGRATOR" click on "Create Block Design". This will create a graphical interface where the user can add his RTL digital designs and IP blocks and interconnect them graphically.
8. Clicking on the "+" icon or alternatively 'ctrl + I' will bring up a search bar with the IPs that Xilinx provides to its users. Search for "Zynq UltraScale+ MPSoC" and double-click on it. It will be added to your central window (the graphical interface of the design).
9. Click on "Run Block Automation", a new window will pop up, click on OK.



10. This IP allows the configuration of various board parameters such as clock frequency, use of various peripherals such as Ethernet, etc.

6.2.2 Add an IP

1. In "Project Manager", go to "IP Catalog", right click anywhere in the white part of the new IP catalog tab that has appeared. In the options that appear, select "add repository", go to the location of the IP you want to add and select it.
2. If the selected file contains valid IPs, a pop-up window appears with the number of IPs found.

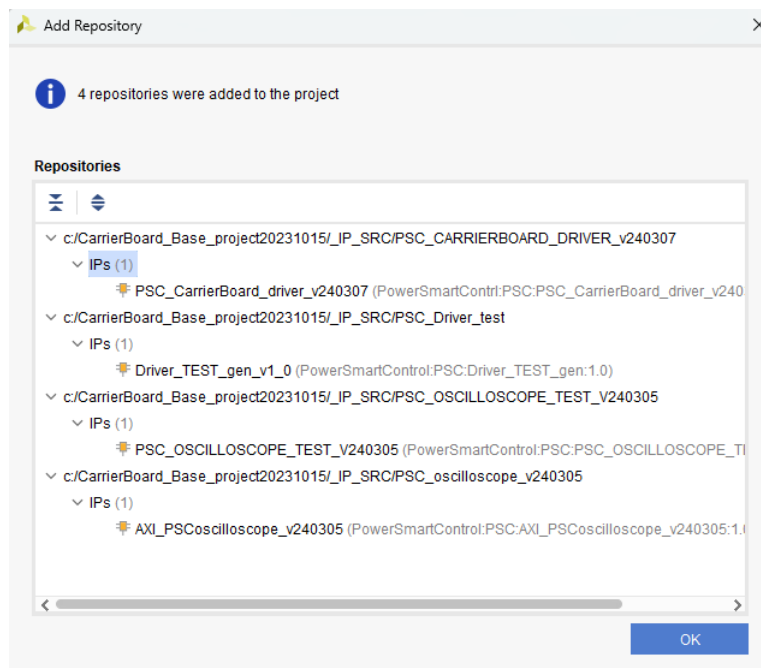


Figure 36: IP repository in Vivado project.

3. In "Diagram", it is now possible to add the IPs to the project in the usual way using the "+" button and searching for it by name.

6.2.3 Use the Constraints and create Ports

The Constraints file contains the declaration of all the physical pins of the SoC. These pins assign the input/output signals handled in Vivado IP integrator with the physical pins of the SOC.

Steps to add the Constraints file:

1. In Project Manager, select "Add Sources"
2. In Add Sources, select "Add or create Constraints" and click next.
3. Select the option to add source
4. Add the file and click OK.

The constraints file defines the characteristics of the pins, assigns them a name and locates them with respect to the physical pins of the SOC. The use of ports is necessary to relate the name assigned to the pins with the input and output signals of the FPGA.

Steps for port declaration:

1. Right click on the signal and select the "Make External" (ctrl + T) option.

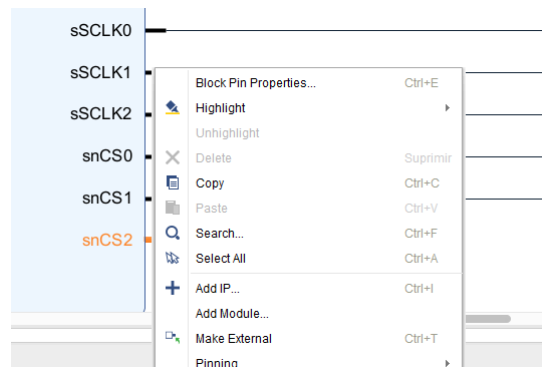


Figure 37: Creating an external port.

2. Change the port name to match the name of the output you want to use from the constraints.

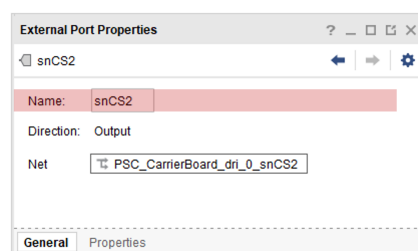



Figure 38: Changing name to an external port.

6.2.4 Generate Bitstream

Bitstream is the file that defines the HW configuration of the PL side of the SOC device. This file is the output of Vivado program and the input into Vitis program. To generate this file, follow the next steps:

1. Click on "validated Design" or F6 
2. Click on "create HDL wrapper"

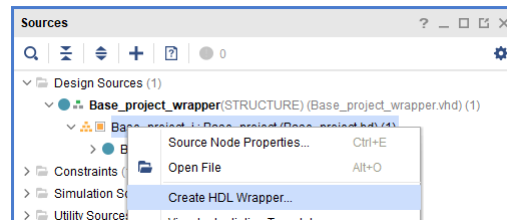


Figure 39: Creating HDL wrapper.

3. Click on “Generate Output Products”, a new window emerges, make sure the following settings are checked.

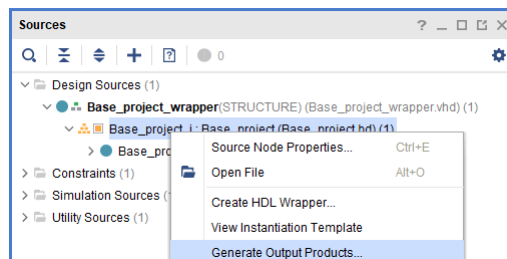


Figure 40: Generating output products.

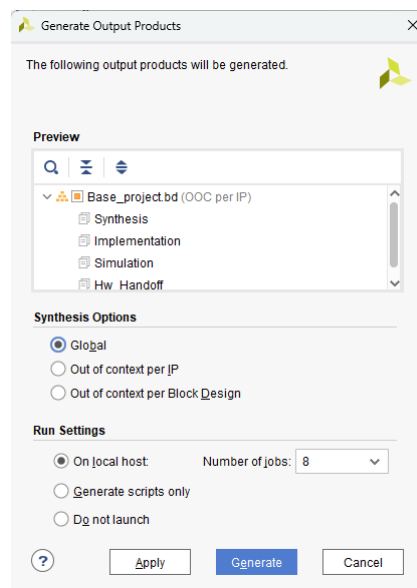
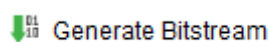


Figure 41: Generating bitstream.

4. Click on Generate Bitstream and click ok in the pop-up window.



5. To continue with the design flow in the other tools of the Vitis environment it is necessary to export the synthesized design together with the bitstream. Click File/Export/Export Hardware.

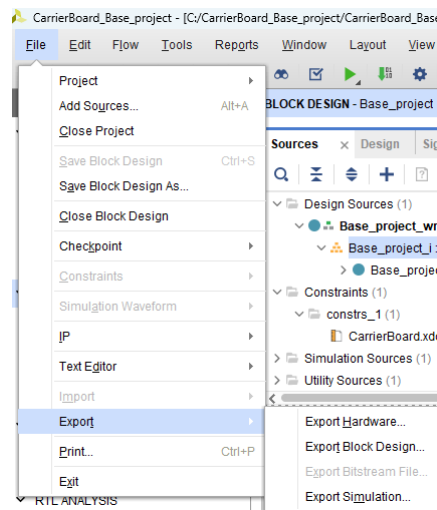


Figure 42: Exporting HW.

6. Click next in the tabs, make sure that in the second tab the option "include bitstream" is checked.

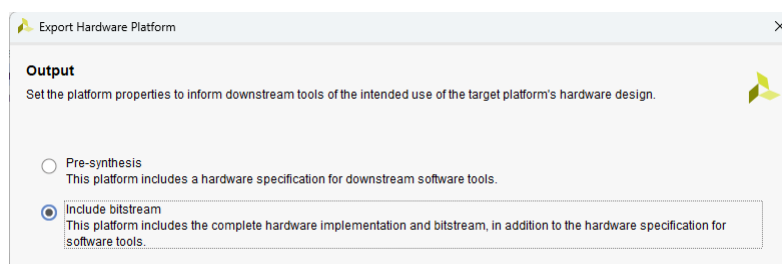


Figure 43: Exporting HW with bitstream.

6.2.5 How to use SmartRCP_Template provided by PSC

An introductory overview of the Xilinx toolchain has been provided; however, to ease the use of SmartRCP system, PSC provides a full template so that most of the configurations have been fully made. This SmartRCP_Template has already defined:

1. All the constraints and ports available in SmartRCP
2. All the IP blocks required to manage SmartRCP system
3. All the configuration of the clocks, RAM, peripherals and its connection
4. Some dummy blocks to generate data and convert the template into a working example

The SmartRCP_Template project provided by PSC is: "SmartRCP_Template". This project contains:

1. Vivado project
2. IP sources already included in Vivado project (_IP_SRC). That includes Control-Board drivers, Oscilloscope and TEST IP.

3. Constraints source already included in Vivado project (src_constraints)
4. Oscilloscope and TEST application microcontroller source files (ELF_boot_src)

To use this template Vivado project, it is important to take into account a series of requirements:

- Project name: “SmartRCP_Template”
- **Vivado version 2022.1.** Its correct operation is only guaranteed under this version.
- For the project to run properly, it must be located on the following path:
- “C: SmartRCP_Template”

Steps for using and customizing SmartRCP_Template:

1. Unzip the project.
2. Open Vivado 2022.1 and select the “Open Project” option, a file browser will appear, go to the project location and open the project file SmartRCP_Template.xpr

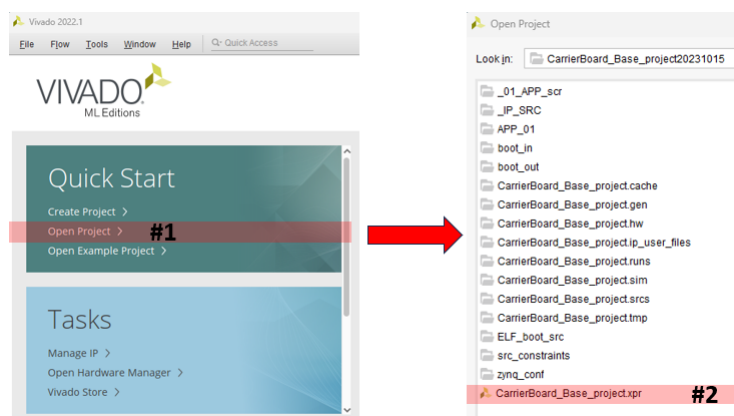


Figure 44: Opening SmartRCP_Template.xpr.

3. At this point the Vivado project will open. As the template project includes all the SOC configuration, there is no need to modify the Zynq Processing System IP block
4. The “Block Diagram” will open. See next Figure. The following types of blocks are placed and connected:
 - (a) The oscilloscope block includes the FPGA data acquisition side of PSC-Oscilloscope product.
 - (b) TEST blocks are used in this example to introduce known data into the drivers. When adapting the template project to a user real application, these TEST blocks shall be removed, and the inputs of the drivers connected to the required user IP blocks. More detail about the procedure is given in the proper section.
 - (c) The oscilloscope block includes the FPGA data acquisition side of PSC-Oscilloscope product.

- (d) ILA block is provided by Xilinx and can be used as a logic analyzer. Its potentiality is covered in more depth in the proper part of this manual.
- (e) Driver_CarrierBoard is an IP which integrates all the drivers required to make the integration of SmartRCP system with CarrierBoard.
- (f) UltraScale+ setup blocks are used to define the clocks and main configuration of the SOC. The user must be cautious when modifying this configuration. PSC provides this block fully configured.

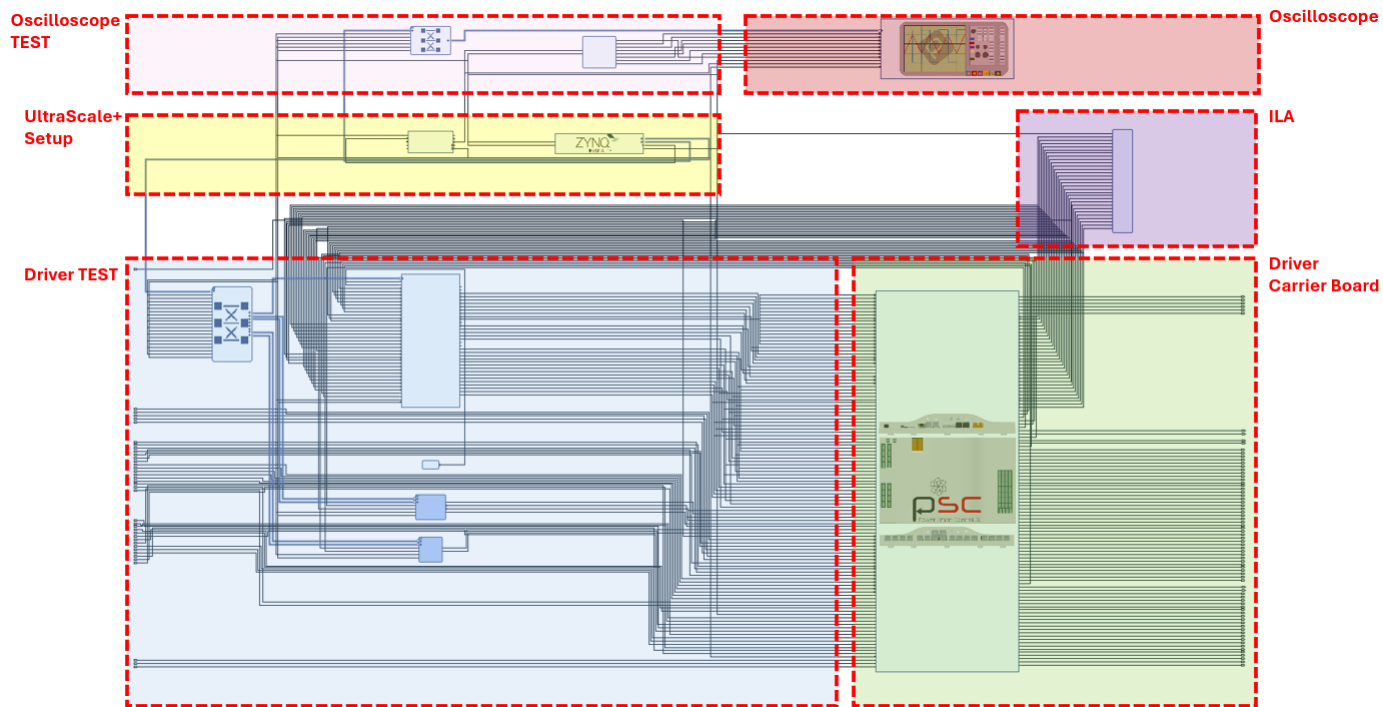


Figure 45: IP Integrator view of SmartRCP_Template Vivado project.

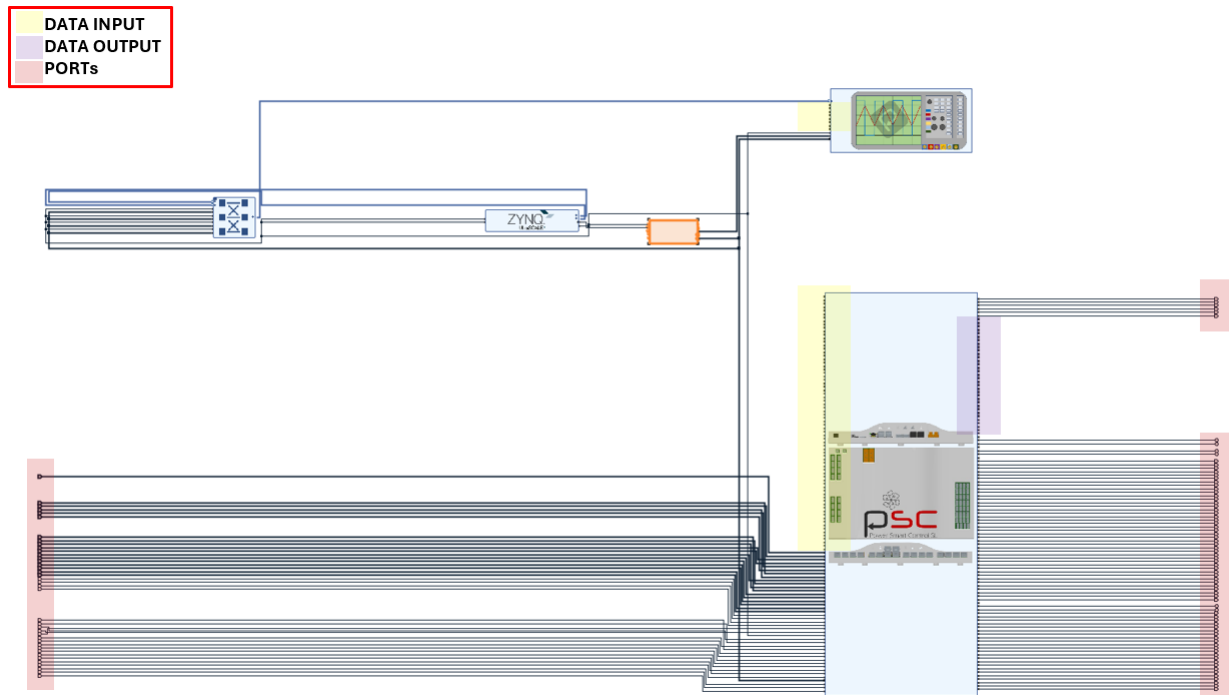


Figure 46: IP Integrator view of SmartRCP _Template Vivado project without TEST blocks.

- (g) Oscilloscope functionality can be used by the customer or deleted. All unused driver IP data inputs should be connected to constants to avoid problems during synthesis, the value of these constants is not relevant.
- (h) When all the IP setting and connections are done, the user shall make the synthesis and bitstream. Those steps have been covered in the previous chapter.

Advanced configurations on the SOC:

SmartRCP_Template provides a full configuration of the SOC, however, that configuration can be modified by the user. Some typical modifications can be:

- a) Creating interrupts
- b) Modifying the baud rate of UARTs or RS485
- c) Modifying the speed rate of ethernet or USB
- d) Adding extra clocks

6.3 Documentation of Driver IPs included in SmartRCP_Template

6.3.1 PSC_CarrierBoard_driv IP

This IP contains all drivers required to behave as a bridge between SmartRCP and CarrierBoard. It contains the drivers for all the peripherals (PWM, DAC, ADC, Digital inputs, Digital outputs, etc.)

In the IP two types of signals can be distinguished:

- “DATA” marked signals: these are used to read and write to the different peripherals. Its names begin with “DATA” followed by its function and the connector to which it belongs on CarrierBoard. User shall modify the connection of those signals to adapt it to their needs.
- Other signals: these pins do not carry the word “DATA” and have the same name as on the schematic, together with information on the port to which they are attached for easy identification. The user shall make no modifications to the connection of those signals. By doing so a HW malfunctioning can be created.

It follows a description of each of the peripherals that are controlled with this IP, their mode of use and the ports and signals contained.

Typical applications will not require the use of all the ports of this IP, the unused input ports must be connected to a constant IP block. Otherwise, Vivado will launch synthesis error. The value of that constant/s is not relevant.

This IP is designed to operate with a 200MHz clock, its operation is not guaranteed if a different clock frequency is used.

1. RS485 communication interface:

Description:

This IP provides a set of input and output data ports to manage the communication by RS485.

Ports denoted with “HW ports” must not be modified by the user. The user can modify the connection of “USER ports”

How to use:

It must be noted that the cross between Rx and Tx signals has been already done. So, they must not be crossed again in user design by the user. The baud rate by default is 115200, user can modify this value.

For the use of the RS485 the user must generate 3 output signals:

- nRE: Receiver Enable Input. This is an active low input. Driving this input low enables the receiver and driving it high disables the receiver.
- DE: Driver Enable. A high level on this pin enables the driver differential outputs, ‘A’ and ‘B’. A low level places the driver output into a high impedance state.
- Data: Data to be transmitted

The IP provides an additional input signal where the user can read the received data.

Connection:

Name	Size	Type	Description	Use
<ul style="list-style-type: none"> – Data_RS485A_DE_J10 – Data_RS485A_TXD_2_J10 – Data_RS485A_nRE_J10 – Data_RS485B_DE_J11 – Data_RS485B_TXD_2_J11 – Data_RS485B_nRE_J11 	1bit	Input	Control lines of each of the RS485 connectors	USER ports
<ul style="list-style-type: none"> – Data_RS485A_RXD_2_J10 – Data_RS485B_RXD_2_J11 	1bit	Output	Data received by the RS485 connector	USER ports
<ul style="list-style-type: none"> – RS485A_RXD_2_J10 – RS485B_RXD_2_J11 	1bit	Input	Connection to the physical pins of the SoC.	HW ports
<ul style="list-style-type: none"> – RS485A_RXD_2_J10 – RS485B_RXD_2_J11 	1bit		Connection to the physical pins of the SoC.	HW ports

Connectors: J10 and J11

2. UART communication interface

Description:

This IP provides a set of input and output data ports to manage the communication by RS232 or UART.

Ports denoted with “HW ports” must not be modified by the user. The user can modify the connection of “USER ports”

How to use:

It must be noted that the cross between Rx and Tx signals has already been done. So, they must not be crossed again in user design by the user. The baud rate by default is 115200, the user can modify this value.

Connection:

Name	Size	Type	Description	Use
<ul style="list-style-type: none"> Data_UART3_RXD_2_J15 Data_UART4_RXD_2_J16 	1bit	Input	Data transmitted	USER ports
<ul style="list-style-type: none"> Data_UART3_TXD_2_J15 Data_UART4_TXD_2_J16 	1bit	Output	Data received	USER ports
<ul style="list-style-type: none"> UART3_TXD_2_J15 UART4_TXD_2_J16 	1bit	Input	Connection to the physical pins of the SoC.	HW ports
<ul style="list-style-type: none"> UART3_RXD_2_J15 UART4_RXD_2_J16 	1bit	Output	Connection to the physical pins of the SoC.	HW ports

Connectors: J15 and J16

3. USER LEDs:

Description:

There are 2 available LEDs: LED0 and LED1. How to use:

Connecting the proper port to '1' will turn on the LED, connecting to '0' will turn it off. LED0 is orange and LED1 is green.

Use denoted with "HW ports" must not be modified by the user. User can modify the connection of "USER ports".

Connection:

Name	Size	Type	Description	Use
<ul style="list-style-type: none"> UART3_RXD_2_J15 UART4_RXD_2_J16 	1bit	Input	LED data input ('1' implies 'ON')	USER ports
<ul style="list-style-type: none"> Data_LED0 Data_LED1 	1bit	Output	Connection to the physical pins of the SoC.	HW ports

Connectors: Ready connected with panel LEDs

4. Ethernet Ring communication interface:

Description:

The IP provides signals to manage a proper communication protocol through ethernet by using two RJ45 connectors.

Connector J17 is used for data transmission, it can transmit two channels in parallel. Connector J18 is used for receiving data, it can receive two channels in parallel.

How to use:

It is assumed that user has developed its own communication protocol and has an IP that implements it. An enable signal must be generated by the user as well to enable/disable the HW.

The communication channels are independent, user can use one channel, both or none; this applies both for receiving and transmitting.

Ports denoted with “HW ports” must not be modified by the user. User can modify the connection of “USER ports”.

Connection:

Name	Size	Type	Description	Use
DATA_ETH1_J17 ← TX CH1 DATA_ETH2_J17 ← TX CH2 DATA_ETH2_ena_0 ← ENABLE/DISABLE SIGNAL	1bit	Input	Transmitting and enabling signals	USER ports
Data_ETH1_J18 ← RX CH1 Data_ETH2_J18 ← RX CH2	1bit	Output	Receiving signals	USER ports
ETH_Rout1_J18 ← RX CH1 ETH_Rout2_J18 ← RX CH2	1bit	Input	Connection to the physical pins of the SoC.	HW ports
ETH_Din1_J17 ← TX CH1 ETH_Din2_J17 ← TX CH2	1bit	Output	Connection to the physical pins of the SoC.	HW ports

RJ45 connectors are wired in the way shown in next figure. Each transmitting/receiving channel contains two wires. CH1 use pins 1 and 2 and CH2 use pins 7 and 8.

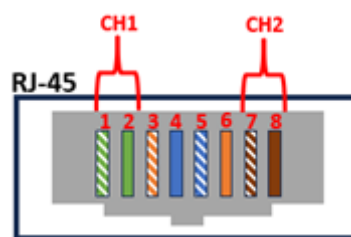


Figure 47: Ethernet Ring RJ45 signal wiring.

Connectors:J17 and J18.

5. Analog outputs:

Description:

This IP provides all the necessary data inputs to control the 16-bits 16 DAC-output-channels available in CarrierBoard.

The DAC has an output voltage range of 0-5V. The output voltage value of the DAC is refreshed every 4.44µs in this IP.

These outputs are directly connected to the output of the DACs, care must be taken when using or connecting those signals.

How to use:

Ports denoted with “HW ports” must not be modified by the user. User can modify the connection of “USER ports”.

The expression needed to calculate the data required to obtain a voltage value at the ADC output is as follows:

$$\text{Data} = (2^{16} - 1) \cdot \frac{v_{\text{out}}}{v_{\text{ref}}} \quad (2)$$

v_{out} = Voltage to be written to the DAC.

v_{ref} = 5V

Data = The data used as input to the IP (with a size of 16 bits).

The nomenclature used in the DAC input data is the following:

DATA_DAC_<m>_<n>_J<xx>

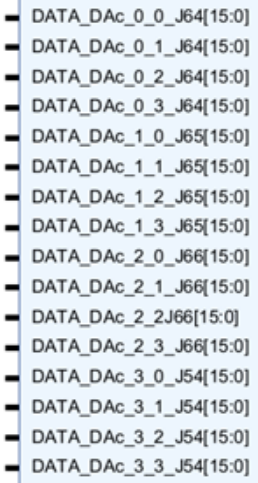
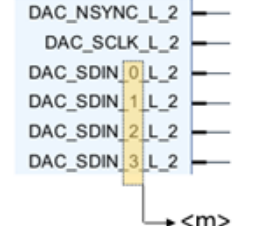
<m> and J<xx>: identifies the connector to which the output belongs.

<n>: identifies the pin inside each connector.



Figure 48: Analog outputs connector terminology.

Connection:

Name	Size	Type	Description	Use
	16bits	Input	DAC user data	USER ports
	1bit	Output	Connection to the physical pins of the SoC.	HW ports

Connectors: J54, J66, J65 and J64.

6. Digital inputs (0-24V):

Description:

CarrierBoard provides a total of 32 isolated multiplexed digital input channels for the user. The IP updates the value of the digital inputs every 600µs.

How to use:

Ports denoted with “HW ports” must not be modified by the user. User can modify the connection of “USER ports”.

The data is delivered as an 8-bit frame. The mapping of the input number – signal – connector pin can be seen in the figure 49:

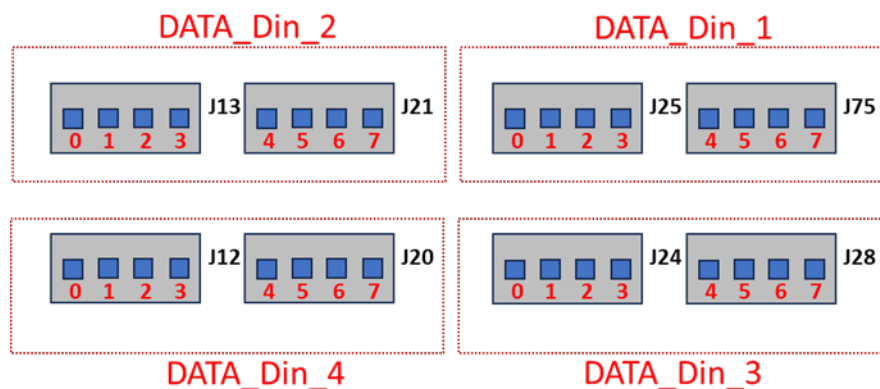


Figure 49: Digital inputs connector terminology.

In the data frame, a value of '1' means the input has 24V applied; a value of '0' means the input is grounded.

Connection:

Name	Size	Type	Description	Use
DATA_Din_1[7:0] DATA_Din_2[7:0] DATA_Din_3[7:0] DATA_Din_4[7:0]	8bits	Output	Digital data frame	USER ports
Din_SCL Din_SDA	1bit	I2C	Connection to the physical pins of the SoC.	HW ports

Connectors: : J12, J13, J20, J21, J24, J25, J28, J75.

7. Digital outputs (0-24V):

Description:

CarrierBoard provides a total of 32 isolated multiplexed digital inputs channels for the user to make use of. The IP updates the value of the digital inputs every 500µs.

How to use:

Use denoted with "HW ports" must not be modified by the user. User can modify the connection of "USER ports".

An 8-bit frame must be provided by the user to the IP to control the state of the digital output pins. In the data frame, a value of '1' means 24V will be applied; a

value of '0' means the it will be grounded.

The mapping of the input number – signal – connector pin can be seen in the figure 50:

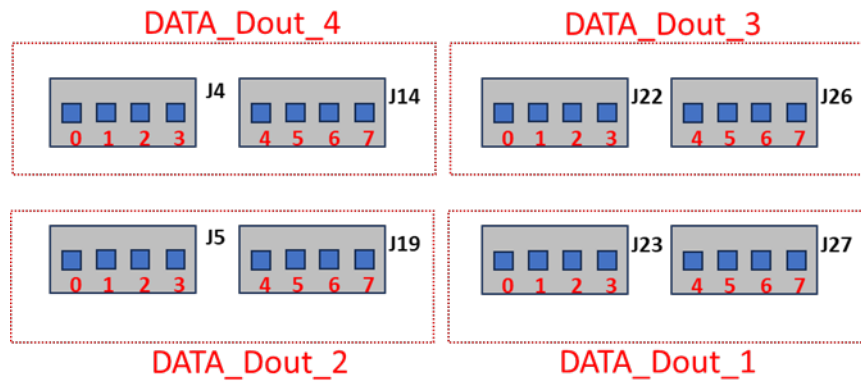


Figure 50: Digital outputs connector terminology (0-24V).

Connection:

Name	Size	Type	Description	Use
<ul style="list-style-type: none"> DATA_Dout_1[7:0] DATA_Dout_2[7:0] DATA_Dout_3[7:0] DATA_Dout_4[7:0] 	8bits	Input	Digital data frame	USER ports
<ul style="list-style-type: none"> Dout_SCL Dout_SDA 	1bit	I2C	Connection to the physical pins of the SoC.	HW ports

Connectors: J4, J5, J14, 19, J22, J23, J26, J27.

8. Digital outputs (0-3.3V):

Description:

CarrierBoard provides 14 digital outputs channels with voltage between 0-3.3V. The refresh time of the digital outputs is 500µs.

How to use:

Ports denoted with “HW ports” must not be modified by the user. User can modify the connection of “USER ports”.

An 8-bit frame must be provided by the user to the IP to control the state of the digital output pins. In the data frame, a value of '1' means 3.3V will be applied; a

value of '0' means the it will be grounded.

The mapping of the input number – signal – connector pin can be seen in the figure 51:

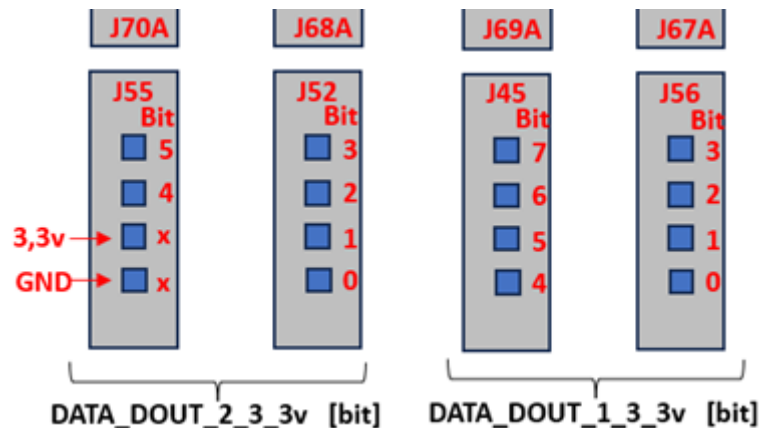


Figure 51: Digital outputs connector terminology (0-3.3V).

Connection:

Name	Size	Type	Description	Use
DATA_DOUT1_3_3v[7:0] DATA_DOUT2_3_3v[7:0]	8bits	Input	Digital data frame	USER ports
PWM_Ena_SCL_2 PWM_Ena_SDA_2	1bit	I2C	Connection to the physical pins of the SoC.	HW ports

Connectors: J45, J52, J55 and J56.

9. PWM channels:

Description:

CarrierBoard manages 36 PWM signals with their corresponding 18 error inputs signals and 18 enable outputs signals, it shall be noticed that each RJ45 is intended for a transistor branch or leg so it includes two PWM channels, and error and an enable signal.

There are 16 single RJ45 connectors and 2 double RJ45 connectors. The details of the signals are:

- 36 PWM signals: updated at the FPGA clock speed, in the template project at 10ns.

- 18 ERROR signals: NOT multiplexed and read at the FPGA clock cycle rate of 10ns.
- 18 enable signals: multiplexed and updated every 500µs.

General connection and pin out of a RJ45 PWM connector:

The pin out of the PWM RJ45 connector can be seen in the figure 52:

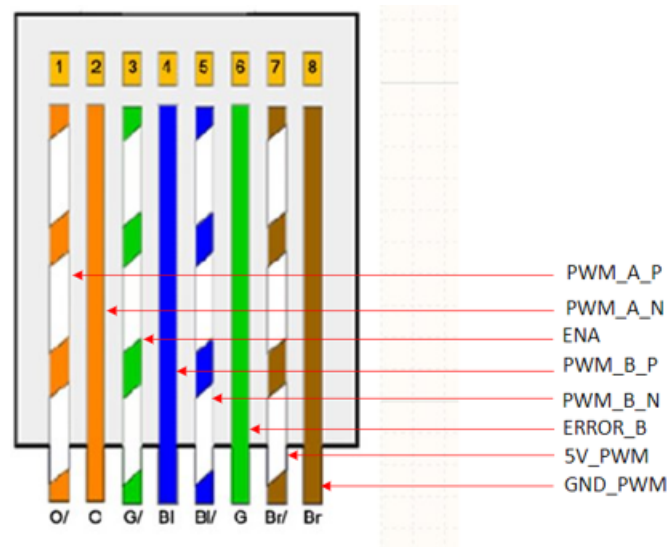


Figure 52: Pin out of the PWM RJ45 connector.

How to use PWM signals:

Format of the Driver IP input data of the PWM signals:

DATA_PWM_<ID>_J<XX>_<L>

<ID>: used to arrange the IP block interface to match the placement of the connectors on the Control Board.

J<XX>: Indicates the connector through which the signal is output. In the double connectors (J41 and J43) an A/B is added depending on whether it is the RJ-45. 'A' is for top connector and 'B' for bottom connector.

<L>: indicates the pair of wires (positive and negative) of the RJ-45 connector on which the signal is generated.

How to use enable signals:

There are three 8-bit data inputs, each bit representing the enable/reset signal of one of the connectors. The correspondence is shown in the next table:

DATA_PWM_00_ENA1		DATA_PWM_00_ENA2		DATA_PWM_00_ENA3	
BIT	Connector	BIT	Connector	BIT	Connector
0	J36	0	J43A	0	J32
1	J34	1	J43B	1	J30
2	J39	2	J41A		
3	J37	3	J41B		
4	J31	4	J35		
5	J29	5	J33		
6	J40	6	J44		
7	J38	7	J42		

How to use error signals:

The IP block provides a 32-bit data output (DATA_PWM_ERROR), the first 18 of these 32 bits contain the signal received by the connectors. The correspondence between the received bits and the connectors is shown in the next table:

BIT	ERROR SOURCE	BIT	ERROR SOURCE	BIT	ERROR SOURCE	BIT	ERROR SOURCE
0	J36	5	J29	10	J41_UP	15	J42
1	J34	6	J40	11	J41_bottom	16	J32
2	J39	7	J38	12	J35	17	J30
3	J37	8	J43_UP	13	J33		
4	J31	9	J43_Bottom	14	J44		

Connection:

Ports denoted with “HW ports” must not be modified by the user. User can modify the connection of “USER ports”.

Name	Size	Type	Description	Use
DATA_PWM_01_J30_A DATA_PWM_01_J30_B DATA_PWM_02_J32_A DATA_PWM_02_J32_B DATA_PWM_03_J42_A DATA_PWM_03_J42_B DATA_PWM_04_J44_A DATA_PWM_04_J44_B DATA_PWM_05_J33_A DATA_PWM_05_J33_B DATA_PWM_06_J35_A DATA_PWM_06_J35_B DATA_PWM_07_J41A_A DATA_PWM_07_J41A_B DATA_PWM_07_J41B_A DATA_PWM_07_J41B_B DATA_PWM_08_J43A_A DATA_PWM_08_J43A_B DATA_PWM_08_J43B_A DATA_PWM_08_J43B_B DATA_PWM_09_J38_A DATA_PWM_09_J38_B DATA_PWM_10_J40_A DATA_PWM_10_J40_B DATA_PWM_11_J29_A DATA_PWM_11_J29_B DATA_PWM_12_J31_A DATA_PWM_12_J31_B DATA_PWM_13_J37_A DATA_PWM_13_J37_B DATA_PWM_14_J39_A DATA_PWM_14_J39_B DATA_PWM_15_J34_A DATA_PWM_15_J34_B DATA_PWM_16_J36_A DATA_PWM_16_J36_B	1bit	Input	PWM signals inputs	USER ports
PWM0_2_J29 PWM10_2_J39 PWM11_2_J39 PWM12_2_J41 PWM13_2_J41 PWM14_2_J41 PWM15_2_J41 PWM16_2_J30 PWM17_2_J30 PWM18_2_J32 PWM19_2_J32 PWM1_2_J29 PWM20_2_J34 PWM21_2_J34 PWM22_2_J36 PWM23_2_J36 PWM24_2_J38 PWM25_2_J38 PWM26_2_J40 PWM27_2_J40 PWM28_2_J42 PWM29_2_J42 PWM2_2_J31 PWM30_2_J44 PWM31_2_J44 PWM32_2_J43 PWM33_2_J43 PWM34_2_J43 PWM35_2_J43 PWM3_2_J31 PWM4_2_J33 PWM5_2_J33 PWM6_2_J35 PWM7_2_J35 PWM8_2_J37 PWM9_2_J37	1bit	Output	PWM signals connection to the physical pins of the SoC.	HW ports

Name	Size	Type	Description	Use
DATA_PWM_ERROR[31:0]	1bit	Output	ERROR input signals	USER ports
<div> <div>ERR0_2_J29[0:0]</div> <div>ERR10_2_J34[0:0]</div> <div>ERR11_2_J36[0:0]</div> <div>ERR12_2_J38[0:0]</div> <div>ERR13_2_J40[0:0]</div> <div>ERR14_2_J42[0:0]</div> <div>ERR15_2_J44[0:0]</div> <div>ERR16_2_J43[0:0]</div> <div>ERR17_2_J43[0:0]</div> <div>ERR1_2_J31[0:0]</div> <div>ERR2_2_J33[0:0]</div> <div>ERR3_2_J35[0:0]</div> <div>ERR4_2_J37[0:0]</div> <div>ERR5_2_J39[0:0]</div> <div>ERR6_2_J41[0:0]</div> <div>ERR7_2_J41[0:0]</div> <div>ERR8_2_J30[0:0]</div> <div>ERR9_2_J32[0:0]</div> </div>	1bit	Input	ERROR signals connection to the physical pins of the SoC.	HW ports
<div> <div>DATA_PWM_00_ENA1[7:0]</div> <div>DATA_PWM_00_ENA2[7:0]</div> <div>DATA_PWM_00_ENA3[7:0]</div> </div>	8bits	Input	Enable input signal	USER ports
<div> <div>PWM_Ena_SCL_2</div> <div>PWM_Ena_SDA_2</div> </div>	1bit	Input	Enable signal connection to the physical pins of the SoC.	HW ports

Connectors: J29, J30, J31, J32, J33, J34, J35, J36, J37, J38, J39, J40, J41, J42, J43 and J44.

10. Quick analog input signals:

Description:

CarrierBoard provides a total of 12 quick-analog-input 12bits channels. This IP manages the communication with the ADCs. The sample refresh period 600ns (1.66MSPS).

How to use:

Ports denoted with “HW ports” must not be modified by the user. User can modify the connection of “USER ports”.

Driver IP is continuously reading the 12 bits ADCs output and making it accessible to the user. The following formula can be used to convert the read data to volts:

$$V_{\text{read}} = \frac{V_{\text{ref}} \cdot N_{\text{read}}}{2^{12} - 1} \quad (3)$$

$$V_{\text{ref}} = 5V$$

$$N_{\text{read}} = \text{Driver IP output data}$$

The ADCs employed have 4 channels per chip and there is a total of 3 chips. Each of those chips is identified by a letter in the channel nomenclature (‘A’, ‘B’ or ‘C’).

Connection:

The nomenclature used by the user in ADC data signals is:

DATA_ADC_quick_<n>_<ID>

<n> is a number used to order the signals in the IP interface.

<ID> identifies to which connector each data output belongs, providing all the information needed by the user for its use.

The nomenclature used for the ADC physical signals is:

qS <n>_<m>_<ID>

<n>_<m> are numbers that simply identify the signals, so that they can be easily traced using the board schematic.

<ID> identifies the connector to be read (source of the input data).

Name	Size	Type	Description	Use
DATA_ADC_quick_00_J71A[31:0] DATA_ADC_quick_01_J73A[31:0] DATA_ADC_quick_02_J72A[31:0] DATA_ADC_quick_03_J74A[31:0] DATA_ADC_quick_04_J71B[31:0] DATA_ADC_quick_05_J73B[31:0] DATA_ADC_quick_06_J72B[31:0] DATA_ADC_quick_07_J74B[31:0] DATA_ADC_quick_08_J71C[31:0] DATA_ADC_quick_09_J73C[31:0] DATA_ADC_quick_10_J72C[31:0] DATA_ADC_quick_11_J74C[31:0]	32bits	Output	Data read by the quick ADC	USER ports
qSCLK_0 qSCLK_1 qSCLK_2 qnCS_0 qnCS_1 qnCS_2	1bit	Output	Connection to the physical pins of the SoC.	HW ports
qS1_0_J71A qS1_1_J71B qS1_2_J71C qS2_0_J73A qS2_1_J73B qS2_2_J73C qS3_0_J72A qS3_1_J72B qS3_2_J72C qS4_0_J74A qS4_1_J74B qS4_2_J74C	1bit	Input	Connection to the physical pins of the SoC.	HW ports

Connectors: J71A, J72A, J73A, J74A, J71B, J72B, J73B, J74B, J71C, J72C, J73C and J74C.

11. Slow analog input signals:

Description:

The board provides a total of 12 slow analog inputs 12bit channels. Driver manages the communication with the ADCs. The sample refresh period 10.56μs (93.8 kSPS).

How to use:

Ports denoted with “HW ports” must not be modified by the user. User can modify the connection of “USER ports”.

Driver IP is continuously reading the 12 bits ADCs output and making it accessible to the user. The following formula can be used to convert the read data to volts:

$$V_{\text{read}} = \frac{V_{\text{ref}} \cdot N_{\text{read}}}{2^{12} - 1} \quad (4)$$

$$V_{\text{ref}} = 5V$$

$$N_{\text{read}} = \text{Driver IP output data}$$

The ADCs have 4 channels per chip and there is a total of 3 chips. Each of those chips is identified by a letter in the channel nomenclature ('A', 'B' or 'C').

Connection:

Name	Size	Type	Description	Use
DATA_ADC_slow_00_067A[31:0] DATA_ADC_slow_01_069A[31:0] DATA_ADC_slow_02_068A[31:0] DATA_ADC_slow_03_070A[31:0] DATA_ADC_slow_04_067B[31:0] DATA_ADC_slow_05_069B[31:0] DATA_ADC_slow_06_068B[31:0] DATA_ADC_slow_07_070B[31:0] DATA_ADC_slow_08_067C[31:0] DATA_ADC_slow_09_069C[31:0] DATA_ADC_slow_10_068C[31:0] DATA_ADC_slow_11_070C[31:0]	32bits	Output	Data read by the quick ADC	USER ports
sDIN0 sDIN1 sDIN2 sSCLK0 sSCLK1 sSCLK2 snCS0 snCS1 snCS2	1bit	Output	Connection to the physical pins of the SoC.	HW ports
sDOUT0 sDOUT1 sDOUT2	1bit	Input	Connection to the physical pins of the SoC.	HW ports

Connectors: J67A, J68A, J69A, J70A, J67B, J68B, J69B, J70B, J67C, J68C, J69C and J70C.

6.3.2 IP AXI_PSCoscilloscope

Introduction

The AXI_PSCoscilloscope IP is a PSC product designed to provide full access to FPGA internal variables inside the SoC. Its controls and capabilities are the same as the ones typically found in desktop devices.

This product requires four parts to operate:

1. An oscilloscope IP that is added to the project and connected to the signals to be measured. It manages the data acquisition and trigger.
2. A microprocessor file (PSC_PSCoscilloscope.elf) added to the Xilinx Vitis project. It manages the Ethernet communication and Windows user interface.
3. A file that contains the oscilloscope configuration (PSCoscilloscope_conf.txt). This file can be modified by the user to update the IP and must be copied inside the microSD card inserted in ControlBoard.
4. A Windows-based software that corresponds to the user interface.

Features and interface:

AXI_PSCoscilloscope IP block interface can be divided into six parts according to its functionality.

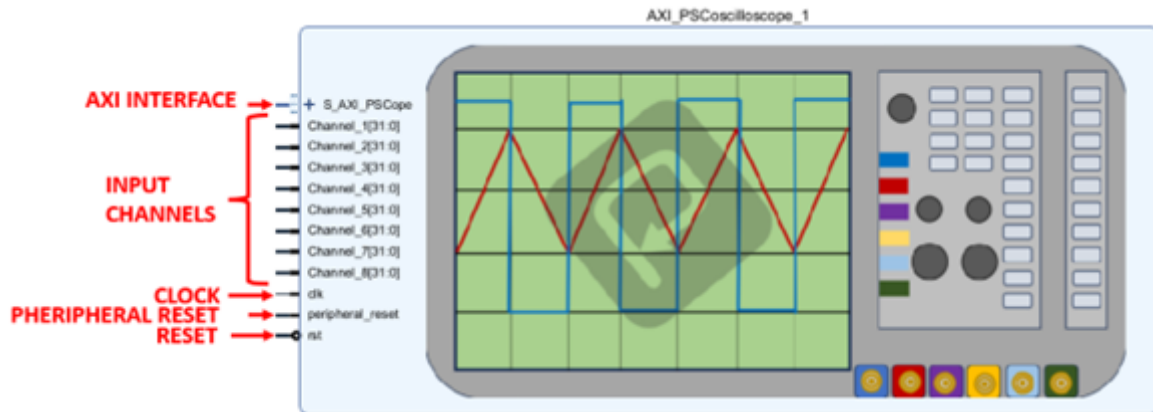


Figure 53: AXI_PSCoscilloscope IP and its functional parts.

AXI interface:

Connects the AXI_PSCoscilloscope with the microprocessor that controls its operation and manages the communication via ethernet. In SmartRCP_Template provided by PSC this connection has already been made.

Input channels:

AXI_PSCoscilloscope can simultaneously measure 8 channels of 32 bits, these can be multiplexed (by using external logic) if it is necessary to measure more signals.

Input channels must be type float numbers. In case the user needs to measure other type of data conversions can be made in VHDL block or in HLS (we recommend using HLS).

Clock:

This signal is automatically connected when the block is added. In SmartRCP_Template provided by PSC this connection has already been made.

Peripheral reset and reset:

It is necessary to connect the reset signal and peripheral reset signal. In SmartRCP_Template provided by PSC this connection has already been made.

Capabilities:

- The minimum time between samples is 50ns.

- Window size = 20k samples

AXI_PSCoscilloscope transmits packages of 20k samples per channel. Example 1: With a resolution of 50ns a total of 1ms would be transmitted. Example 2: With a resolution of 0.5us a total 10ms would be transmitted.

- Trigger. There are two trigger modes: auto mode and channel mode.

Auto mode: the oscilloscope is synchronized with an internal signal whose frequency (period) and phase can be modified by the user in the Windows application.

Channel mode: A channel and a reference voltage value are selected, when the signal cuts with this value, the data capture will start.

How to use and implement the AXI_Oscilloscope in a project

If the user is using SmartRCP_Template project provided by PSC skip steps 1, 2 and 3.

Please, follow the following steps:

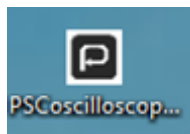
1. Add the IP to the Vivado project repository.
2. Add the IP to the diagram (graphical interface) and accept the option to make the connections automatically.
3. Manually connect the peripheral_reset signal.
4. Connect the signals to be measured to the oscilloscope probes (channels). These signals must be floating types.
5. Once the design is finished in Vivado and exported to Xilinx Vitis, there are two possibilities: using the debug mode and loading the design (HW and SW) using the USB or use a microSD. In the first case it will be necessary to add the PSC_Oscilloscope.elf file to the debugger on the cortex r5_0 microprocessor. The procedure for doing this is covered in the Vitis section of this manual as well as the procedure for creating the BOOT.BIN file that is added to the microSD card.
6. Save the oscilloscope configuration file (PSCoscilloscope_conf.txt) on the microSD card. This file contains the information shown in the next figure.
PSC_ETH_AUTO_CON: If 'Y' is selected, the router will automatically assign an IP address to the oscilloscope, if 'N' is selected, the IP address of the oscilloscope will be assigned manually.
PSCope_client_status: If 'N' is selected, the oscilloscope will communicate only with the client whose address has been defined, if 'Y' is selected, the oscilloscope will communicate with the first user who makes a configuration request.

```
<>PSCoscilloscope==>
PSC_ETH_AUTO_con="Y"      {Y/N}
PSCope_IP="192.168.1.78"
PSCope_DEFAULT_IP_MASK="255.255.255.0"
PSCope_DEFAULT_GW_ADDRESS="192.168.1.1"
PSCope_cliente_status="Y"  {Y/N}
PSCope_client_IP="192.168.1.22"
```

Figure 54: Contents of PSCoscilloscope_conf.txt file.

7. If IP address is, it can be read as it is sent by microUSB cable at power-up.

8. Start the PSCoscilloscope application interface:



PSCoscilloscope application interface

The next figure provides an overview of the PSCoscilloscope application. It provides two different graphs in which internal variables can be plotted and several controls. See next figure for more detail.

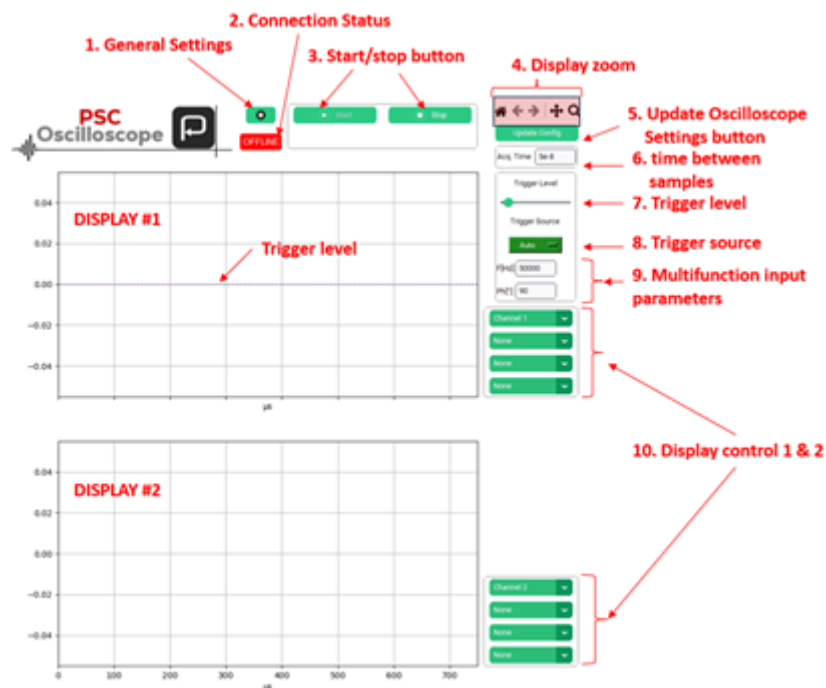


Figure 55: PSCoscilloscope application.

Important note:

Any modification done in the SW requires the press of button Update Config, otherwise the change will not be applied.

General Settings:

Allows the definition of:

- IP address. Use the same address defined in PSCoscilloscope_conf.txt file.
- Clock period. This value must be 10ns.

It is necessary to press “update configuration” button for the changes to take effect.

Connection Status:

If it is red with the message "offline" it indicates that it is not communicating. This situation may be due to an incorrect NET configuration or even that SmartRCP is not switched-on.

Start/Stop button:

The start button enables data acquisition. The stop button stops data acquisition. To be able to analyze or zoom in on the data Stop button must be pressed, to resume data acquisition just press the Start button.

Time between samples:

PSCoscilloscope application always represents 20M samples, this field represents the separation between the samples. The minimum value is 50ns.

Trigger level, source and additional parameters:

The trigger level is the voltage value that initiates data acquisition. The trigger source defines the channel that will be applied to trigger settings. It can be AUTO or a particular channel.

If AUTO is selected two extra fields will appear to define the frequency of the internal trigger signal and its phase, in the case a channel is selected as trigger, two fields will appear asking for the trigger value range.

Display control:

Each graph can represent up to three signals at the same time, these fields allow to select the viewing signals.

6.3.3 Integrated Logic Analyzer (ILA)

Integrated Logic Analyzer (ILA) IP core is a logic analyzer core provided by Xilinx that can be used to monitor the internal signals of a design. Because ILA core is synchronous to

the design being monitored, all design clock constraints that are applied to the design are also applied to the components inside the ILA core. Key Features and benefits are:

- User-selectable trigger width, data width, and data depth.
- Multiple probe ports, which can be combined into a single trigger condition.
- AXI Interface on ILA IP core to debug AXI IP cores in a system.

The use, configuration and definitions of the ILA IP can be found covered in depth in Xilinx documentation. SmartRCP does not limit in any aspect the use of this tool, from PSC we recommend the use of this tool to debug digital signals which are used in state machines or as alarms.

6.4 Xilinx Vitis software tools

6.4.1 Introduction

Vitis is the tool provided by Xilinx for programming the microprocessors incorporated in their SoC devices. This section provides a brief introduction to the basic concepts and steps required to program SmartRCP system.

6.4.2 Project creation in Vitis

1. It is necessary to have created and exported the bitstream in the Vivado project.
2. Launch Vitis.
3. Select the "Create Application Project" option and click next.

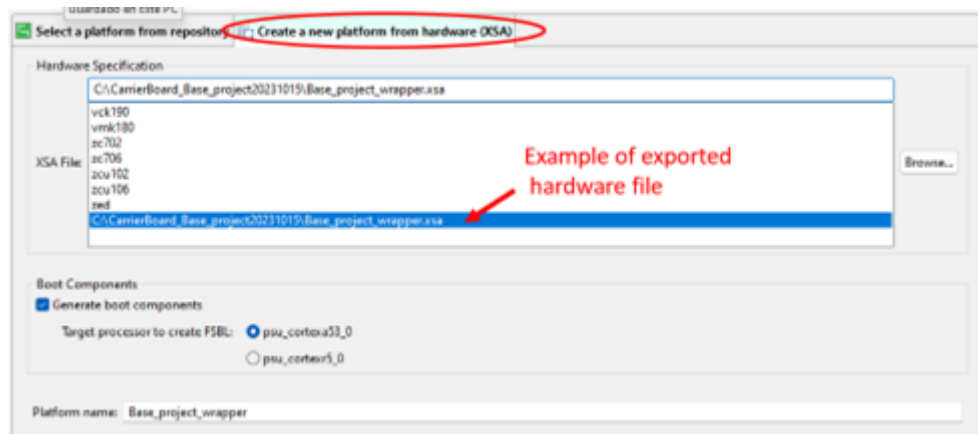


Figure 56: Vitis architecture definition.

4. In the "Platform" window, go to the "Create a new platform from hardware (XSA)" option and select the .XSA file that was exported in the Vivado project. Click next.
5. In the "Application Project Details" window user can choose the name of the application to be created, the name of the system (including all microprocessors and applications) and which of the 6 microprocessors is used for the application. Cortex-R5 use should be avoided in case PSCoscilloscope is used as it already uses those

microcontrollers. In this example, a Cortex A53 processor has been used and the following names employed:

The Application Project: APP_example.

The System project: APP_example_userGuide.

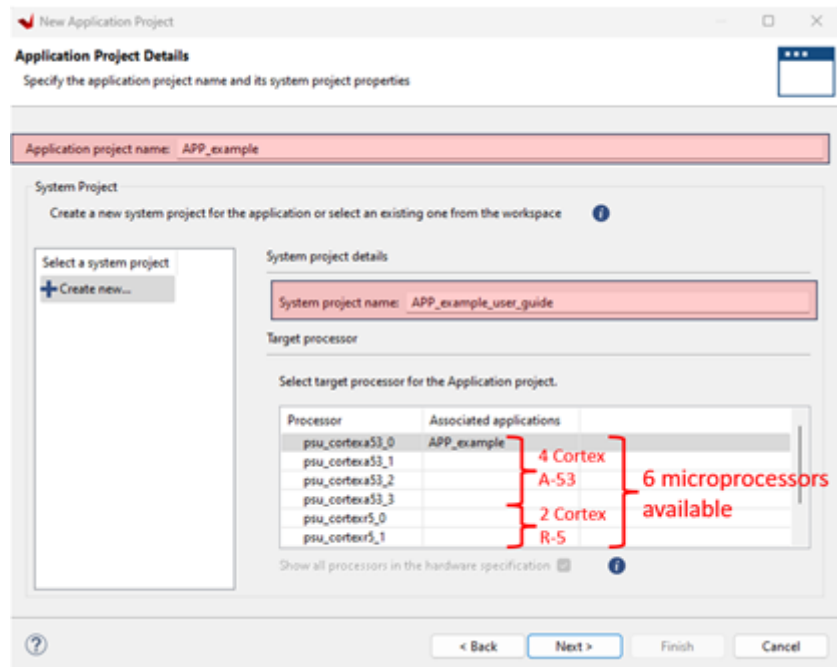


Figure 57: Vitis application names and resources.

6. In the "Domain" screen the user can choose between using a Freertos10_xilinx operating system or using a standalone design. For this example, standalone is selected. Click next.

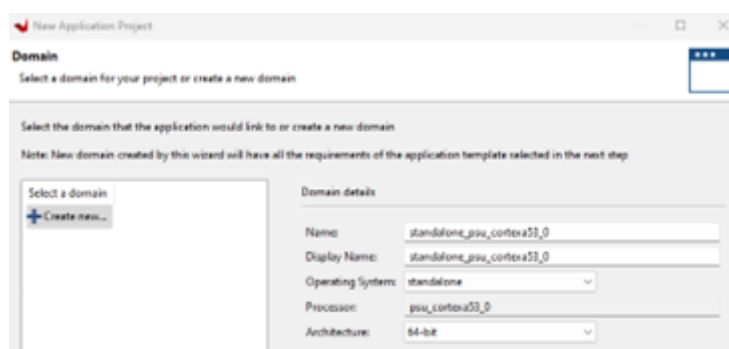


Figure 58: Vitis OS or standalone mode selection.

7. In the last step, the user can select if a template like "Hello world" is used or a blank project is created. Press the finish button.

6.4.3 Procedure for loading and debugging the design on SmartRCP

This subsection shows how to generate a boot.bin file so SmartRCP system will load the design from the microSD card and how to use the microUSB cable to load the design from the computer and to debug it. Using the microUSB method is intended only for debugging, it is the microSD card that provides space for long-term program storage.

MicroUSB program loading and debugging:

The next steps must be followed:

1. Click on "Build the active configuration of the selected project".
2. The default configuration of CarrierBoard is booting directly from microSD card. To enable microUSB debugging, some registers need to be modified. To do so, open the XSCT Console and type the commands below:

```
connect
targets -set -nocase -filter name =~*PSU*~
mwr 0xff5e0200 0x0100°
rst -system
```

The board has switched to microUSB boot mode. This mode will be held until CarrierBoard is switched-off; in the following start-up it will be configured automatically to boot from microSD card.

3. There are two possibilities for loading the design, using DEBUG or RUN. The first time, the user must click in the arrow and select "Single Application Debug".
4. Click on RUN/DEBUG icon depending on the user's needs.

microSD card loading:

It must be noted that this boot mode is intended for a mature state development as it does not allow debugging. The user must follow the steps:

1. Click on the System project and select the "Create Boot Image" option.

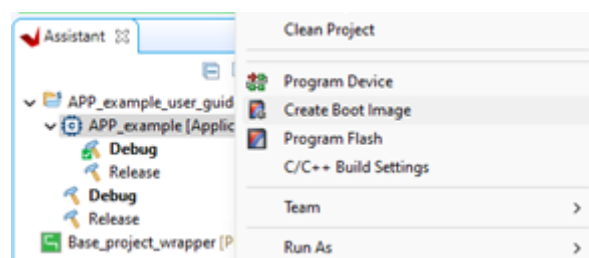


Figure 59: Creating boot image in Vitis.

2. A configuration window in which the path of all the needed and generated files is shown. Click on "Create Image".

3. Copy the Boot.bin file generated into the micro-SD card (The micro-SD card should be formatted as FAT32).
4. Press CarrierBoard reset button.

6.4.4 How to make multi-processor designs

This section provides some basic information about the procedure to add several micro-processors to the design. To do so, follow the steps.

1. Open the System Project Settings and click on add "Application Project".

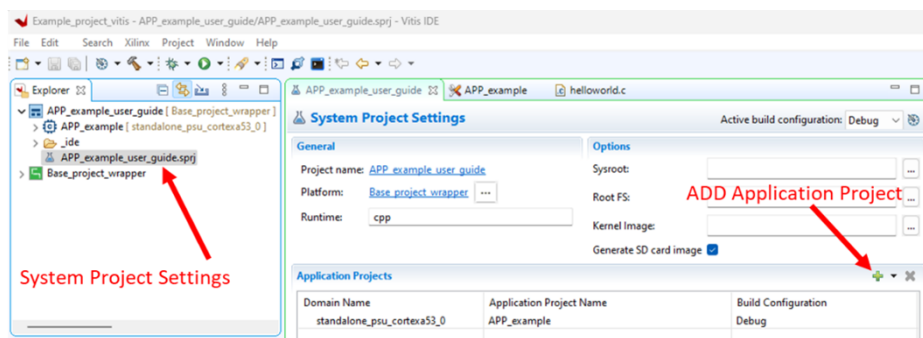


Figure 60: Adding additional application project in Vitis.

2. Check the "Show all processors in the hardware specification" option.
3. Select the processor to be used and follow the steps of the project creation procedure already covered in the manual.
4. The new processor will appear in the System Project, in this example it is APP_UP_2.

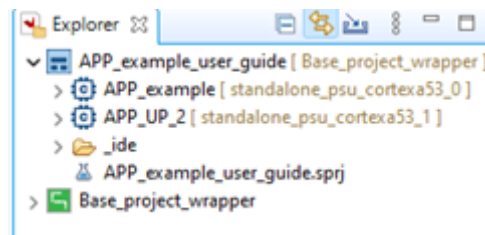


Figure 61: New processor added to the design in Vitis.

NOTE. The procedure for creating the BOOT.BIN file does not vary regardless of whether one or more microprocessors are used.

5. Go to "Application" and select the new microprocessor.

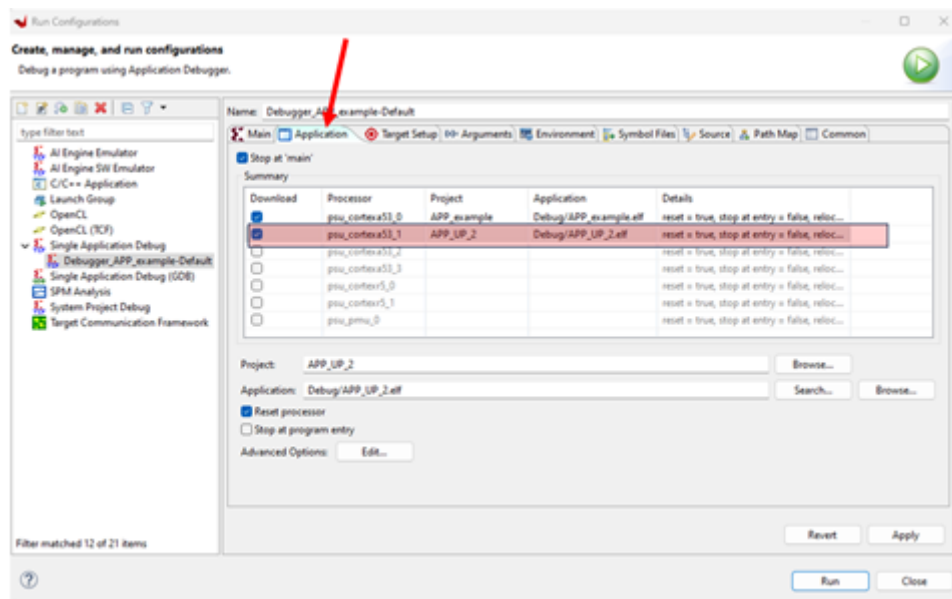


Figure 62: Selecting new processor in Vitis.

If several microprocessors are working in parallel, the user must be cautious with resource sharing. In the case of memory, the user must assign a specific memory space to each microprocessor. This is done in the Linker Script located in the /src folder of each microprocessor. The parameters to consider are marked in the figure 63.

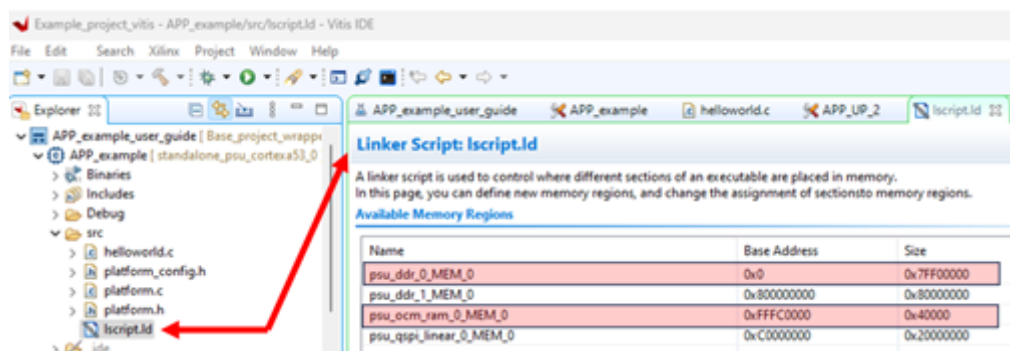


Figure 63: Linker memory positions in Vitis.

The available DDR memory segments and its type is shown in the figure 64:

Address Range	Zynq UltraScale+ MPSoC
0000_0000 to 0000_FFFF	DDR
0001_0000 to 0002_FFFD	DDR
0002_FFFE to 0003_FFFF	DDR
0004_0000 to 0005_FFFF	DDR
0006_0000 to 0007_FFFF	DDR
0008_0000 to 000B_FFFF	DDR
000C_0000 to 000F_FFFF	DDR
0010_0000 to 3FFF_FFFF	DDR
4000_0000 to 7FFF_FFFF	DDR
FFFC_0000 to FFCF_FFFF	OCM

Figure 64: DDR memory available in KRIA K26.

If the user project includes AXI_PSCoscilloscope, it will be using Cortex-R5_0 processor and the following memory segments:

DDR = > Base Address = 0x40000000 | Size = 0x7FE00000

OCM => Base Address = 0xFFFC0000 | Size = 0x4000

7 SmartRCP Test Mode

SmartRCP systems include by default a program that provides the user the capability to test all HW chains of CarrierBoard. In case of any HW issue suspicion, this program can be used.

If any HW issue is detected, please contact PSC by email to support address and be sure to include the QR info placed on the bottom plastic cover. Further contact data is provided in this document in the appropriate section.

For the use of this test interface, it is necessary to know the COM port identifier of the board, so connect a computer with a microUSB-USB cable (type B) to connector J6 of SmartRCP and check the COM port associated. Insert the COM value in the Windows RCP_Test_Tool application by clicking on configuration button. Power up the board and press the “start” button.

All of the tests in RCP_Test_Tool application include a question mark button which provides a set of instructions for manual validation of the HW chain of CarrierBoard.

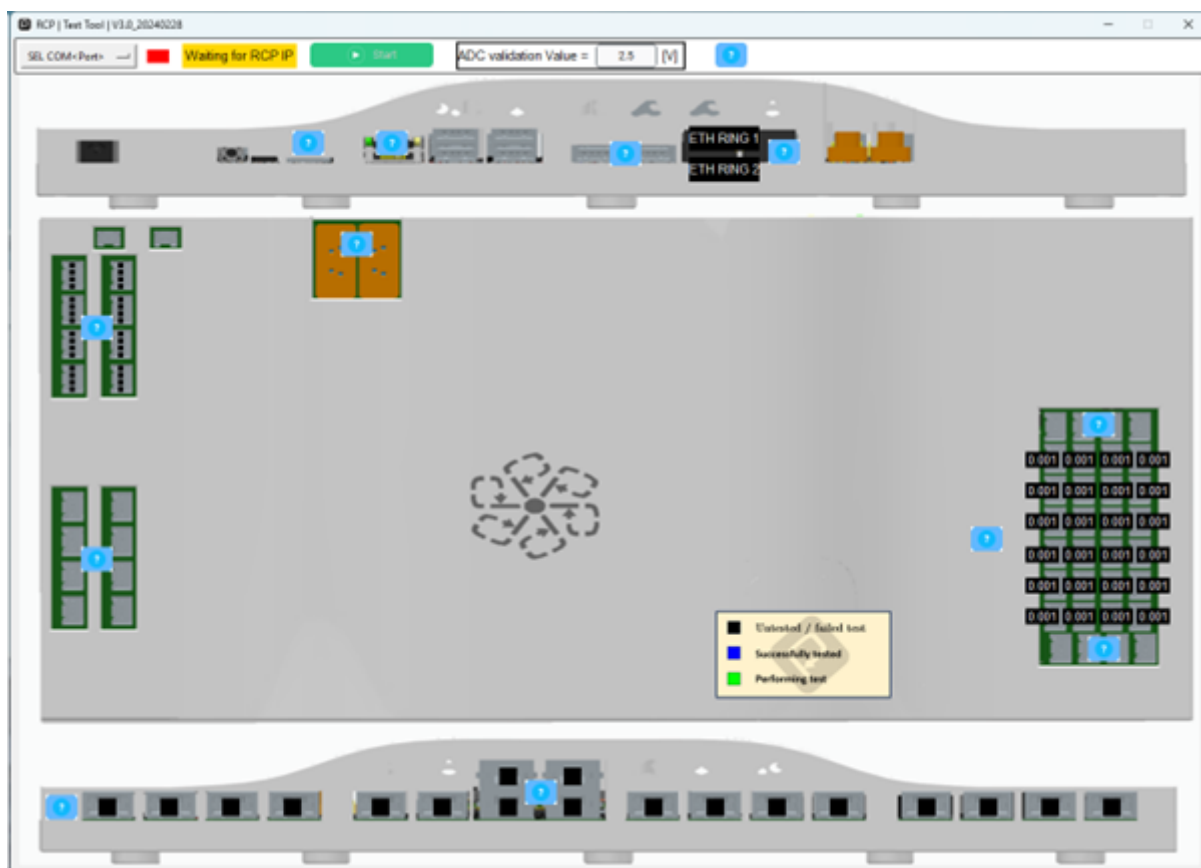


Figure 65: RCP_Test_Tool Windows application main window.

All test details are provided inside RCP_Test_Tool application so no extra details are provided in this chapter.

8 Synchronous Buck Converter demonstrator

8.1 Hardware

To explore the basic features of the RCP system and to demonstrate its immense capabilities, a tutorial has been created with a small 5W synchronous buck converter prototype. This converter is based on the following voltage step-down topology:

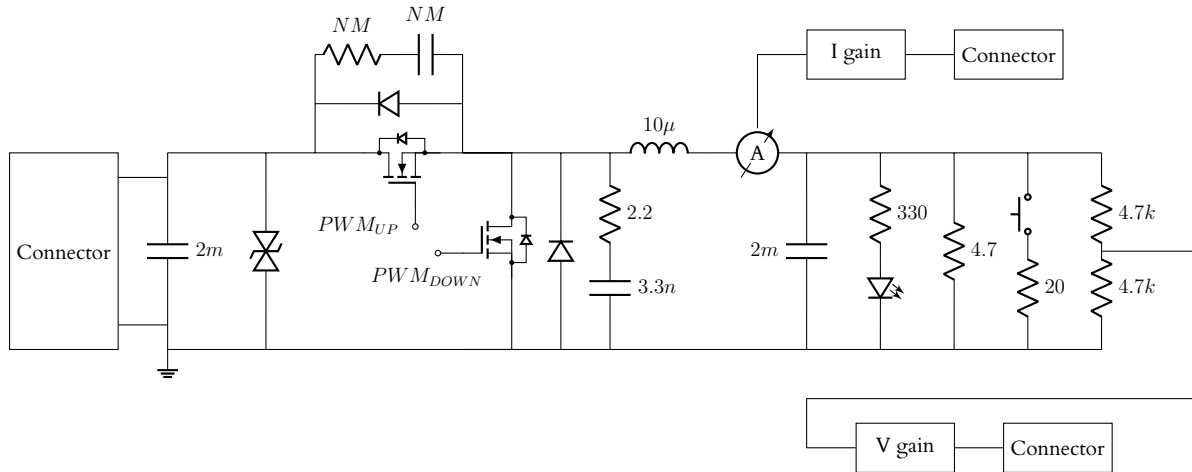


Figure 66: Buck Converter topology included in RCPBuckBoard

In the Figure the main components of the topology are shown:

- Two power MOSFETs with diode and snubber
- Output LC filter
- Output voltage sensing
- Test points included with its silkscreen labels

The power ratings of the converter are:

- Input: 5Vdc via USB wire (A standard 5V 2A phone charger can suit the input perfectly)
- Output: 0.5V to 4.5V regulable via duty cycle
- Load: fixed resistor: 5W 4.7Ω.

Experienced users can easily modify the converter to suit different voltages, power levels, measuring points, switching techniques, control strategies, load, etc. However, these modifications will not be covered by the product warranty.

The next Figure provides a detailed view of the board and the location of the main parts and sub-systems:

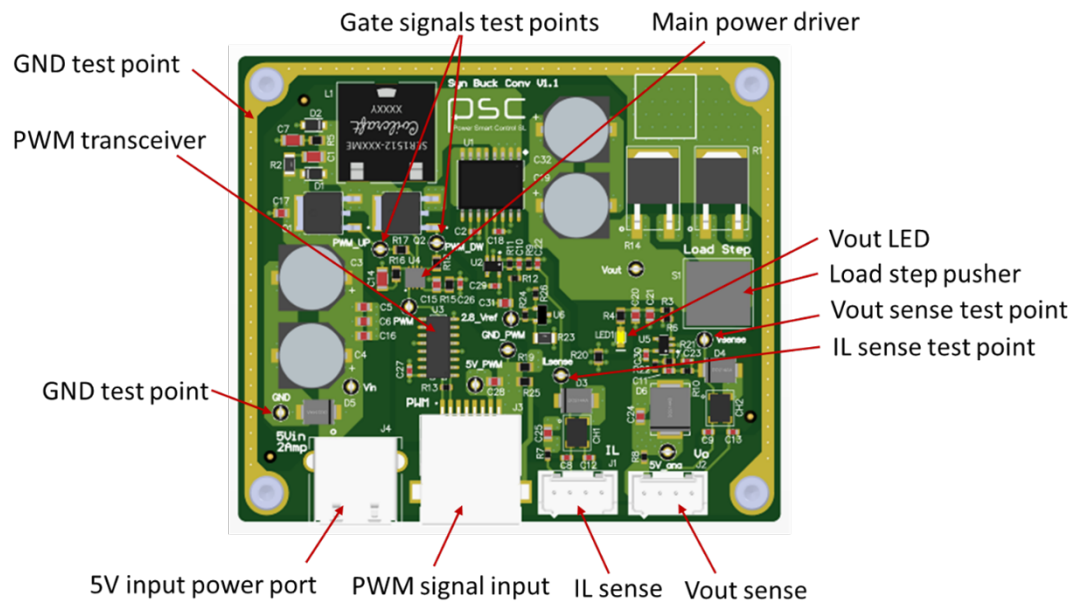


Figure 67: RCPBuckBoard.

Next Figure provides an overview of the connection between RCPBuckBoard and SmartRCP. There are multiple connection possibilities thanks to the high versatility of the SmartRCP system which will lead to the same solution, in this example one particular configuration has been chosen.

All connecting wires have been included.

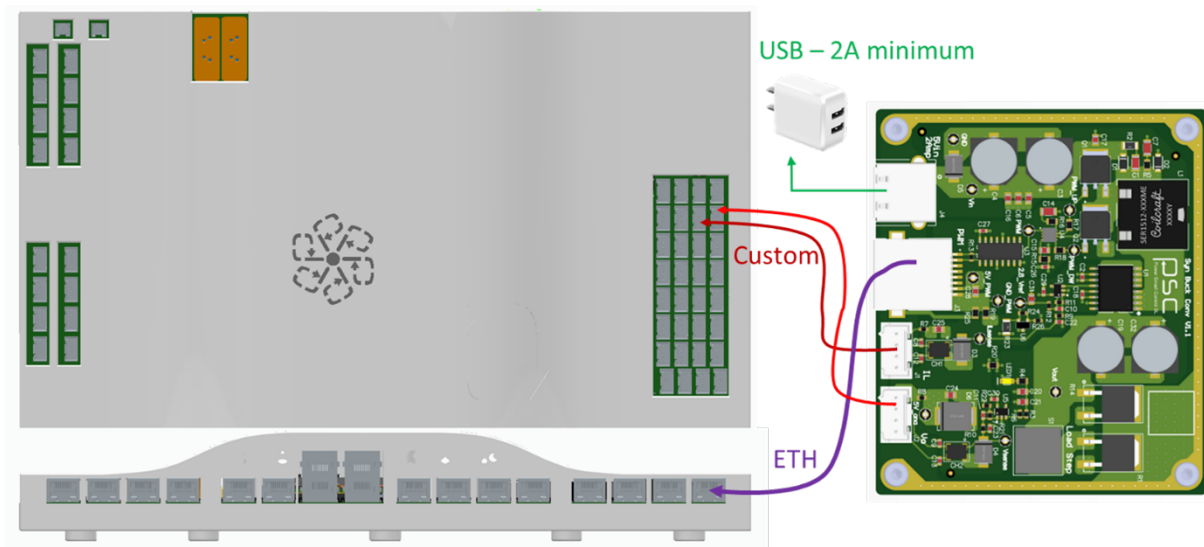


Figure 68: Connection among RCPBuckBoard and SmartRCP.

Next figure provides the full schematic of the Sync Buck demo board to provide the user full understanding of the components and connections made on it.

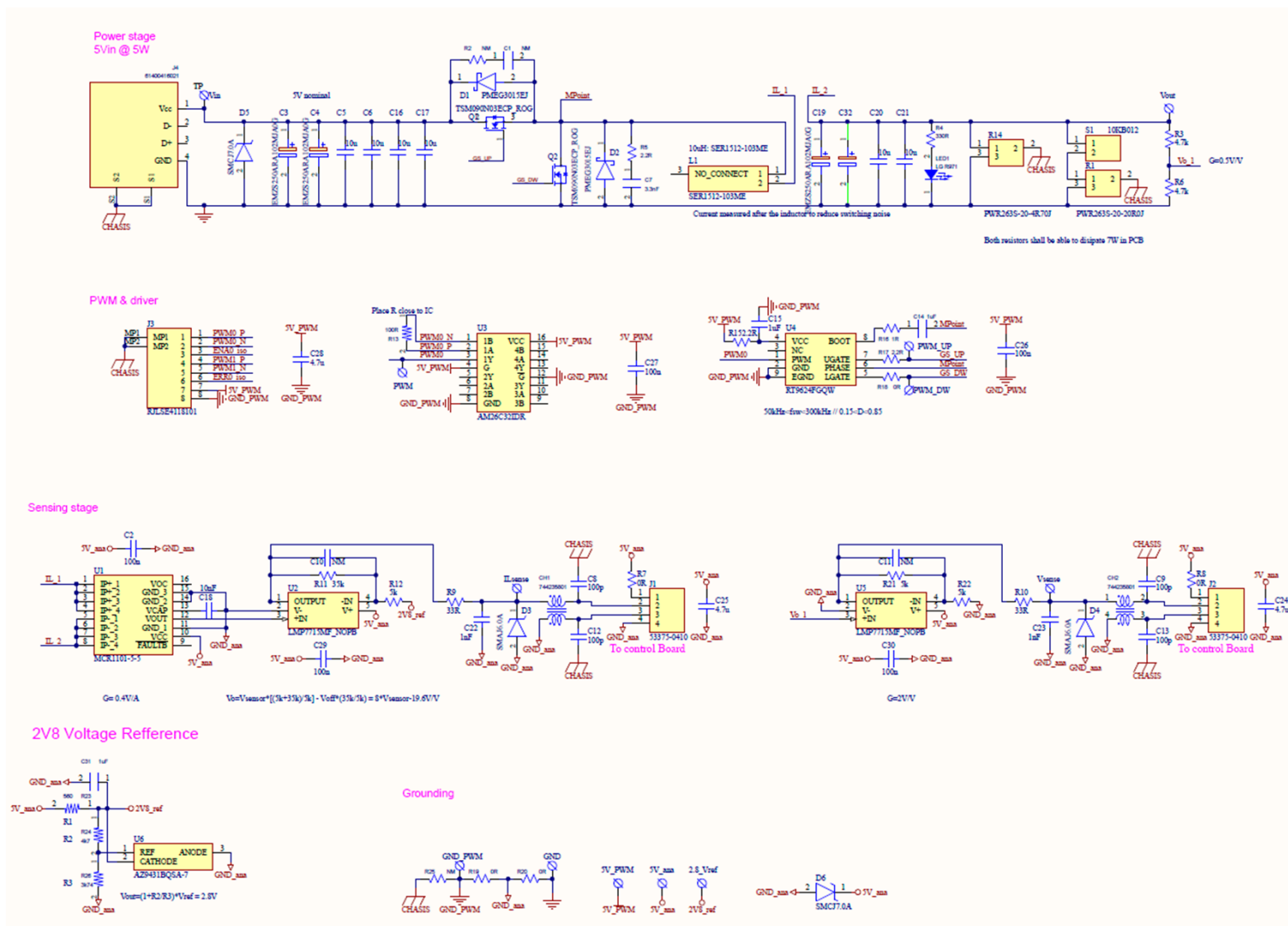


Figure 69: RCPBuckBoard schematic.

Next figure provides a connection overview between an external oscilloscope and RCP-BuckBoard. With this configuration, the PWM signal, output voltage and sensing voltage can be measured. These measures can be also taken inside SmartRCP system thanks to PSCoscilloscope.

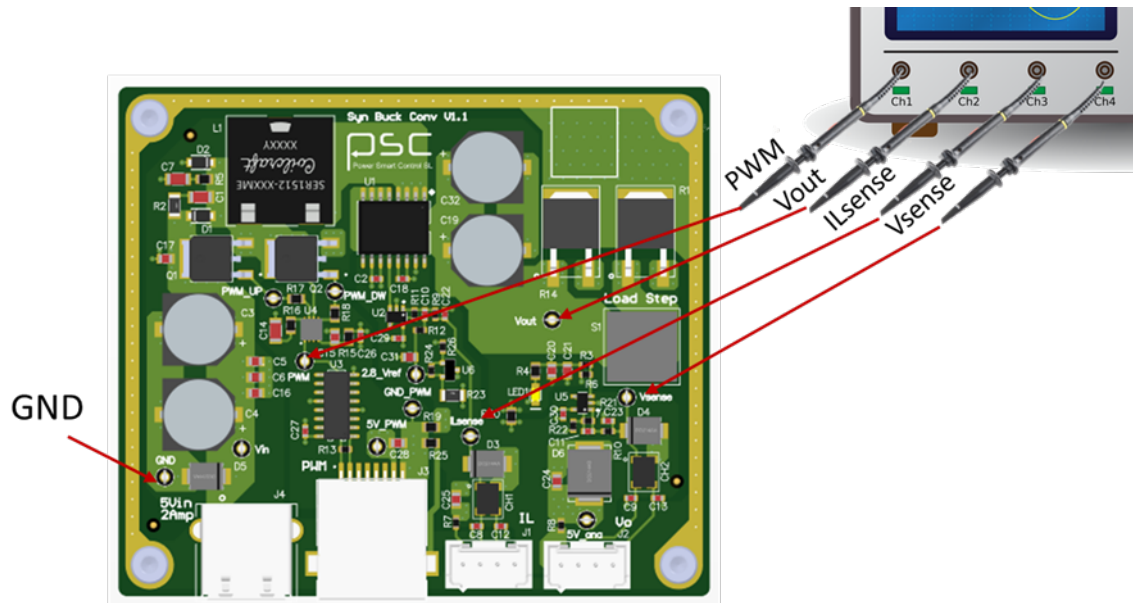


Figure 70: RCPBuckBoard connection to a desktop oscilloscope.

Next figure provides the connections to measure the PWM pulses of both transistors. With this configuration dead times, switching instants, etc. can be perfectly seen and measured. It shall be noted that both signals are not referred to the same point so two isolated voltage probes are required or a short-circuit will be performed among grounds.

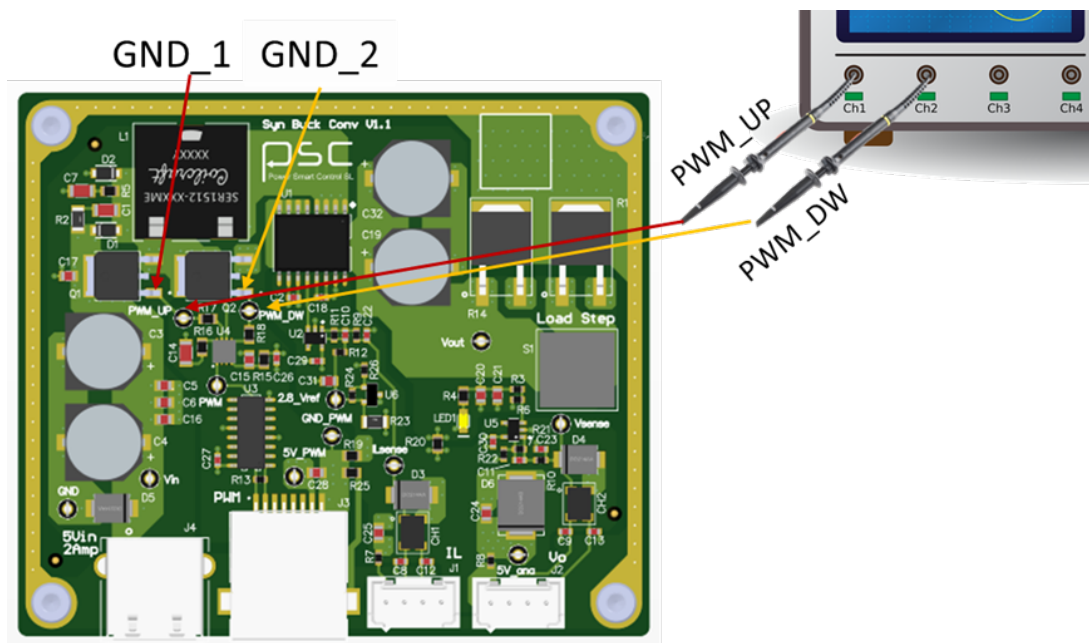


Figure 71: RCPBuckBoard connection to a desktop oscilloscope.

The driver used in this board is RT9624FGQW which is a sync buck converter driver capable of switching up to 500kHz. This device automatically sets the dead times to ensure that both devices are not ON at the same time, so it requires a single PWM signal from SmartRCP.

8.2 SmartRCP configuring

8.2.1 Introduction

This section provides a detailed example of how to adapt SmartRCP_Template to control the buck converter inside RCPBuckBoard by performing a closed voltage loop. The following steps must be followed:

1. High-level block design using Vitis HLS. Implementation of complex functionalities such as Moving Average, PID Compensator, among others.
2. Modification of SmartRCP_Template using Vivado.
3. Connection of PSCoscilloscope.
4. Implementation of a simple console on one of the microprocessors available to facilitate communication between the user, the microprocessor and the FPGA.
5. Inclusion of the necessary software for PSCoscilloscope.

The next figure shows the block diagram of this new project where the “CHi” labels correspond to the channel number of the embedded oscilloscope.

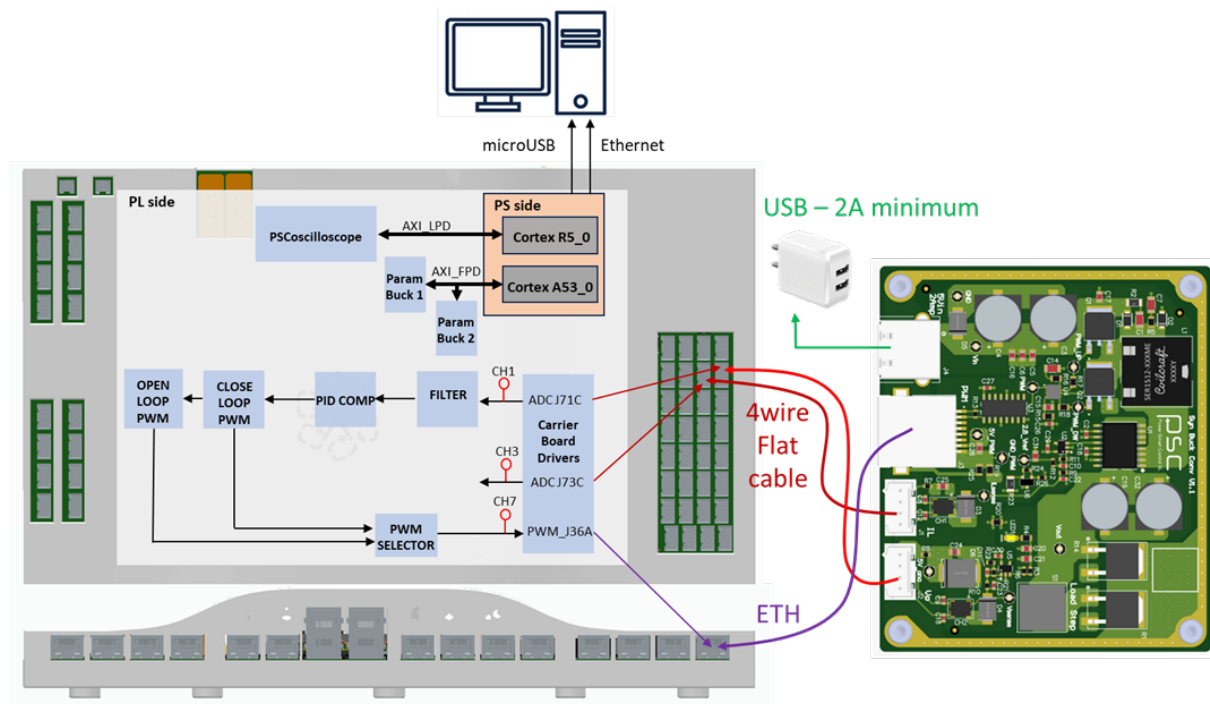


Figure 72: RCPBuckBoard project. Main block diagram and connection.

The following image provides a general view of the control loop of the buck converter, this very same control scheme can be seen in the previous figure (72).

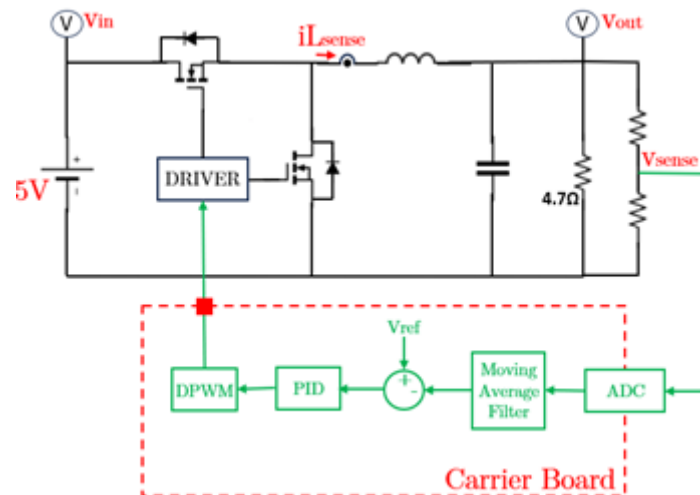


Figure 73: Main block diagram of the voltage control loop.

8.2.2 Creating C-code based IP cores using Vitis HLS

Creating IP cores based on High-Level Synthesis allows the user to start with C-code and automatically translate it to RTL code. To do so, the user has to use Vitis HLS tool. This section provides a full example of how to use Vitis HLS to create an IP core and include it in the Vivado SmartRCP_Template.

Moving average filter:

This IP block performs a low pass filteraction to analog input signals, such as output voltage feedback signals, to reduce the ripple and the measuring noise naturally present in analog measures. It must be noted that the effect of this filtering is unneeded in the case of the RCPBuckBoard because its LC filter cross frequency is way below the switching frequency of the converter so no ripple issues shall appear.

This averaging or smoothing action however is necessary in noisy environments or converters with a smaller filtering action.

With this IP the whole process of using Vitis HLS is covered, this process is not repeated with the following IPs as the user only needs to repeat the steps and indications provided changing the source files.

The description of the required functionalities is expressed by the C++ code adapted to Vitis HLS. See the figure 74.

```

#include "ap_int.h"
using namespace std;

void mov_ave_fil (ap_uint<12> data_in, float scale_bits, ap_uint<12> *data_out){

    static ap_uint<12> x_i, xr[4] = {0};
    static ap_uint<4> i = 0;
    const ap_uint<4> N = 4;
    const float factor = 0.25;
    float xx;
    static float data_out_prev;
    ap_uint<12> data_N_bits;

    //scale to N bits
    data_N_bits = scale_bits*data_in;

    //stores the values in an array
    x_i = xr[i];
    xr[i] = data_N_bits;

    //Calculate the average value
    xx = factor*(xr[i] - x_i);
    data_out_prev = data_out_prev + xx;

    //Reset the array index
    i = i+1;
    if (i >= N){
        i=0;
    }

    *data_out = data_out_prev;
}

```

1. IP input parameters

2. IP OUTPUT parameters

3. static variables

4. local variables

5. descriptive code of functionality

6. Output value assignment

Figure 74: Moving average filter C code.

For writing C++ code compatible with Vitis HLS, the user must consider the following technical aspects:

- The IP block interface is defined as a function declaration. Input signals are declared as function parameters (ap_uint<12> data_in) whilst output signals are declared as pointers (ap_uint<12> *data_out).
- When declaring variables, it's essential to note that only those declared as static will retain their previous cycle's value. Non-static variables are reinitialized in each cycle of the IP block.

Steps to create this IP using Vitis HLS:

1. Run Vitis HLS and select the option to create a new project (File>New project).
2. A prompt will appear on the screen, allowing you to designate a name for the project and specify the desired location for storing all associated files. Proceed by clicking "Next" to continue.

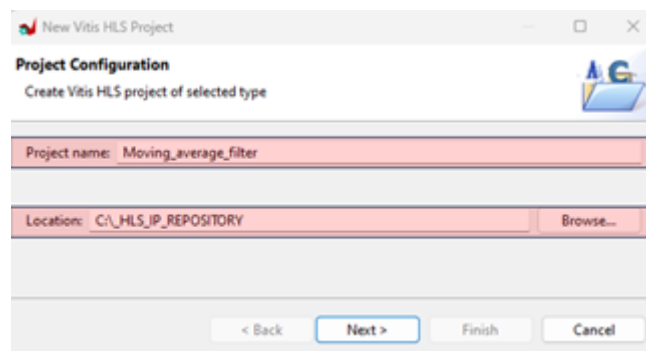


Figure 75: Vitis HLS project settings I.

3. In the subsequent two windows, proceed by clicking "Next."
4. Specify the frequency as 10ns (100MHz) and the target device (xcck26-sfvc784-2LV-c). Click "Finish" to finalize the setup.

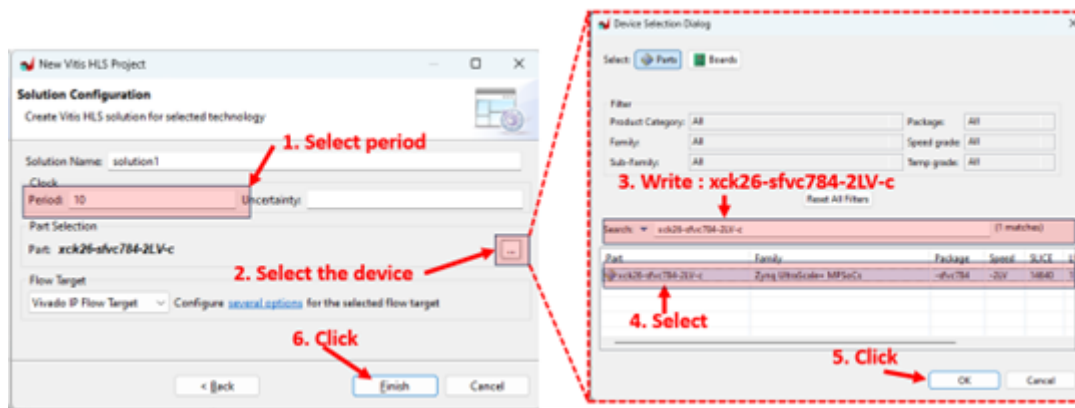


Figure 76: Vitis HLS project settings II.

5. Create a new source file with a ".cpp" extension bearing the identical name as the IP intended for generation.

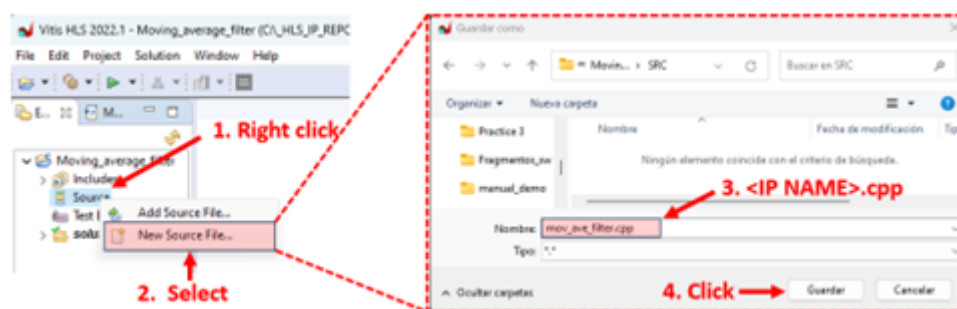


Figure 77: Vitis HLS project settings. Add sources.

After creating the .cpp file, simply copy and paste the descriptive code outlining the necessary functionality for the IP. Save it by pressing the saving shortcut Ctrl + S.

6. Navigate to the "Project/Project Settings, select "Synthesis" and designate the declaration of the top function.

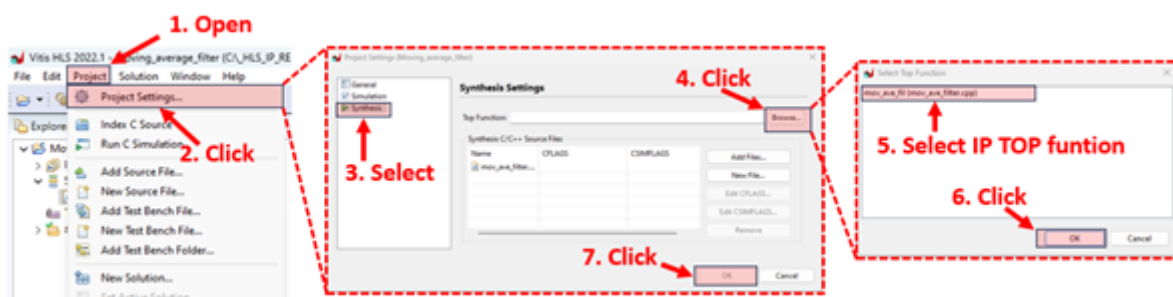


Figure 78: Vitis HLS project settings. Select top function.

7. Add the "INTERFACE" directive with the option "AP_NONE" to all the IP inputs and outputs to simplify the IP interface generated.

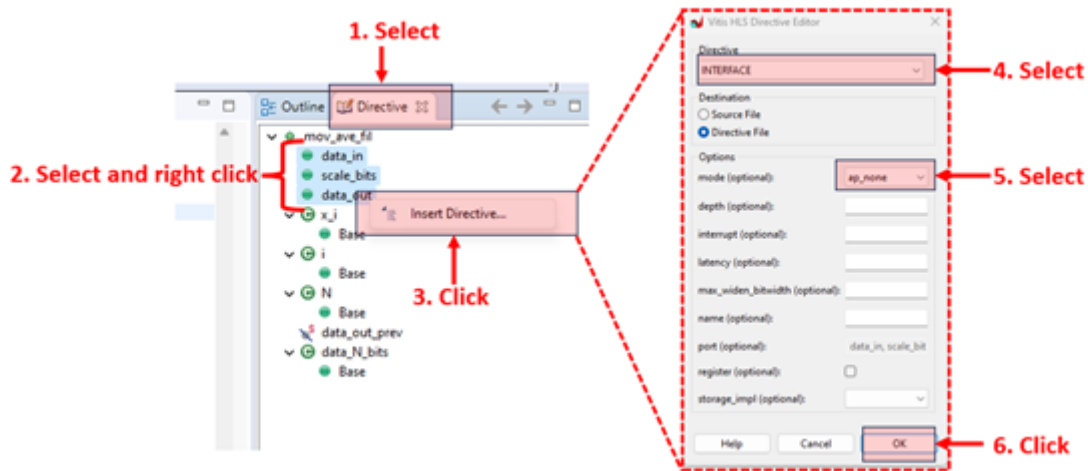


Figure 79: Vitis HLS project settings. Adding directives.

Vitis offers a wide variety of directives that allow precise control of the IP to be synthesized: resources used, process parallelization, timing, etc. Detailed information can be found at: <https://docs.xilinx.com/r/2022.1-English/ug1399-vitis-hls/Adding-Pragmas-and-Directives>

8. Initiate the synthesis process by clicking the play button on the interface. Subsequently, in the pop-up window, proceed by clicking "OK" with the default values.

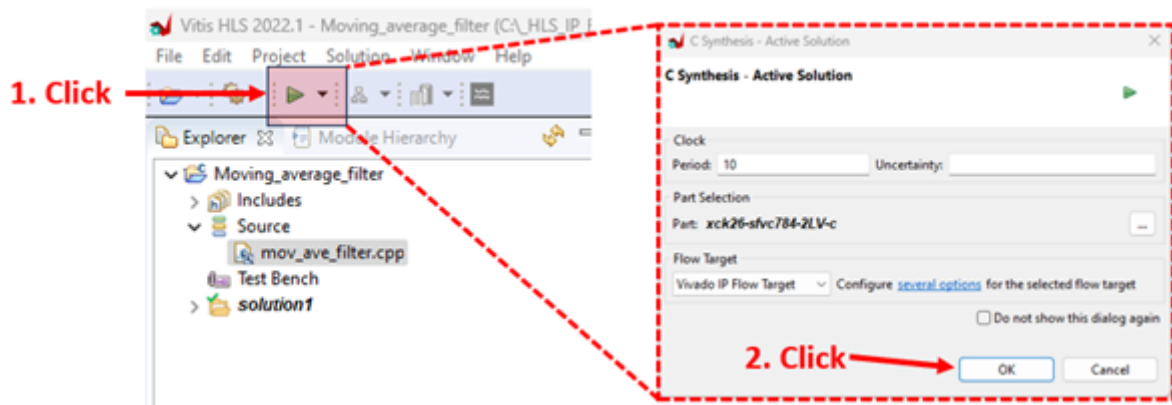


Figure 80: Synthesizing the IP in Vitis HLS.

9. Perform the implementation of the IP to make it accessible by a Vivado project. See details in the figure 81.

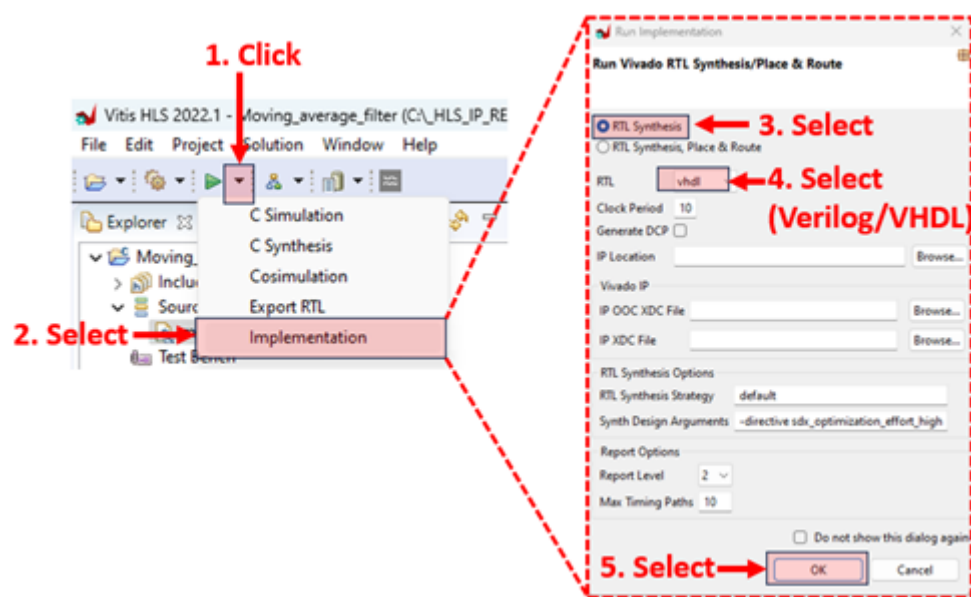


Figure 81: Implementing the IP in Vitis HLS.

10. Check the summary provided of the IP resulting from the process. Special attention must be given to:
 - Latency. The execution time is determined by latency. Latency multiplied by clock period is the total time that the IP requires to fully execute.

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Tripp Count	Pipelined	BRAM	DSP	FF	LUT	URAM
mov_ave_fil					19	190.000		20		no	0	5	597	1177	0

- Resources to be consumed by the IP.

Resource Usage	
	VHDL
SLICE	0
LUT	789
FF	677
DSP	5
BRAM	0
URAM	0
LATCH	0
SRL	0
CLB	0

11. Final result: moving average filter IP

The IP block is fully created and can be included in a Vivado project, the IP block generated following this procedure will present the following interface:

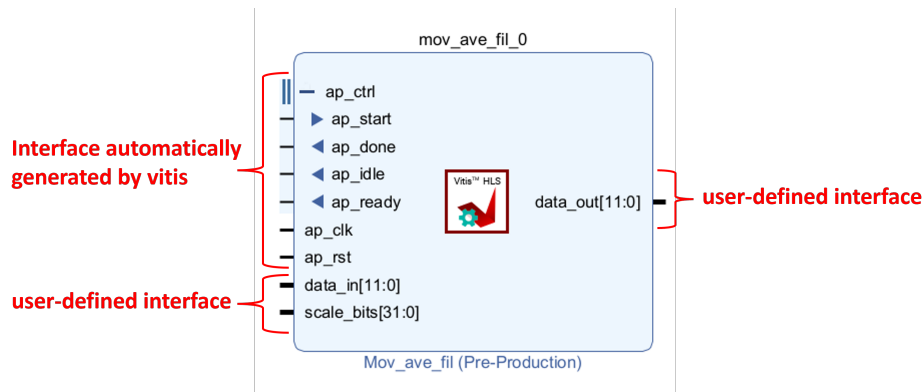


Figure 82: Result of the IP generated in Vitis HLS.

Note: an HLS IP includes two parts in its interface: a user-defined interface resulting from the function arguments definition and an automatic interface that includes some signals for managing the IP behavior, see figure 83 for detailed functioning of the automatic interface signals.

Interface ap_ctrl hs

ap_start: input to order start of the execution
ap_idle: this output indicates that the IP cores is busy
ap_ready: this output indicates that the IP cores can admit new data
ap_done: this output indicates that the execution has concluded

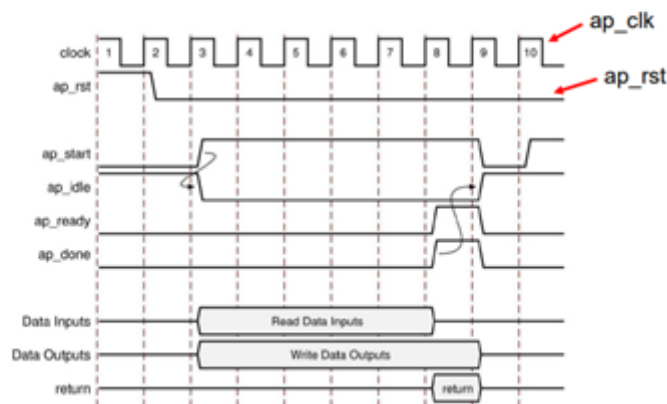


Figure 83: IP extra signals created by Vitis HLS.

PID compensator:

This IP block implements a PID controller that regulates the voltage value at the converter output. The user has to follow the same steps shown in the previous IP procedure but using the source files provided for this IP. The next figure provides a view of the implemented algorithm and of the finished IP, note A0, A1, and A2 are the compensator coefficients.

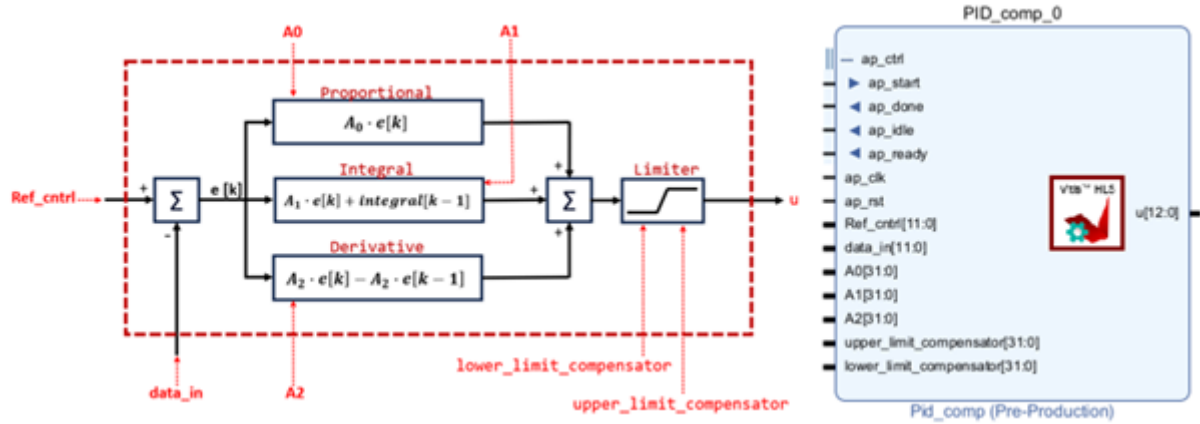


Figure 84: PID compensator IP. View of inner algorithm and IP general view.

DPWM:

This IP generates the PWM signal that controls the MOSFET of the Buck converter and a signal that triggers the analog acquisition of output voltage. The user has to follow the same steps shown in the previous IP procedure but using the source files provided for this IP. The next figure provides a view of the implemented algorithm timing and of the finished IP.

Nr is a 13 bits integer that limits the upper level of the sawtooth carrier signal. **HB** is the gating signal. **Cs** is the sample step, an integer of 13 bits. This number defines how many 100MHz pulses will be among to sampling instants.

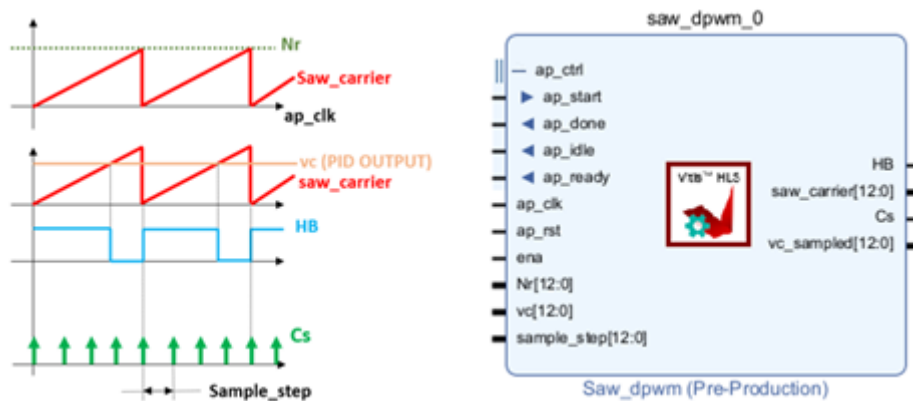


Figure 85: PID compensator IP. View of inner algorithm timing and IP general view.

8.2.3 Configuring SmartRCP_Template in Vitis Vivado

This section details the required steps for using the SmartRCP_Template and adapting it for creating a new project. This section emphasizes on:

- Adaptation of SmartRCP_Template to meet the requirements of a new project.
- Communication between the microprocessors and the FPGA.
- Synthesis, implementation, and export. These steps are critical to translating the conceptual design into a functional implementation on the FPGA.

It is essential to keep in mind that this tutorial is focused on serving as a guide for the creation of projects based on SmartRCP_Template. For this reason, certain more advanced aspects, such as the generation of IP designs in VHDL or Verilog, will not be explained in detail in this tutorial. There are additional IPs that cover auxiliary functionality. All those IPs which are not the core of the system are not covered. As they are included, users can explore them and recycle them into future projects.

Adaptation of the SmartRCP_Template project:

Follow the steps:

1. Open SmartRCP_Template project and eliminate the components used for testing purposes leaving only PSC_CarrierBoard_driv IP, IP AXI_PSCoscilloscope and UltraScale blocks. The IP blocks remaining in Vivado project are shown in the figure 86:

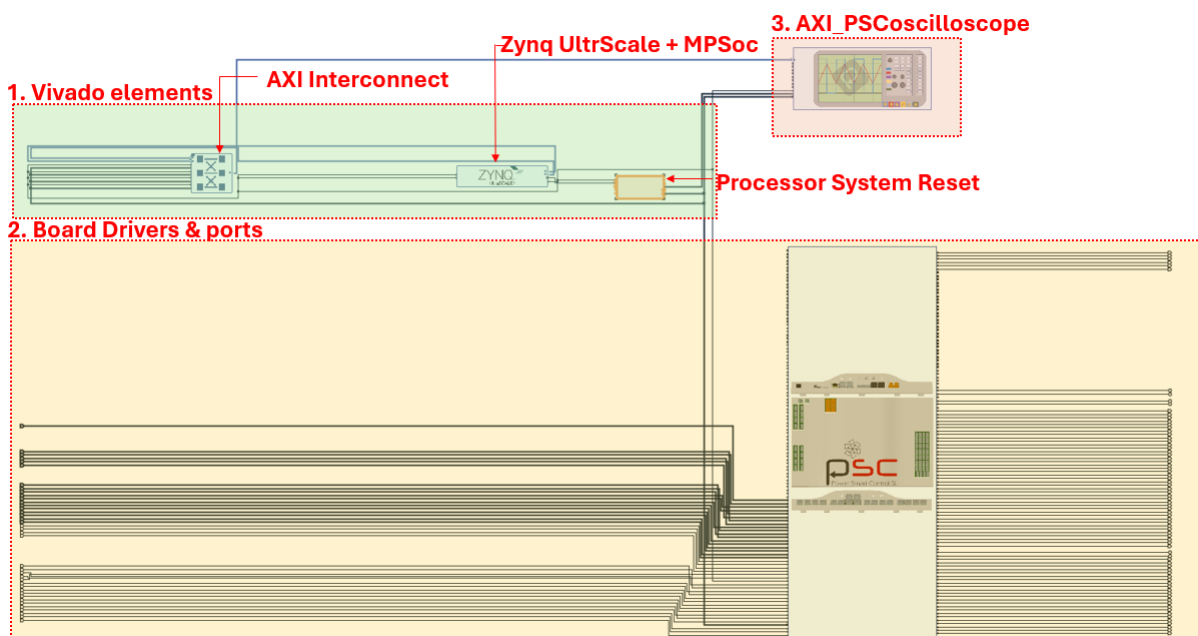


Figure 86: Vivado IP integrator IP blocks after deleting testing blocks.

2. Add the IPs generated in Vitis HLS to the Vivado project (Moving average filter, PID compensator and DPWM IPs) and add the IPs to Vivado IP integrator.

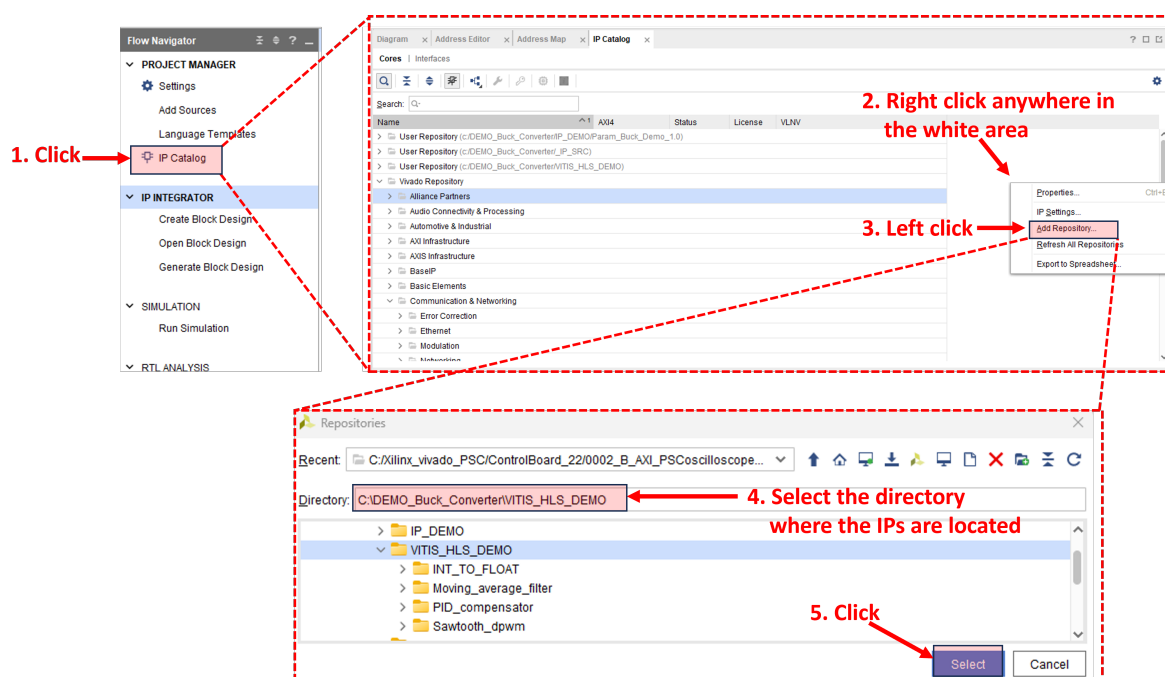


Figure 87: Adding Vitis HLS IP blocks created to Vivado IP repository.

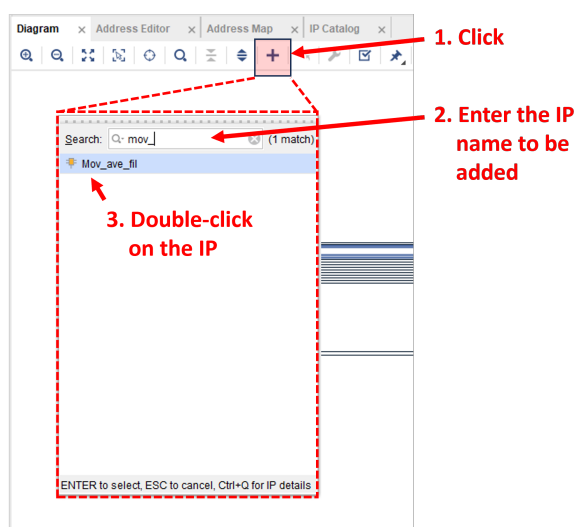


Figure 88: Adding Vitis HLS IP blocks to Vivado IP integrator.

Note: Some additional IP blocks are required to create the whole system. One of them will act as a coordinator so that the PID IP is only executed when the moving average has finished its calculations, another block is used to transform the measured signal of the ADC into a 12 bits signals. Those IP blocks have been provided in the example and are implemented as simple VHDL IPs.

3. Connection of IP blocks. A green banner (Designer Assistance) will appear at the top of the window, allowing you to perform all automatic connections (CLK, RESET, AXI, etc). Select the "Run Connection Automation" option, make sure all checks are enabled and click OK. See the next two figures (89 and 90).

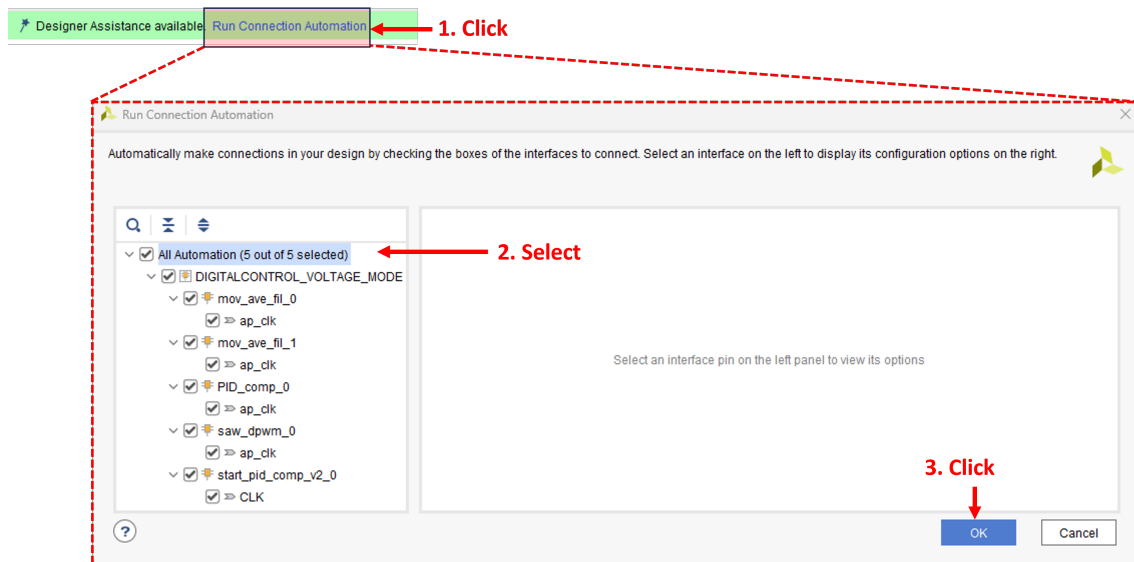


Figure 89: Designer Assistance configuration.

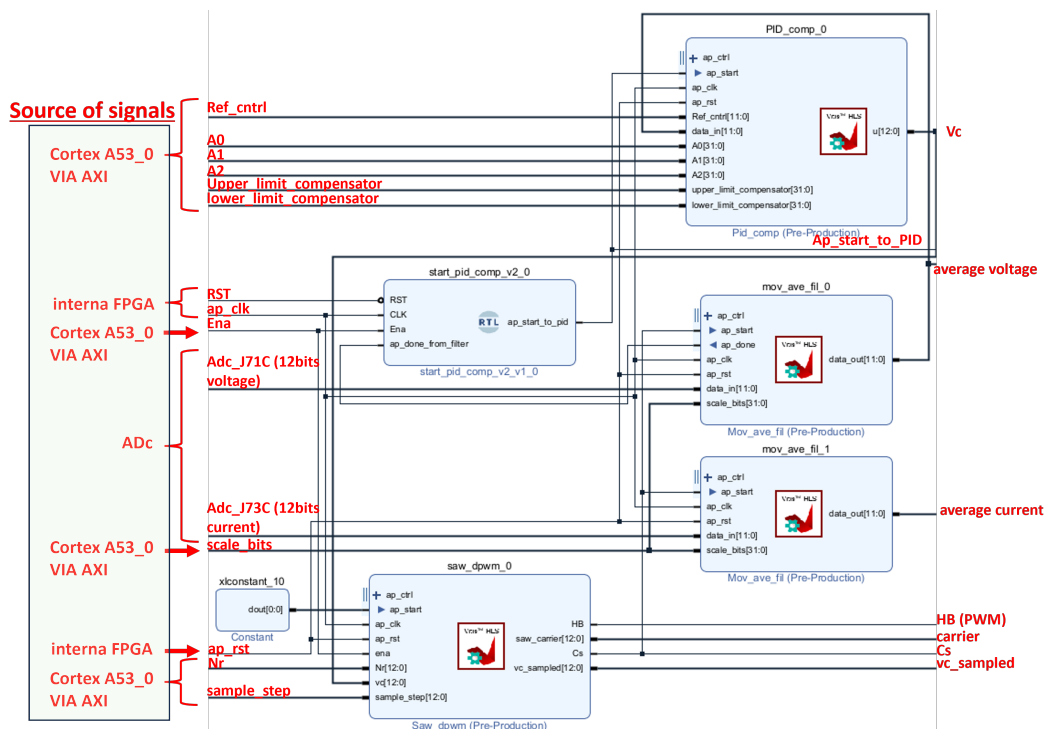


Figure 90: Required connection of signals between IP blocks.

4. Connection to PSC_CarrierBoard_driv IP. Data outputs that are not required can be left unconnected. In the case of the inputs of the IP blocks, if they are not used, they cannot be left unconnected, so it is necessary to use constants IP blocks to connect them. PSC_CarrierBoard_driv IP will be connected as shown in the figure 91:

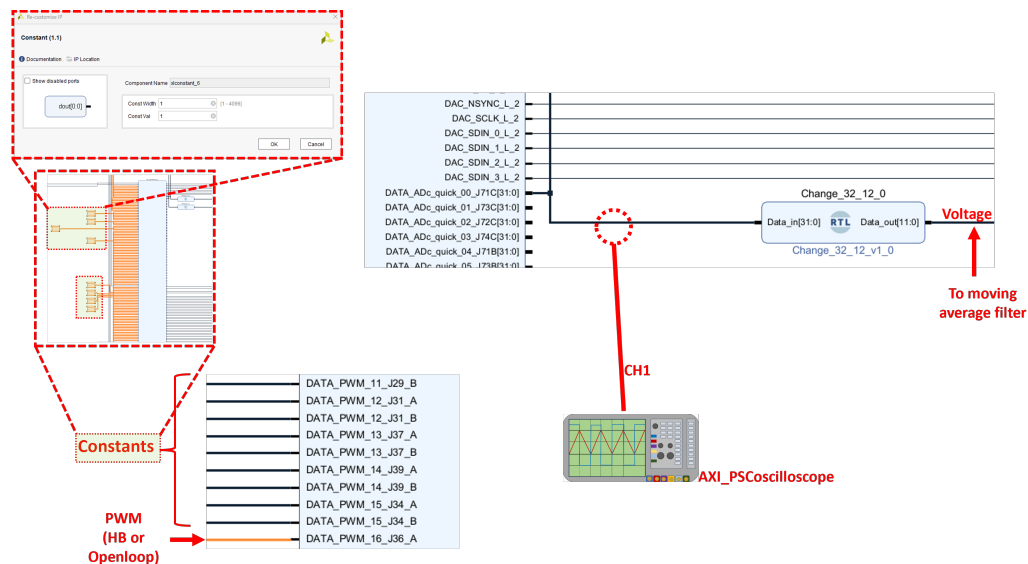


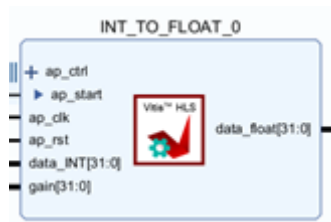
Figure 91: PSC_CarrierBoard_driv IP connection.

5. Connection of remaining IP blocks. A brief description of remaining Ips and their functionality is provided below:

- **Change_32_12:** IP block made in VHDL that generates an output data consisting of the 12 least significant bits of the 32bit input data. It is used to adapt the ADC data to the input of the moving average filter.
- **Constant:** IP block provided by Vivado. It provides a constant value of a selectable number of bits. It is used to assign values to input ports that are not being used. The value used has no relevance as the ports are not used. If input ports are left unconnected a place-routing error will arise in Vivado.
- **Concat:** It allows to create a new signal by joining several smaller signals. It is used to adapt the data width. Extra bits required are filled with '0'.
- **PWM_SELECTOR:** It behaves as a multiplexer, allowing to provide at the output a fixed PWM (open loop mode) or a variable PWM (closed loop mode). The selection is done by the mode CTRL_SEL input.

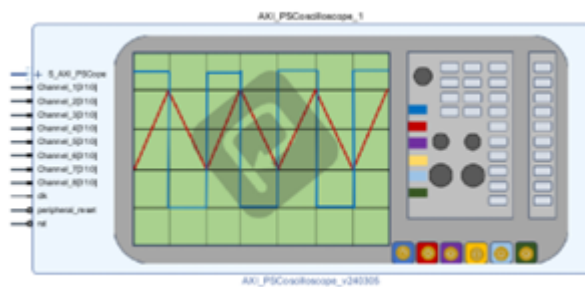


- **INT_TO_FLOAT:** It converts integer data types to float types and multiplies by a gain value. AXI_PSCoscilloscope works with floating data so it is necessary to convert data.



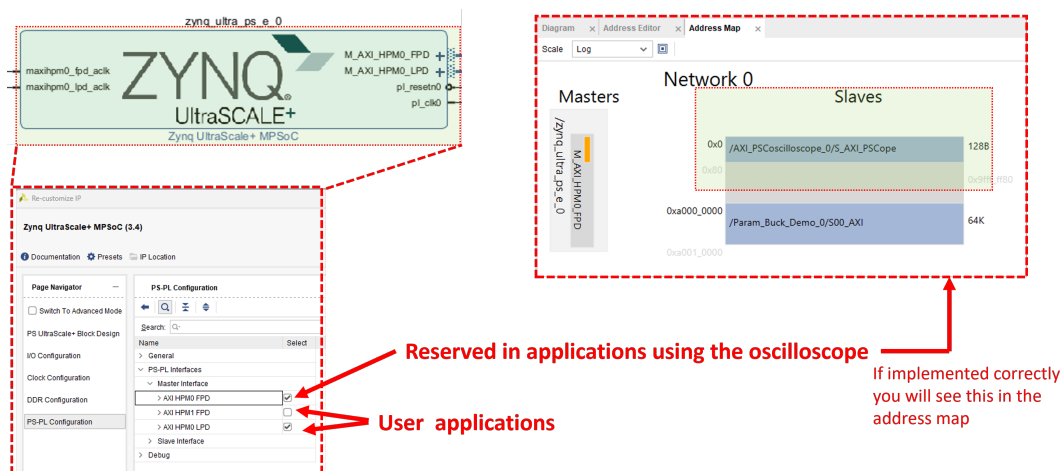
• **AXI_PSCoscilloscope:** This IP implements oscilloscope functionality inside the FPGA. It allows to observe internal signals. The following waveforms have been measured in this tutorial:

OSCILLOSCOPE CHANNEL	MEASURED WAVEFORM
1	Voltage read by ADC
2	Moving average of the voltage (channel1)
3	Last version date (Vivado Project)
4	Last version date (PSC Oscilloscope)
5	PID control signal
6	Modulation carrier signal
7	PWM OUTPUT
8	Control selector signal



The values measured by each channel must be in floating data type. These values are sent to PSCoscilloscope application (via ethernet) so that the user can manage the oscilloscope with an interface quite similar to a desktop oscilloscope.

Note: PSC_CarrierBoard_driv transmits a large volume of data to the processor core where it is implemented. For this reason, one of the three available AXI interfaces is reserved so that there are no concurrency problems if several processor cores want to communicate with the FPGA.

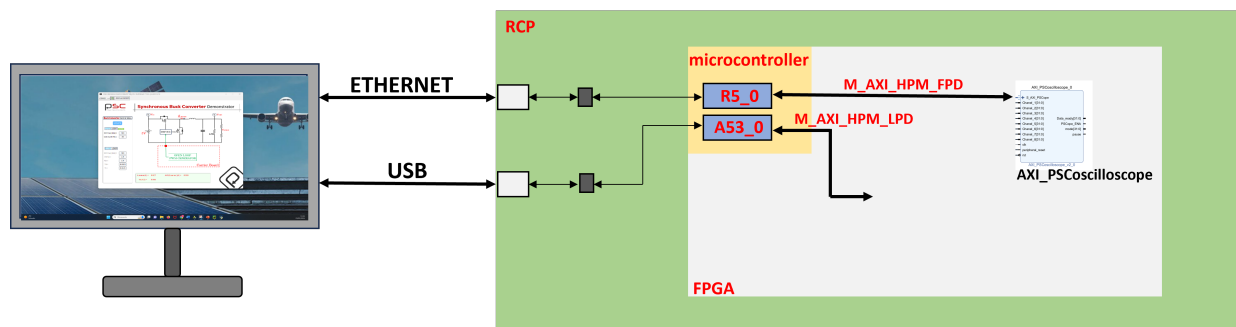


• **ILA:** ILA is a tool provided by Xilinx that provides a logic analyzer functionality useful for debugging the designs. The process for adding it to the block diagram is the same as any other IP.

Configure the IP by double-clicking on it for a data depth of 4096 and 10 probes:

Probe	Signal name
0	PWM_OUTPUT
1	CTRL_SEL
2	MOV_AVE_VOLTAGE
3	MOV_AVE_CURRENT
4	ADc_VOLTAGE
5	Cs
6	Vc_sampled
7	Ap_start_to_pid
8	SAW_Carrier
9	Vc

• **Param_Buck_Demo:** It is an AXI4 peripheral whose function is to transmit information between the processors and the FPGA. It is used to transmit the necessary parameters to the IPs, these parameters are configurable by the user by the provided computer application.



Microprocessor-FPGA communication (AXI4 peripheral):

1. Define the input and output signals needed. The following table summarizes all the signals to be transmitted between the Cortex A53-0 microprocessor (application developed for this tutorial) and the FPGA.

Signal name	I/O	Number of bits
OpenLoop_ENA	O	1
N_clk_period	O	11
N_clk_dutyCycle	O	11
CloseLoop_ENA	O	1
Nr	O	13
Samp_step	O	13
Scale_bits	O	31
Vref_cntr	O	12
A0	O	32
A1	O	32
A2	O	32
Upper_limit	O	32
Lower_limit	O	32
CTRL_SEL	O	2
AVG_Vsense	I	32
AVG_iLsense	I	32

2. Create the AXI peripheral. See the following figures (92, 93, 94. and 95) where the steps are detailed.

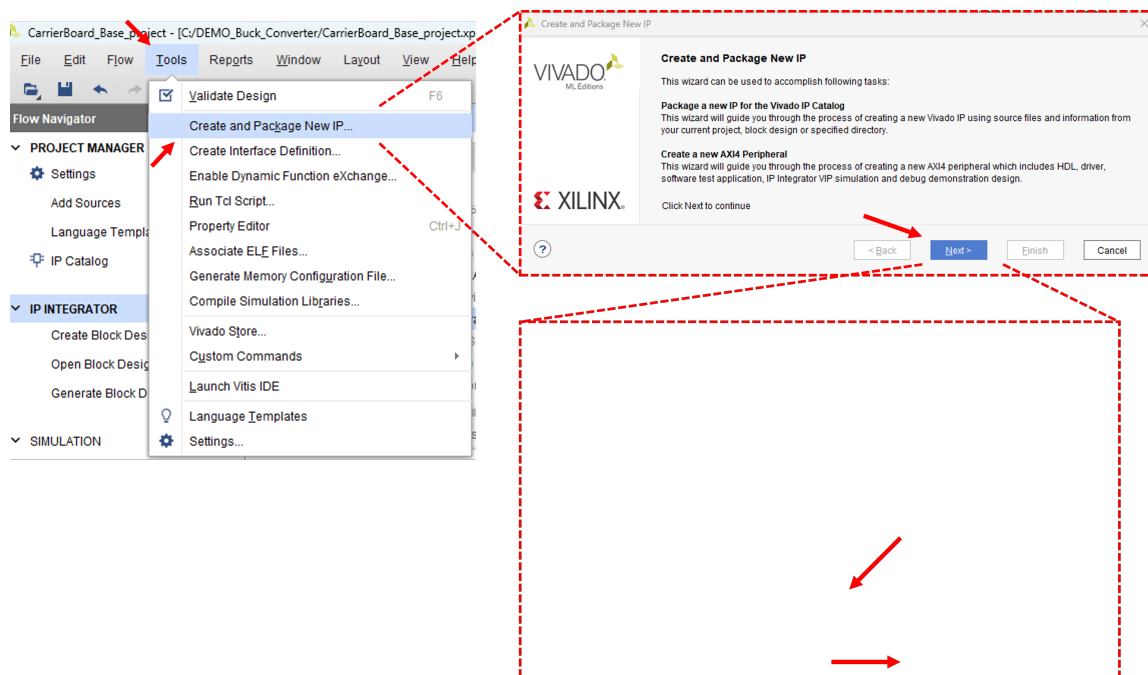


Figure 92: AXI peripheral creation I.

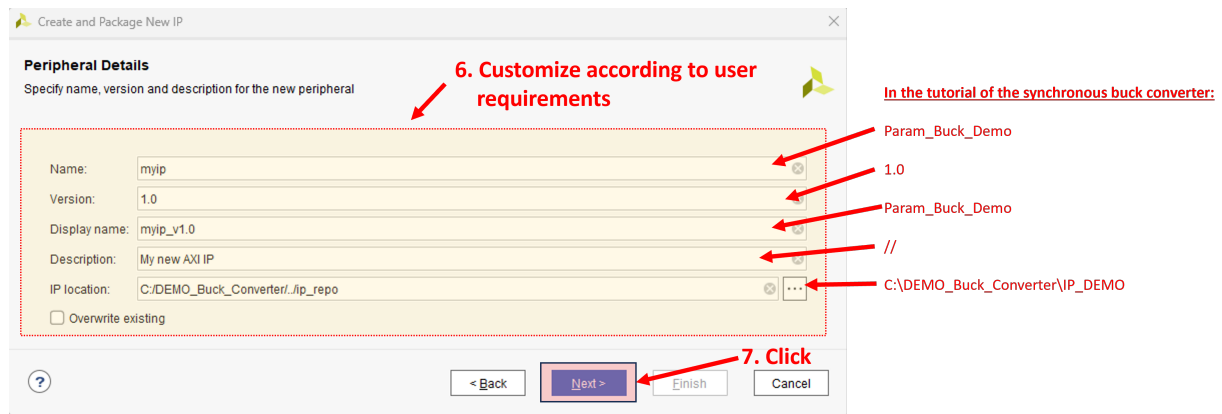


Figure 93: AXI peripheral creation II.

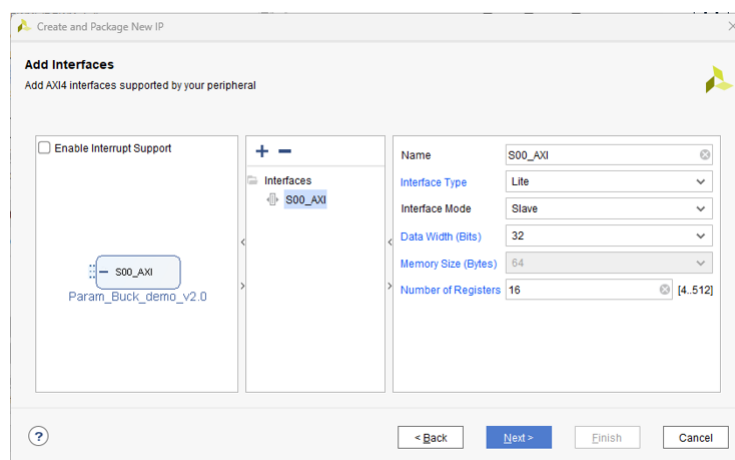


Figure 94: AXI peripheral creation III.

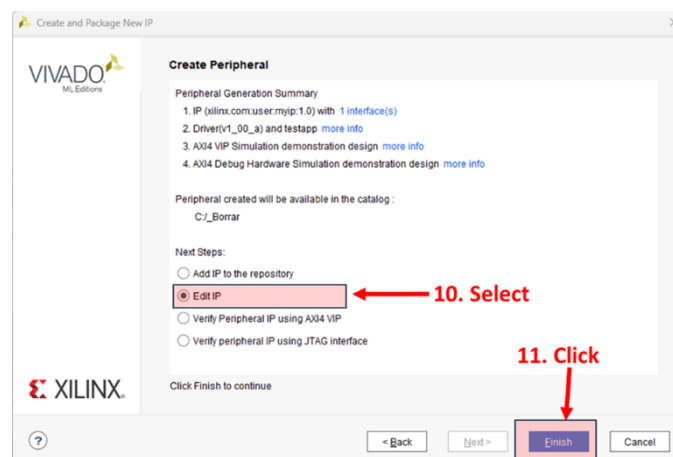


Figure 95: AXI peripheral creation IV.

3. Click on finish. The following interface will appear.

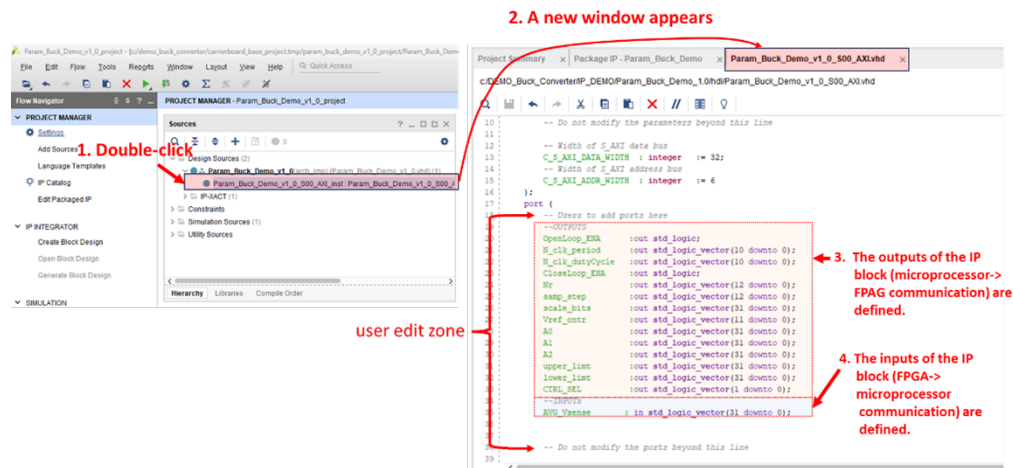


Figure 96: AXI peripheral creation V.

Go down in the code to the part of "reg_data_out" signals, where you can define in which direction each of the input signals will be read. See figure 97.

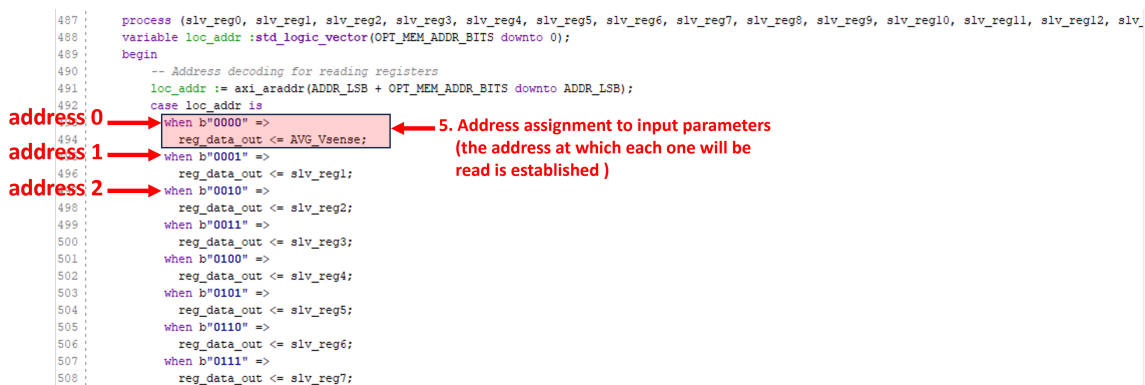


Figure 97: AXI peripheral creation VI.

Go down to the "Add user logic here" section, almost at the end of the code. And update the code according to the figure 98:

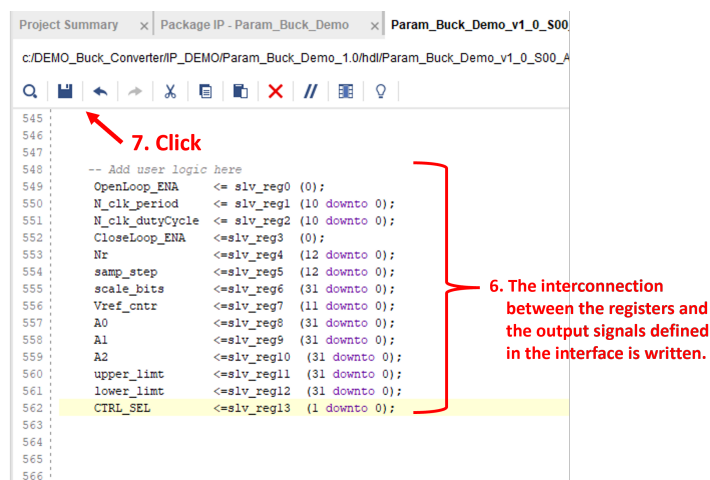


Figure 98: AXI peripheral creation VII.

Open the other code tab (the tab whose name ends in “_V1_0.vhd”) and update the codes, see the figures (99, 100 and 101) below:

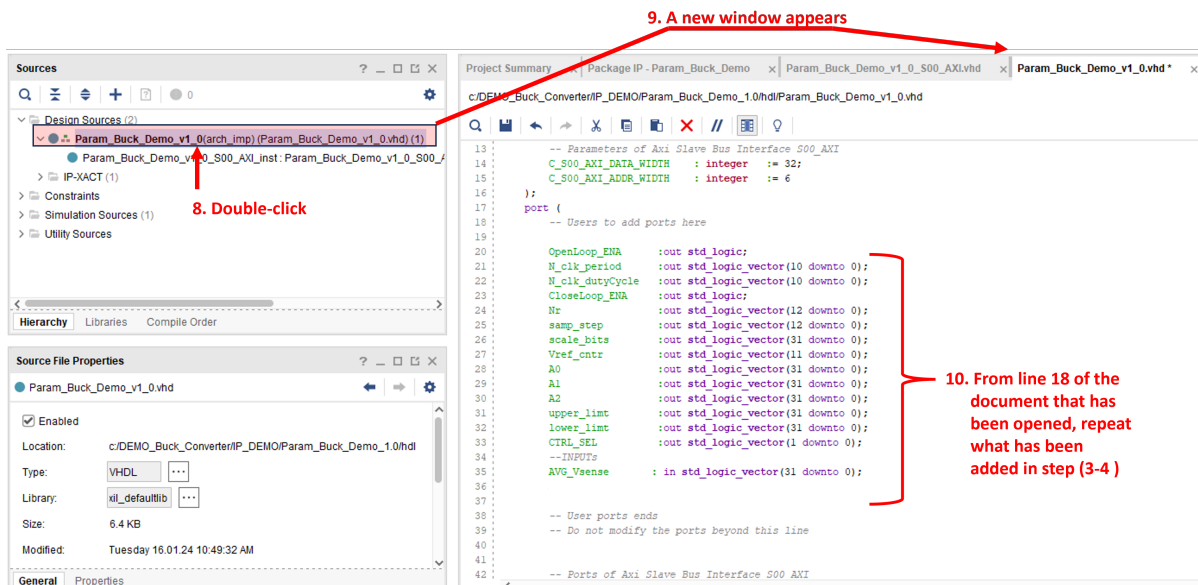


Figure 99: AXI peripheral creation VIII.

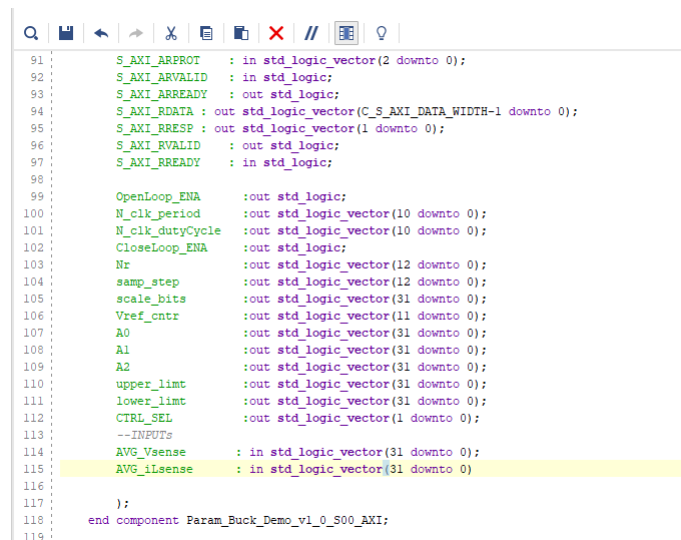


Figure 100: AXI peripheral creation IX.

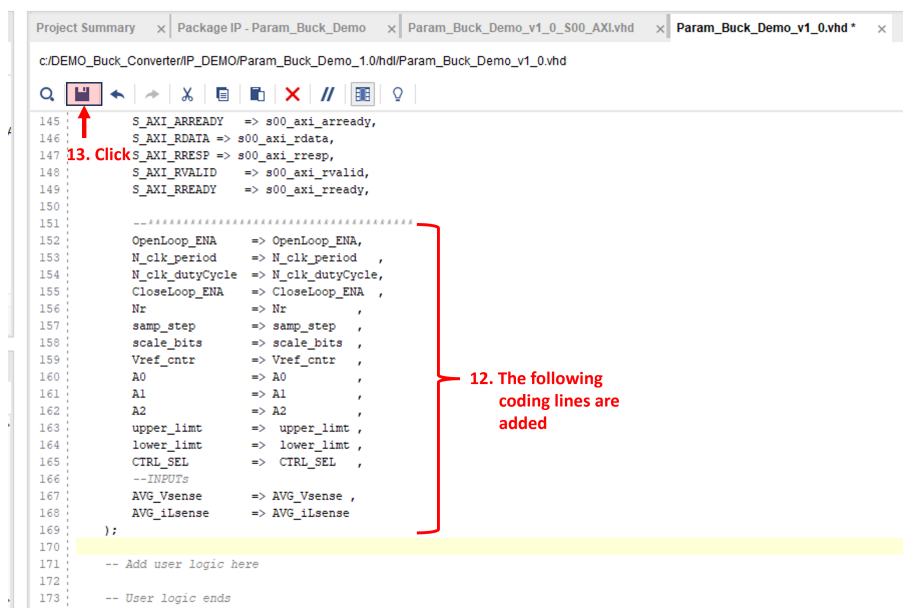


Figure 101: AXI peripheral creation X.

4. Package the IP and close it. Steps are included in the following images (102, 103 and 104):

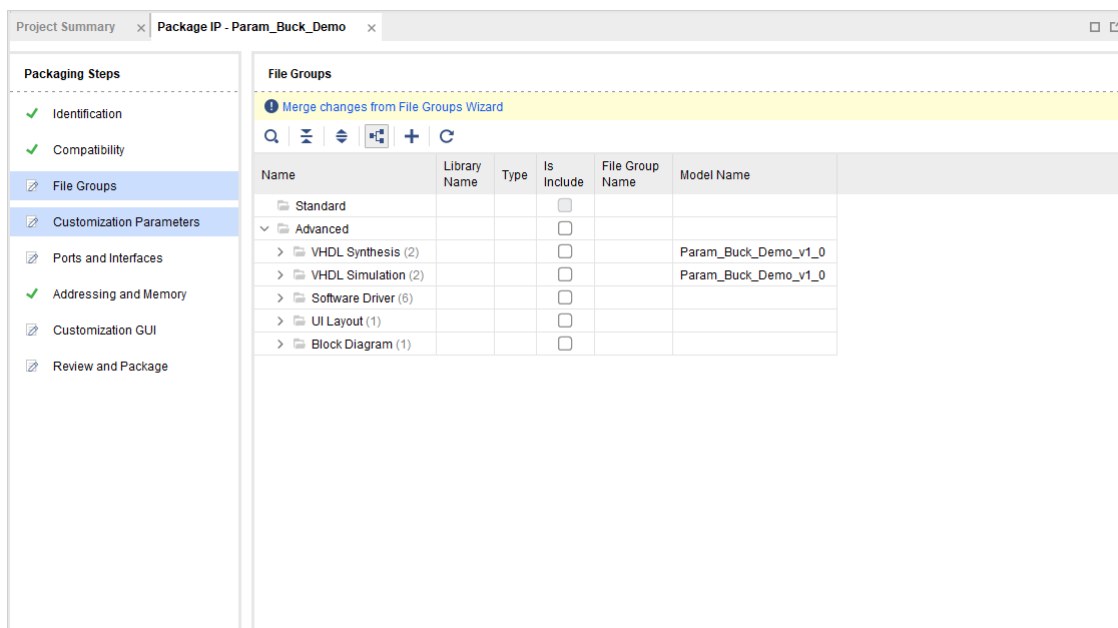


Figure 102: AXI peripheral creation XI.

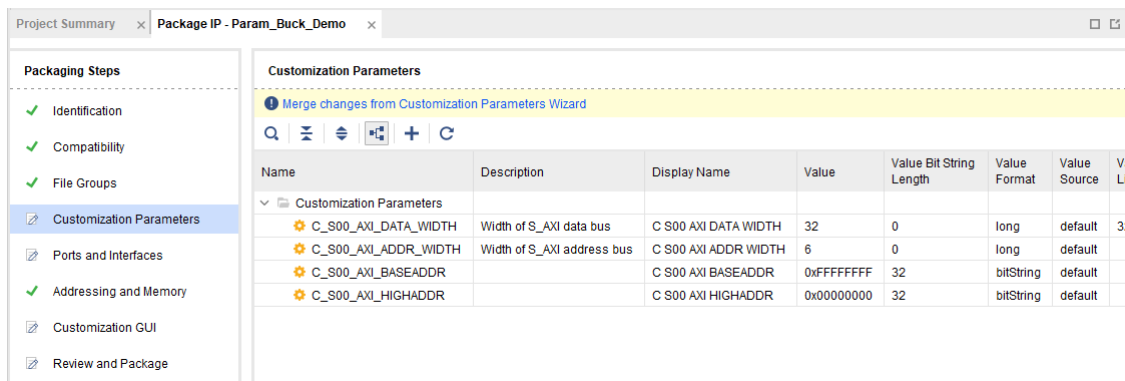


Figure 103: AXI peripheral creation XII.

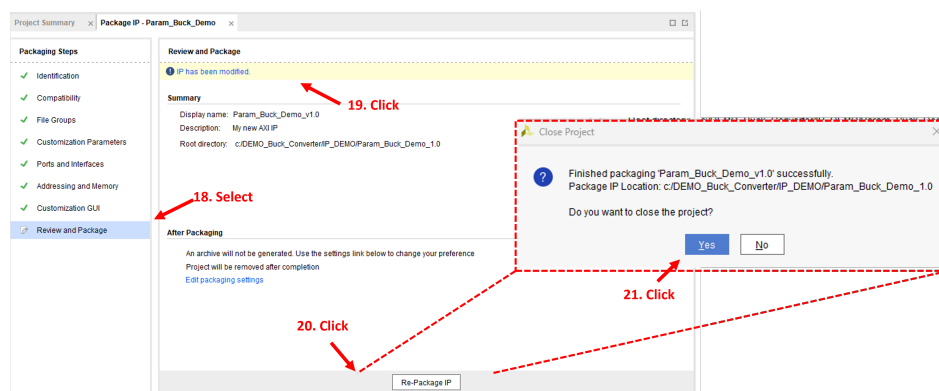


Figure 104: AXI peripheral creation XIII.

Important note: Due to a Xilinx Vivado known issue, AXI IPs will not automatically include all source files. To overcome the issue, AXI IP makefile has to be manually modified. To do so:

a) Note the IP real path:

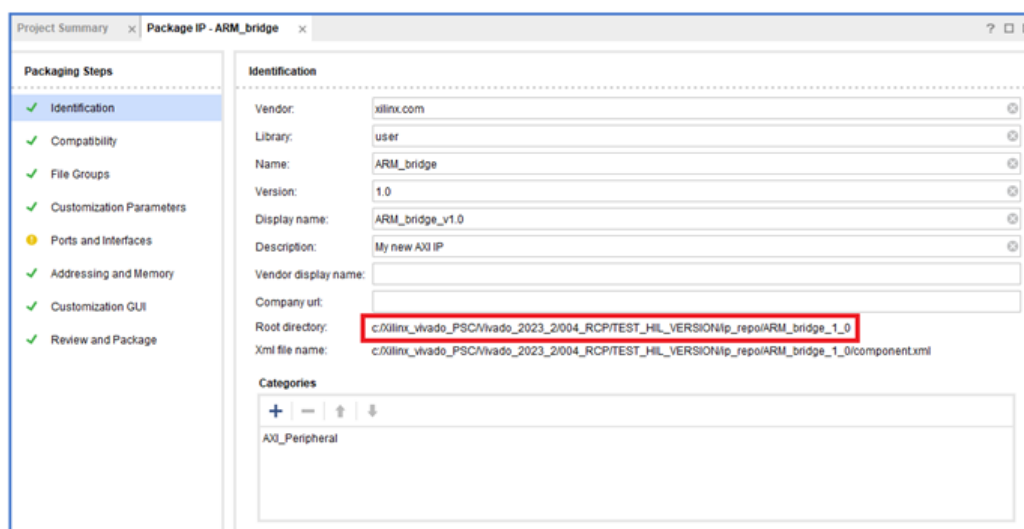


Figure 105: AXI IP real path

- b) Go to the path \drivers\YOUR_IP_NAME\src and locate make file

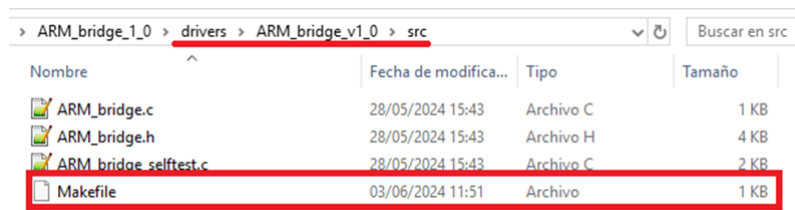


Figure 106: Makefile path

- c) Identify issue lines

```
INCLUDEFILES=* . h
LIBSOURCES=* . c
OUTS = * . o
```

- d) Substitute by these ones

```
INCLUDEFILES=$(wildcard *.h)
LIBSOURCES=$(wildcard *.c)
OUTS=$(wildcard *.o)
```

- e) Click on Re-package IP (step 20), otherwise the change in makefile will not be taken into account and the issue will persist.
5. The IP that has just been generated (AXI4 PERIPHERAL) is available in the IP catalog so that it can be added to the block diagram. See the next figure: Finally, connect the data inputs and outputs.

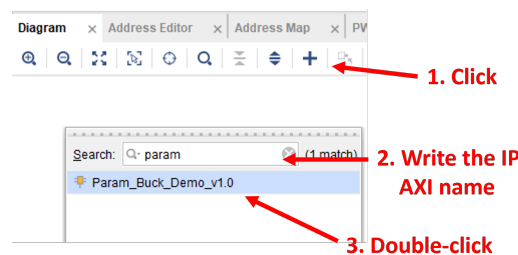


Figure 107: AXI peripheral creation XIV. Including IP in Vivado.

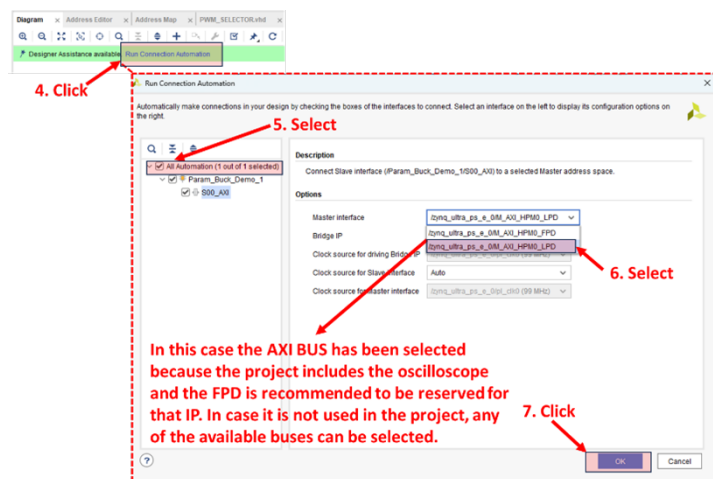


Figure 108: Configuration of the AXI IP in Vivado.

Finally, connect the data inputs and outputs.

Synthesis, implementation and export of the Vivado project:

Up to this point, SmartRCP_Template project has been customized to control a buck converter. To optimize the organization of the design subset has been created, those subsets include a '+' mark that expands the block to reveal all functionality inside when pressed.

Follow the next steps:

1. Validate the design and create the wrapper. See the next figures (107, 108 and 109).

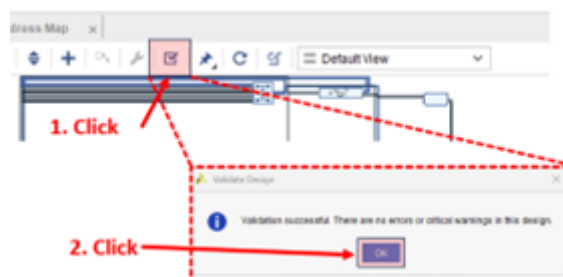


Figure 109: Validate the Vivado design.

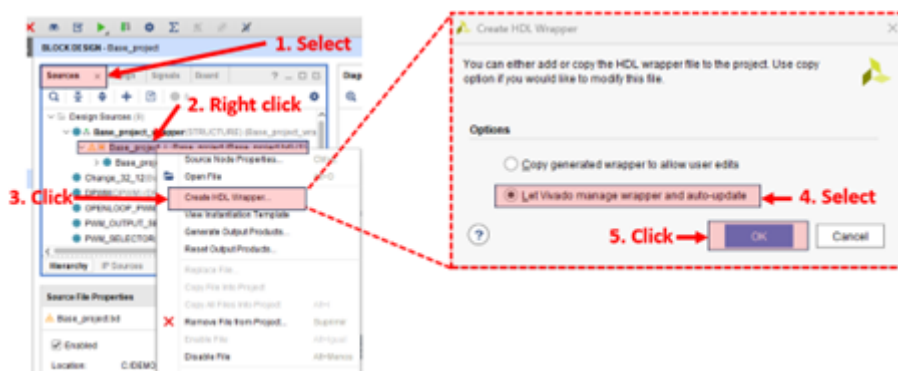


Figure 110: Create the wrapper I.

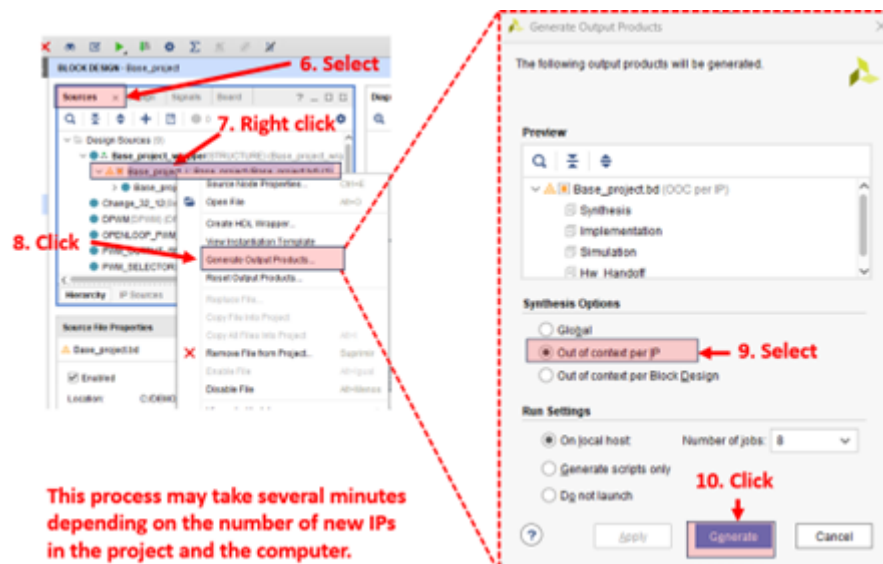


Figure 111: Create the wrapper II.

2. Click on Generate Bitstream and ensure the option “Launch runs in local host” is enabled and click OK. This process will take some time.
3. Export the results files so they can be used in Vitis SDK.

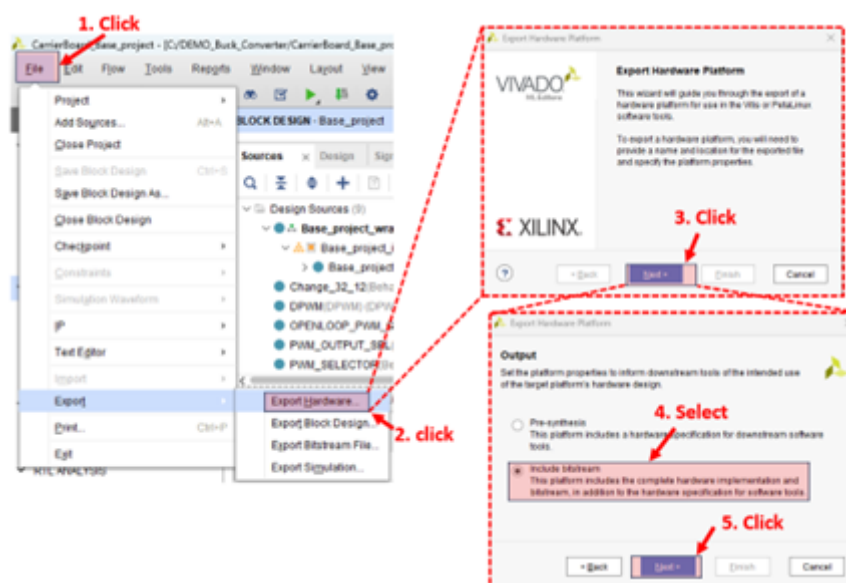


Figure 112: Export Vivado project I.

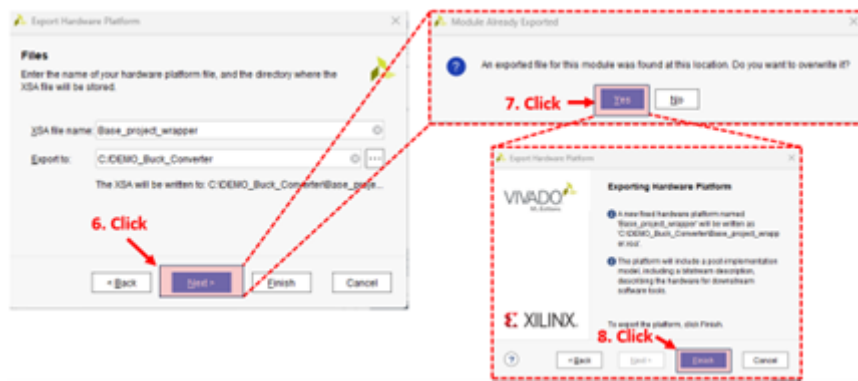


Figure 113: Export Vivado project II.

8.2.4 Configuring processors with Vitis IDE

This section provides a detailed description of the procedure for programming the processors available and to use debug mode. The generation of the files needed to load the design directly from microSD card is also covered. The next figure provides a view of the resources available on the K26. Note there are 2 real time-oriented microprocessors (ARM – R5) and four application-oriented microprocessors (ARM – A53); ARM-R5-0 is used by PSCoscilloscope and it is not accessible to the user, that is the only limitation in Ps side to the user.

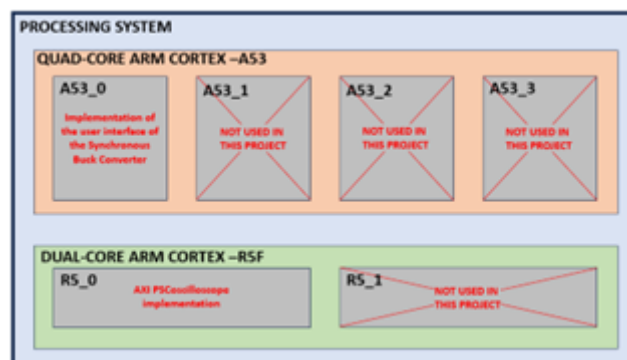


Figure 114: Microprocessor resources available in KRIA K26

User interface (cortex A53_0):

Launch Vitis IDE and follow the steps covered in the next figures (113, 114, 115, 116, and 117):

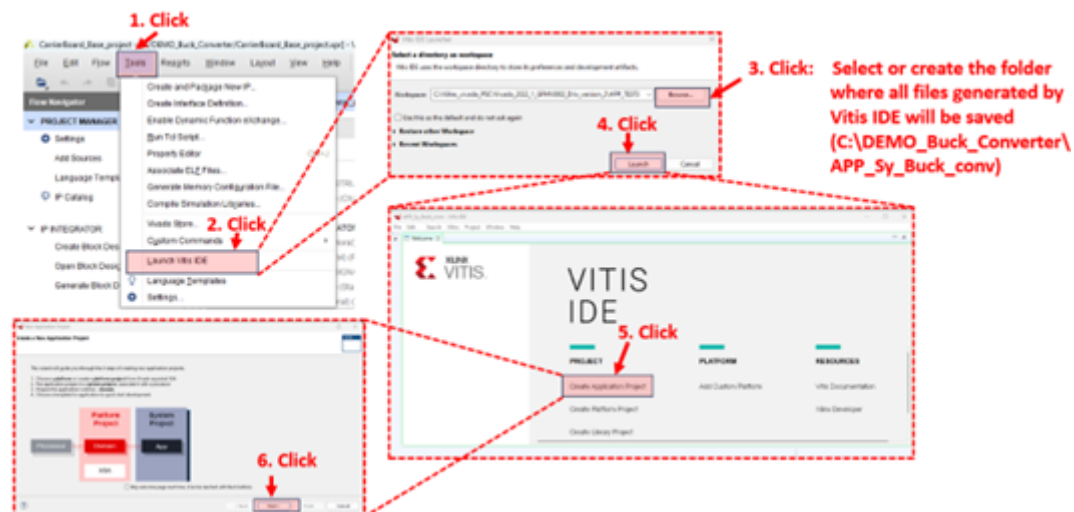


Figure 115: Configuring Vitis project I.

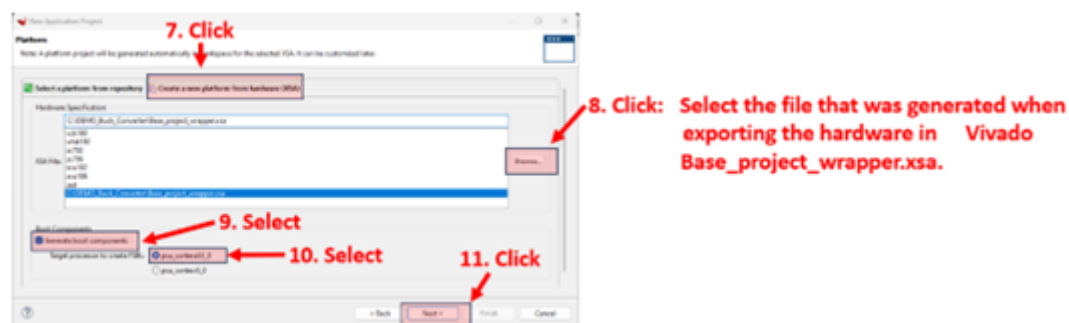


Figure 116: Configuring Vitis project II.

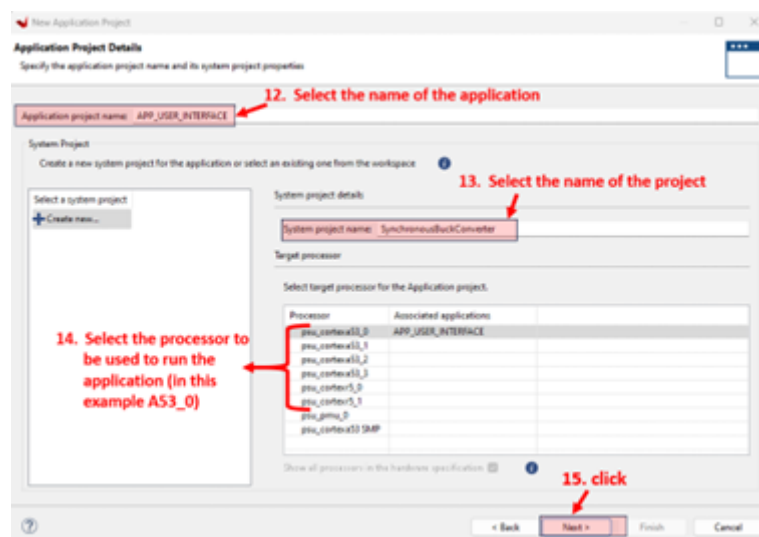


Figure 117: Configuring Vitis project III.

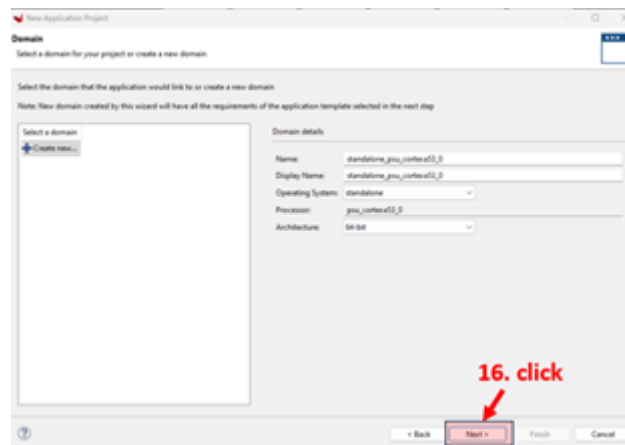


Figure 118: Configuring Vitis project IV.

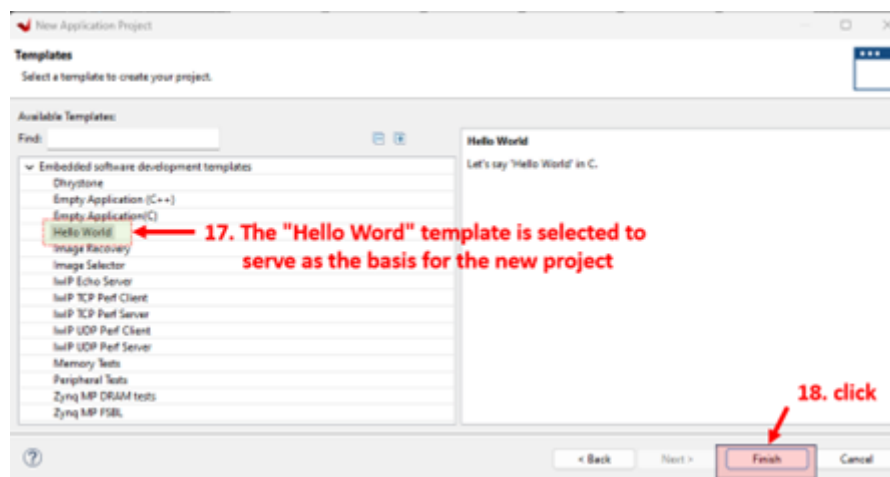


Figure 119: Configuring Vitis project V.

After clicking on finish, a new window will appear, that window is comprised of three main parts:

- Explorer: locates all the files and source files that need to be modified to adapt the template to the current project.
- Quick options panel: you will find the necessary options to compile the code and download, and program the K26 card, among others.
- Main panel: the area where the project files are opened and edited.

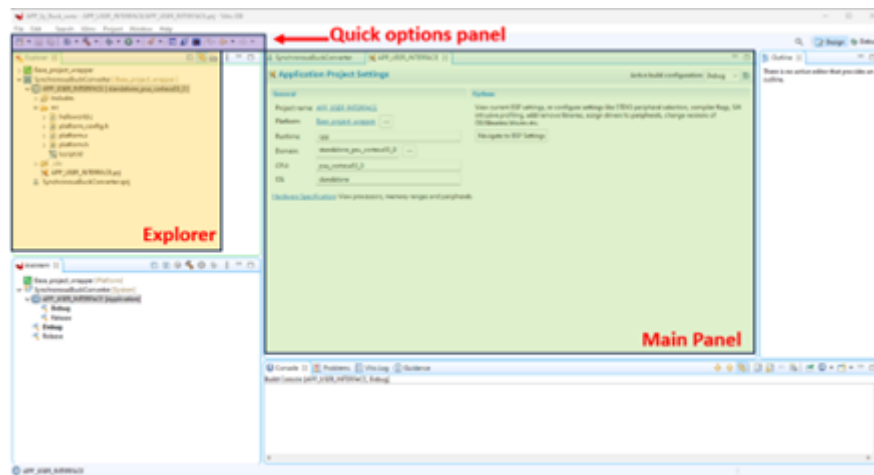


Figure 120: Vitis IDE main sections.

Before starting to generate the code to be executed in the A53_0 processor, the project is built to update the drivers that Vivado generates to facilitate the use of the AXI4 peripheral. This process may take a few minutes depending on the computer on which it is executed.



Figure 121: Building in VITIS IDE.

The next step consists of including the required code in the project. The code will be in charge of two main tasks:

- Assign the drive control start values and enable the PWM output. The processor will send to the FPGA the parameters by writing to the AXI4 IP registers.
- The processor will listen to the serial port waiting for any user command, data request, or control mode change.

To write to the AXI4 peripheral registers, a library has been generated that provides the necessary functions to write to the IP outputs and to read the existing values in the IP inputs. To access the functions this sentence must be included: `#include "Param_Buck_Demo.h"`. This library includes a list of the available registers, and write and read functions to those registers. For writing and reading any register, it is required to know the base address of the IP. That address is written down in the Vivado Address editor, see the figure below 120:

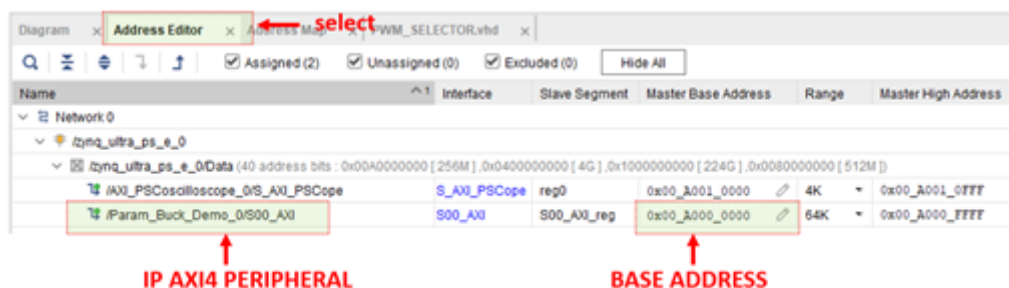


Figure 122: Checking IP base address in Vivado Address Editor.

When the provided code is placed instead of the Hello-World code, the project will get the appearance of next figure. To do so, follow the steps in the figure 121.

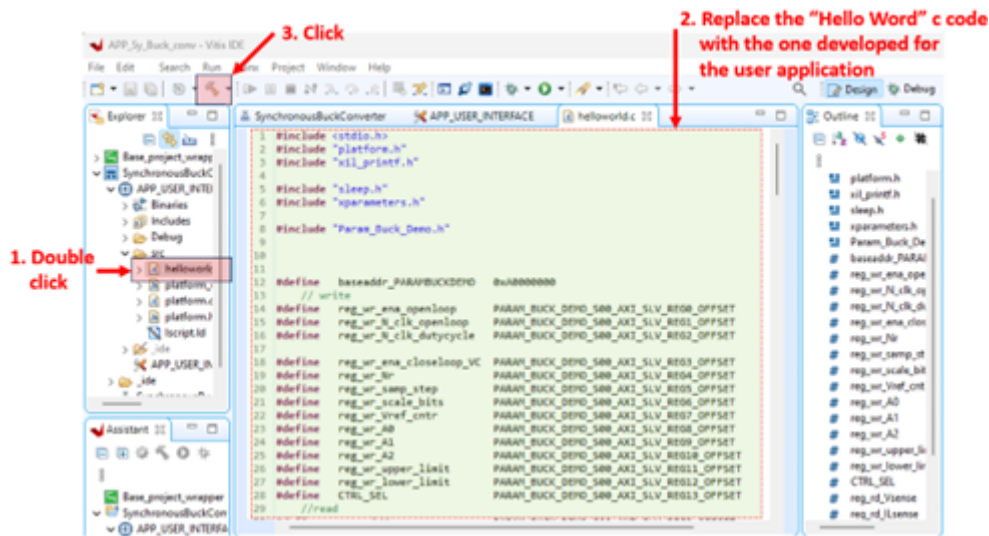


Figure 123: Update the code in Vitis IDE.

Procedure to include AXI_PSCoscilloscope in a new design: The PSCoscilloscope has two parts:

1. An IP that is included in Vivado project, called AXI_PSCoscilloscope.
2. The executable file APP_PSC_OSCILLOSCOPE_V1.elf contains the program to be run by the cortex R5_0 processor.

The way the .elf file is loaded depends on how the board is booted (debugging or by creating an SD binary file). In the following subsections, the necessary procedure is detailed.
Design debugging procedure (microUSB boot):

While in the development phase, it is usually necessary to validate design changes and correct any bugs that are identified. For this purpose, debugging mode is required.

To debug the system, the C code has been included in the Vitis project and compiled and a USB-microUSB cable has been connected between the computer and SmartRCP (connector J6). Open the console functionality in Vitis and type the following commands, see the figure 122:

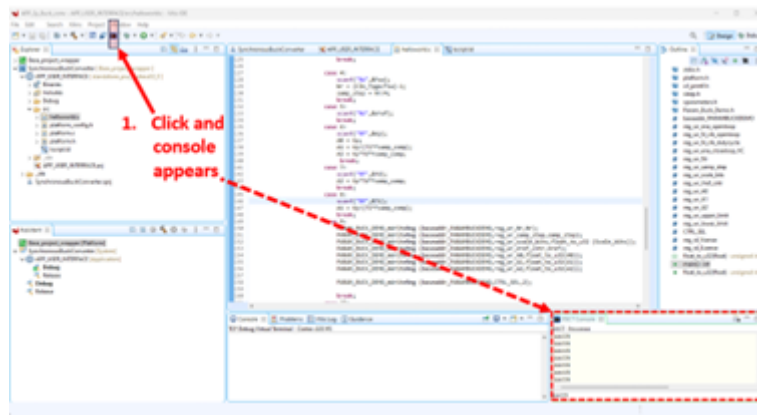


Figure 124: Open console at Vitis IDE.

- connect
- targets -set -nocase -filter name = "*PSU*"
- mwr 0xff5e0200 0x0100
- rst -system

Note: if the board shuts down or resets, it is required to retype the commands again.

Configure and launch the debugger as shown in the following figure (123):

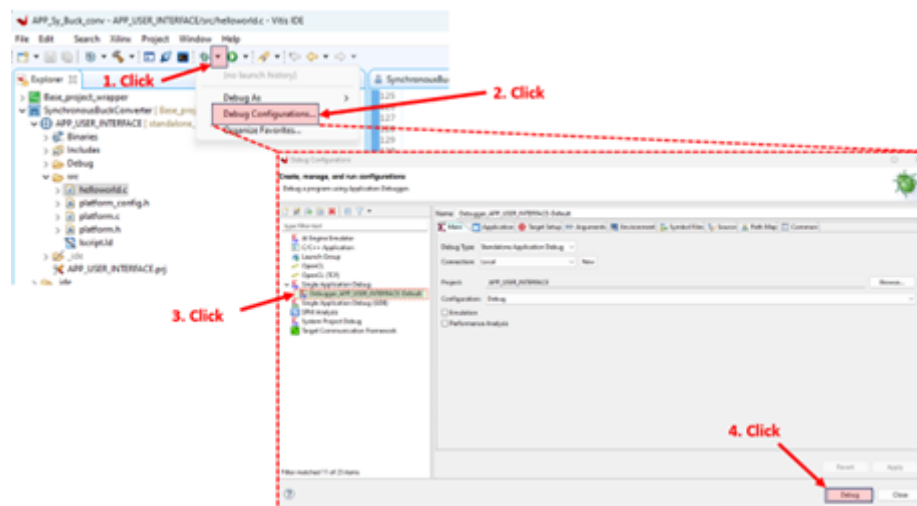


Figure 125: Launching debugger in Vitis IDE I.

Once SmartRCP is programmed it is possible to use the ILA to capture the signals that were selected during the Vivado design flow to check the correct operation of the probed IPs.

The steps required to use the ILA query are shown in the figure below, it must be accessed through Vivado:

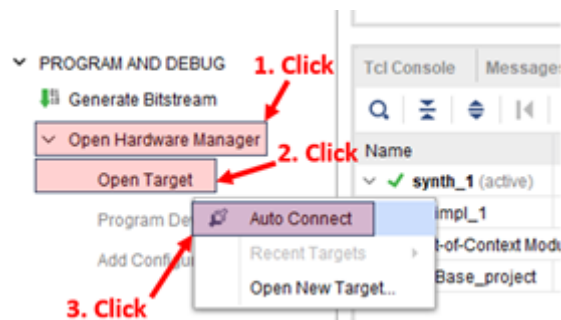


Figure 126: Accessing ILA when debugging.

A Cortex R5_0 processor is used to implement PSCoscilloscope. Some considerations must be made prior to the steps explained above:

1. Limit the memory addresses to which the cortex A53_0 processor has access to avoid data collisions between processors. See the figure 125:

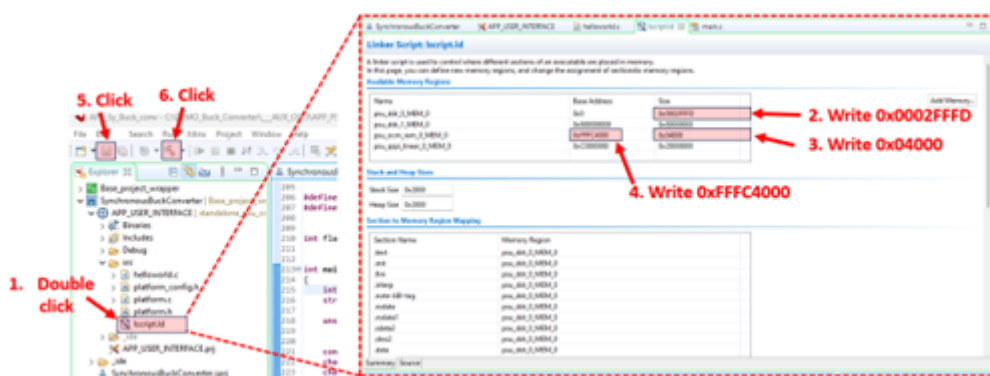


Figure 127: Change linker file in ARM A53 when PSCoscilloscope is included.

2. Include the PSCoscilloscope elf file when programming. See details in the figure 126:

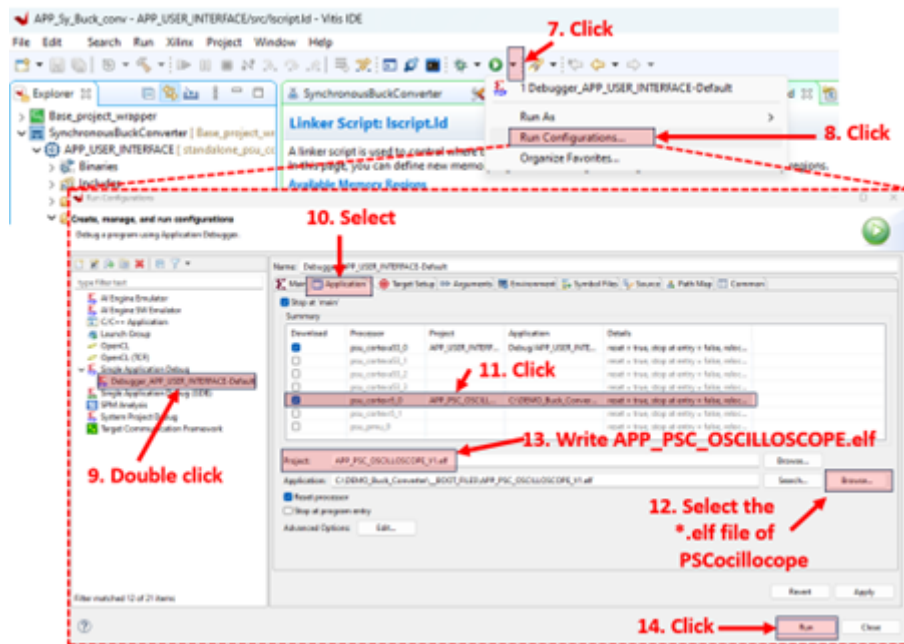


Figure 128: Include PSCocilloscope .elf file when programming or debugging.

Generation of files required for power-on via microSD:

The following procedure is used to generate a boot file that allows SmartRCP to self-program at start-up:

1. Locate the files shown in the next figure inside the project folder tree and note its path and configure Vitis as shown:

new folder			
APP_PSC_OSCILLOSCOPE_V1.elf	25/01/2024 10:17	Archivo ELF	1.167 KB
Base_project_wrapper.bit	25/01/2024 14:06	Archivo BIT	7.616 KB
APP_USER_INTERFACE.elf	25/01/2024 11:01	Archivo ELF	374 KB
fsbl.elf	24/01/2024 17:02	Archivo ELF	457 KB

Figure 129: Files required for creating booting files.

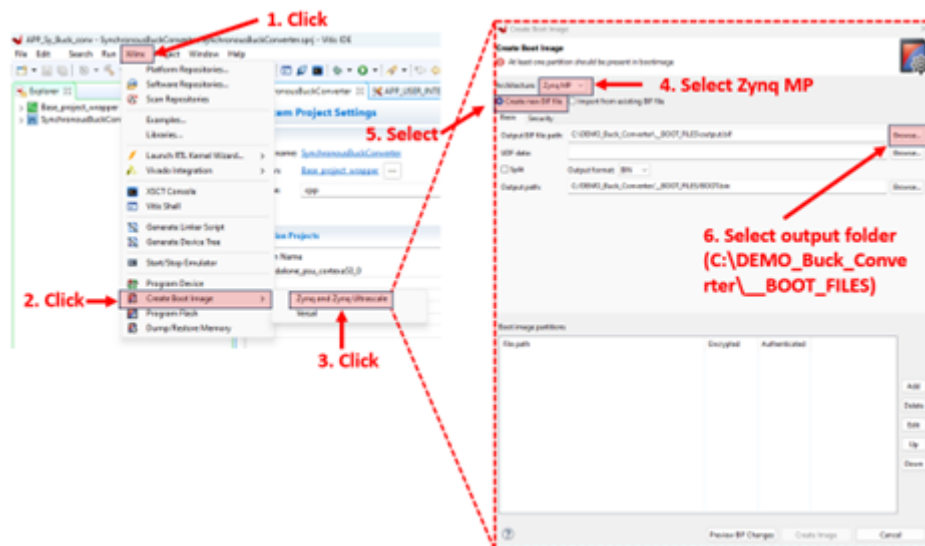


Figure 130: Vitis configuration for creating boot files I.

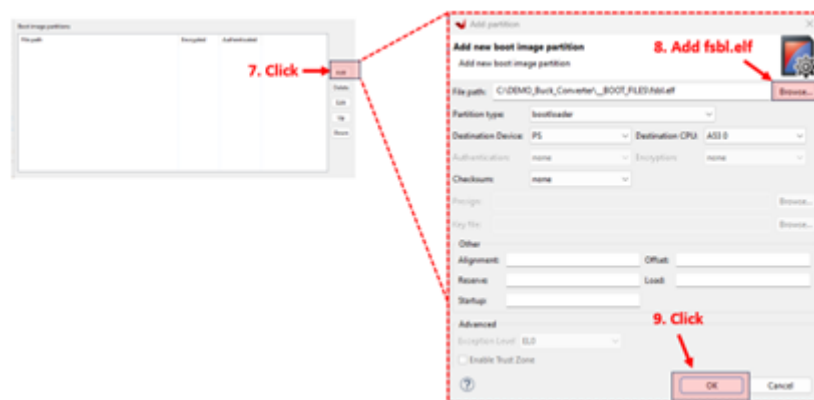


Figure 131: Vitis configuration for creating boot files II.

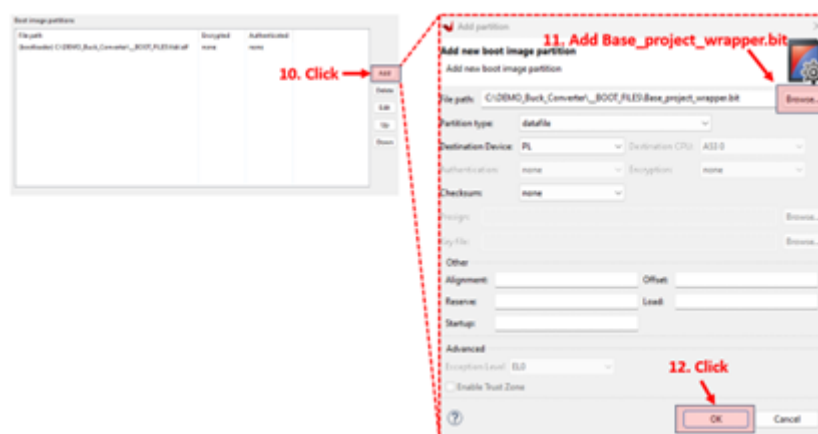


Figure 132: Vitis configuration for creating boot files III.

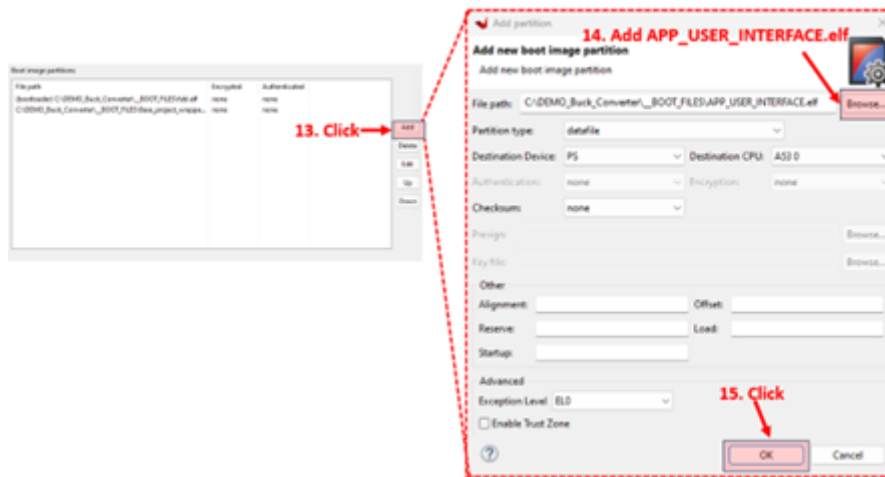


Figure 133: Vitis configuration for creating boot files IV.

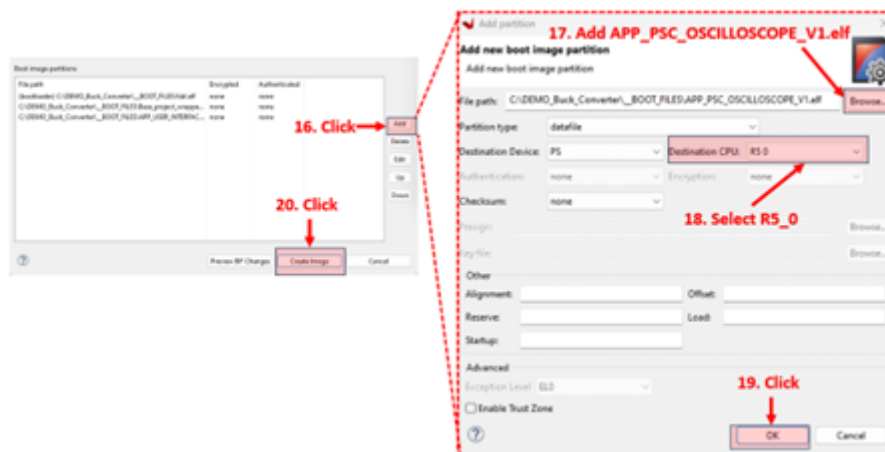


Figure 134: Vitis configuration for creating boot files V.

- After clicking OK, a file called BOOT.ini has been created. Copy that file into the microSD card and install it into SmartRCP system connector. As soon as SmartRCP is reset or powered, the system will be automatically programmed with that file.

If using PSCoscilloscope, it is required to include an extra configuration file into microSD, this file will be in charge to perform NET configuration. This file has the content shown in the figure 133:

```
<>PSCope==>
PSC_ETH_MAC="0x00, 0x0a, 0x35, 0x00, 0x00, 0x01"
PSC_ETH_AUTO_con="Y"
PSCope_IP="192.168.1.78"
PSCope_DEFAULT_IP_MASK="255.255.255.0"
PSCope_DEFAULT_GW_ADDRESS="192.168.1.1"
PSCope_cliente_status="N"
PSCope_client_IP="192.168.1.22"
```

Figure 135: RCP_conf.txt file content.

1. Device MAC: this parameter is important if several oscilloscope boards are connected to the same router as it may cause problems in IP address assignment.
2. PSC_ETH_AUTO_con parameter: allows you to choose the option to set an IP address "N" or let the router assign it automatically "Y".
3. IP address of the device: If the address is assigned manually, these parameters are used to set the necessary variables.
4. PSC_client_status parameter: if "N" is selected the oscilloscope will communicate with the first client that makes a data request, if "Y" is selected the oscilloscope will only send data to the client with the duration set below.
5. PSC_client_IP parameter: If "Y" was selected in the PSC_client_status parameter, this will be the IP address to which the oscilloscope sends the data.

8.2.5 Using the graphical interface

Graphical interface developed for this demo and all its field, and functions can be seen in figure below (134):

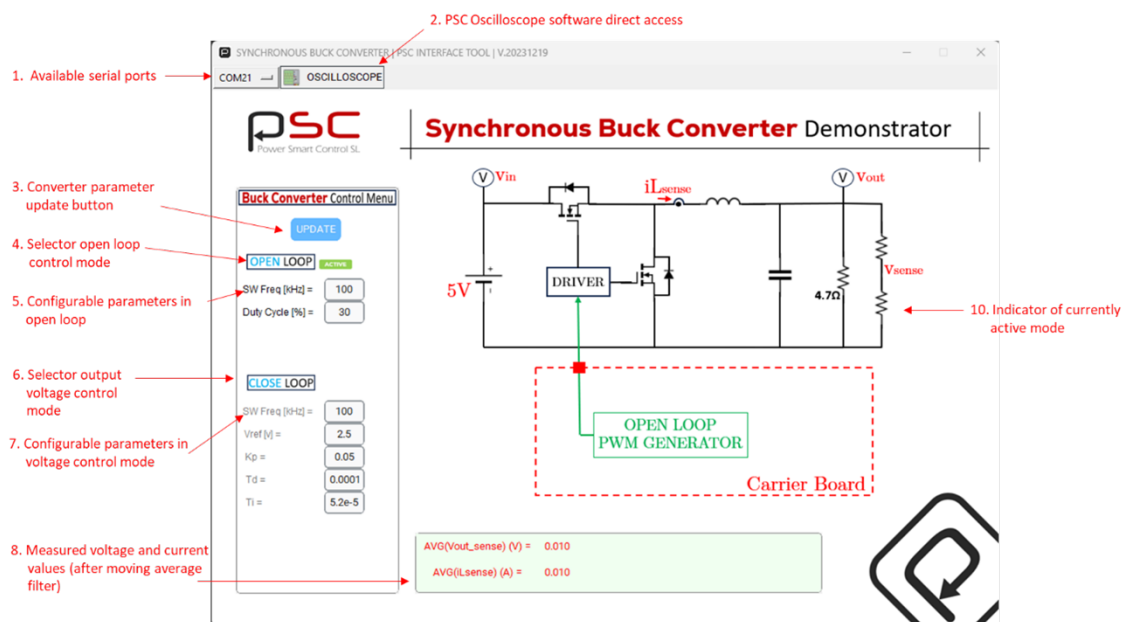


Figure 136: RCPBuckBoard graphical user interface.

- Converter parameter update button. This button must be pressed to update all changes.
- Selector open loop control mode allows to change from open loop to closed loop.
- Configurable parameters in voltage control mode. The next figure provides a view of the parameters and their nature:

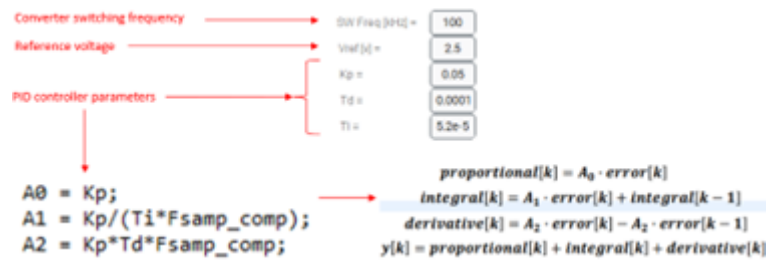


Figure 137: Compensator parameters.

How to use:

1. Before powering SmartRCP, make sure that the microSD card (formatted in FAT32) contains RCP_conf.txt and BOOT.bin files.
2. Indicate the COM port assigned to SmartRCP by the computer. The user can check it by using the Windows Device Administrator tool.
3. Changes made to any of the parameters will be effective after clicking the UPDATE button.

9 Terms of Use, Limitations, and Warranty for SmartRCP

9.1 Scope of SmartRCP (use, applications, and regulations)

SmartRCP is not intended for general consumer use and is not considered a finished end product. SmartRCP is intended solely for use by specialized technical personnel duly qualified to safely handle electrical/mechanical components, systems and subsystems, and only for R&D/prototyping tasks in appropriate laboratories/facilities (evaluation and testing purposes for educational and industrial applications). SmartRCP is not intended to comply with any normative or regulation regarding electrical safety, electromagnetic compatibility, restricted substances (RoHS), recycling (WEEE), FCC, CE, UL, or any other standard, and may not meet the technical requirements of these or other related directives.

The buyer agrees to purchase SmartRCP exclusively for the use described above, being strictly prohibited any other use/application. The buyer is prohibited from reselling, redistributing, reverse engineering and/or replicating SmartRCP, as well as using/incorporating SmartRCP as a constituent part of its own products for distribution or sale to other entities or end consumers. The buyer shall assume full liability for the final use or application of SmartRCP in accordance with the mentioned restrictions, ensuring that SmartRCP will be operated in an appropriate manner that complies with applicable safety requirements, standards and regulations. Power Smart Control S.L. shall not be liable for any damage/loss (of any nature, without limitation) arising from improper use of SmartRCP (including, but not limited to, unauthorized use, modifications, or carelessness).

Power Smart Control S.L. makes no representation or warranty with respect to the adequacy or accuracy of SmartRCP and any related documentation. Power Smart Control S.L. disclaims all representations and warranties, express or implied, as to SmartRCP and documentation, including, without limitation, any warranty of merchantability, accuracy, completeness, fitness for any particular purpose, non-infringement, or likelihood of success. In no event will Power Smart Control S.L. or its direct or indirect suppliers be liable for any damages whatsoever including, but not limited to, direct, indirect, incidental, consequential or special damages/losses (of any nature, without limitation), loss of business profits, data, business information, or any other commercial damages or losses. In no event shall Power Smart Control S.L. aggregate cumulative liability for any claims arising out of or related to SmartRCP exceed the list price paid for SmartRCP. In addition, the following aspects shall apply:

- **Changes and updates:** Power Smart Control S.L. is committed to continuous improvement, and, therefore, reserves the right to modify both SmartRCP and software components/functionalities without prior notification.
- **Accuracy of information:** Power Smart Control S.L. does not guarantee the accuracy or completeness of the information regarding SmartRCP. The buyer shall assume full liability for verifying any critical information regarding SmartRCP.

9.2 Acceptance of goods (by the buyer)

The buyer must inspect shipments within ten (10) days of receipt and report any defects, shortages, or damages in writing. Failure to report issues within this period constitutes acceptance of SmartRCP “as-is”.

- If no claims are raised within ten (10) days of receipt, the shipment will be deemed accepted by the buyer (acceptance of SmartRCP “as-is”).
- If the buyer identifies any damage or defects within ten (10) days of receipt, the buyer shall inform Power Smart Control S.L. in writing within ten (10) days of receipt, providing details (including, but not limited to, photographs and appropriate reports) and requesting Power Smart Control S.L.’s acknowledgement and validation in advance. Unless otherwise specified, the buyer shall bear the risk and cost of returning SmartRCP accordingly (please, contact Power Smart Control S.L. and your local distributor).

By using the product, the user agrees to the terms and conditions outlined in this clause of disclaimer and limitation of liability and acknowledges having read and understood the contents of this document.

9.3 Limited warranty for SmartRCP

SmartRCP is provided “as-is”, with a limited warranty period of one (1) year. The warranty period starts at the delivery date.

All deficiencies which cannot be proved to be attributable to defective original material or workmanship are explicitly excluded from the warranty. Any costs related to the demonstration of defects or faults shall be fully borne by the buyer.

If the buyer provides (in advance) a written notification and details (including, but not limited to, photographs and appropriate reports) of a defect, Power Smart Control S.L., at its sole discretion and only providing that SmartRCP was proven in advance to be defective during the warranty period and that the defect is attributable to original material or workmanship, will:

- a) Repair the defective product or replace defective components, or
- b) Provide a replacement unit if repair is not feasible, or
- c) Advise the customer on alternative solutions to achieve the intended functionality, or
- d) Issue a refund of the purchase price or a portion of it due to depreciation.

For any warranty claim, the buyer shall bear all return shipping costs for Power Smart Control S.L. inspection and validation. SmartRCP shall be shipped to POWER SMART CONTROL S.L. | AVDA. GREGORIO PECES BARBA, N° 1, LEGANÉS (MADRID, SPAIN), in a package equal to or in the original packaging. SmartRCP will be returned freight prepaid and repaired or replaced only if Power Smart Control S.L. determines that the defects can be attributable to defective materials or workmanship. Otherwise, the corresponding fees and extra costs will be charged to the buyer. Any alterations made to the

product, tampering with warranty seals, improper use, or suspicion of product modification by the buyer will result in an immediate nullification of the warranty, immediately exempting Power Smart Control S.L. from any liability related to SmartRCP. The warranty will be void specifically under, but not limited to, the scenarios where SmartRCP is:

- Operated exceeding the nominal operating conditions and specifications.
- Exposed to dust, moisture, corrosive substances, or excessive vibration.
- Stored, handled, transported, or installed improperly.
- Modified, repaired, or altered without explicit authorization of Power Smart Control S.L.
- Used with broken, tampered, or missing warranty seals.
- Overstressed for any reason (electrical, thermal, mechanical, etc...).
- Overheated due to inadequate ventilation or cooling.
- Damaged by external factors (short-circuits, electromagnetic interferences (EMI), electrostatic discharges (ESD), carelessness, etc...).
- Excessive wear and tear or improper/abusive use.

9.4 Third parties

KRIA K26, Vivado, Vitis and Vitis HLS are products developed and under the ownership of external third parties (not linked in any manner to Power Smart Control S.L.)

The information contained in this document relating to these products (KRIA K26, Vivado, Vitis and Vitis HLS) is non-official, non-binding, and only for information purposes for the user, based on the best possible understanding, knowledge, and good faith of Power Smart Control S.L.

10 Contact and Support

Power Smart Control S.L. is dedicated to assisting and supporting users in both hardware (HW) and software (SW) matters. For inquiries or assistance, please reach out to Power Smart Control S.L. via email at:

sales@powersmartcontrol.com
support@powersmartcontrol.com

References

- [1]] Kria K26 System-on-Module. <https://www.xilinx.com/products/som/kria/k26c-commercial.html>
- [2] Kria K26 SOM Data Sheet (DS987). PL Resources Table
- [3] Zynq MPSoC Book. Figure 3.2. Pag 34.

11 Annex A SensingBoard design

This section provides several guidelines and base circuitry that can be used as a template when designing a custom SensingBoard.

SensingBoard may have the number of channels required by the user and can be connected to any number of channels among the 24 possible CarrierBoard analog input channels available.

CarrierBoard provides 24 channels, the connector is a 4-wire connector which includes 2 pins for a differential signal and also 2 pins for an analog 5V power and its ground reference. Users can use this power but care must be taken with the consumption.

Maximum current consumption (counting everything connected to CarrierBoard analog input channels) must be below 0.1A.

The maximum output analog span of any SensingBoard must be between 0 and 5V. Negative or higher voltages may result in a CarrierBoard HW damage.

Four basic examples are given in this document. They are intended as a starting point and PSC has no liability in the use of those designs; they are presented as-is and without any warranty or responsibility by PSC side:

1. Designing a DC isolated voltage sensor analog chain
2. Designing an AC isolated voltage sensor analog chain
3. Designing an isolated current sensor analog chain
4. Designing a temperature sensor analog chain

Designing a temperature sensor analog chain

NTC temperature sensors are quite useful and can be found in a high variety of formats including ones which a screwable metallic termination (lug) that is electrically isolated. However, their response is non-linear and it is usually advisable to linearize it before taking the measurement.

The next figure presents a simplified schematic of an NTC sensor, linearized and with the total gain controlled by an Operational Amplifier (OpAmp):

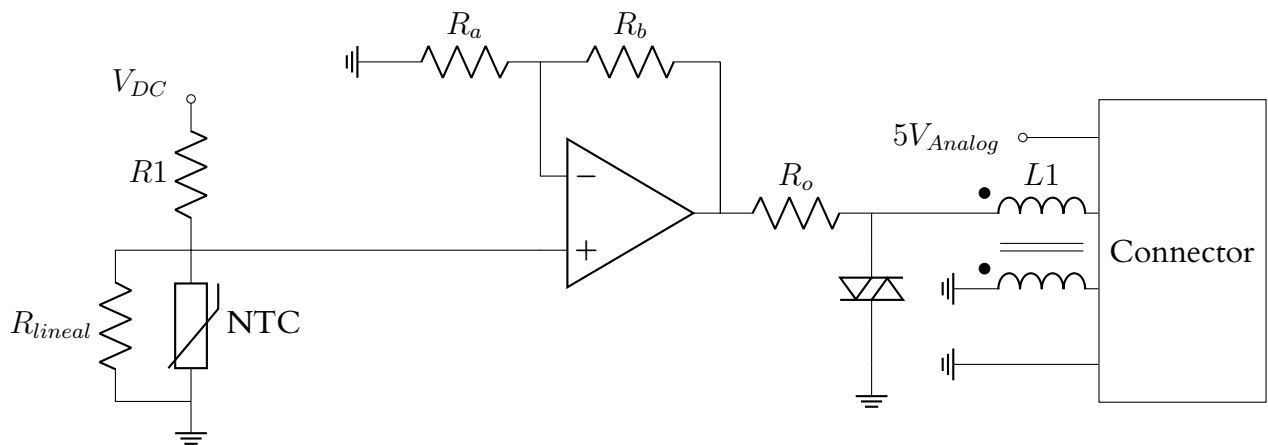


Figure 138: NTC temperature sensing analog chain

The NTC reference shall be selected from a catalog to fit the measuring range required. R_{lineal} is used to ensure the relationship between temperature and voltage drop in the NTC is linear. It should be selected so that its value equals the NTC value in the middle of the desired temperature range.

$R1$ is selected to modify the voltage range generated by the NTC- R_{lineal} system and increase it as much as possible to fill the OpAmp input span (typically 0-5V).

R_a and R_b shall be selected to adapt the output value of the chain to the input span of CarrierBoard analog input channel (0-5V). OpAmp shall be selected to be rail-to-rail, with enough slew-rate to fulfill the application and with single-supply voltage for easiness. Measuring temperature does not usually require a high-quality OpAmp.

An output TVS can be included to protect the next stage, which will be the Carrier-Board analog input. It should be selected to clamp at a voltage higher than the span and lower than the maximum rating of the next stage, a value of approximately 6V is suggested.

A common mode choke $L1$ can be included in case additional filtering is needed, since it will create a more EMI-resistant connection. It is usually recommended in case of long sensor wiring.

Application example:

- NTC model: NTCALUG01A103JA
- TVS model: SMAJ6.0A
- Operational amplifier model: OPA192
- Choke model ($L1$): Würth Elektronik 744235801
- Connector model: Molex 53375-0410
- R_{lineal} =910Ohm as $R(NTC)$ at 90°C = 910Ohm
- $R1$ = 10kOhm

- $R_a=5k\Omega$, $R_b=55k\Omega$
- $V_{out}(0^\circ C) = 4.92V$
- $V_{out}(50^\circ C) = 4.08V$
- $V_{out}(100^\circ C) = 2.28V$
- $R_o = 100\Omega$
- Power is generated by additional power supply, $5V_{ana}$ is left unconnected.
- **Maximum output analog spam must be between 0V and 5V**

Figure 137 provides a complete schematic for this sensor.

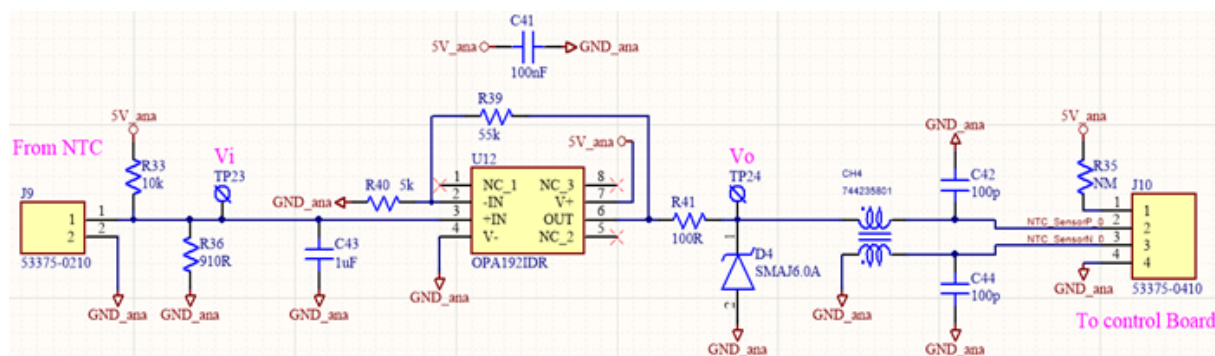


Figure 139: NTC temperature sensing schematic.

Here it has been considered the use of NTCs but there are lots of different available sensors, another highly relevant power electronic suitable sensor is the thermocouple which can be adapted to suit by making slight modifications in the presented analog chain.

In case the user needs support, they can contact PSC for consultancy services.

Designing an isolated current sensor analog chain

To measure currents there are lots of possibilities: Shunt resistors, hall sensors, LEM sensors, etc. Here it has been selected a LEM sensor to measure both AC and DC currents.

Figure 138 provides the simplified scheme for a LEM current sensing stage:

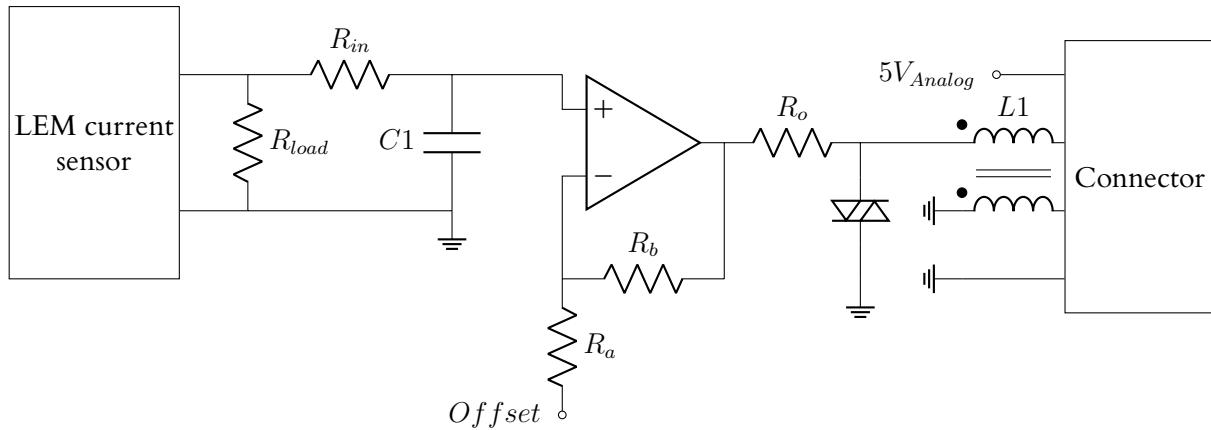


Figure 140: LEM current sensing analog chain

R_{load} and R_{in} tend to be defined in value and tolerance by the LEM sensor provider, $C1$ is set for removing extra unneeded bandwidth and avoiding external noise to be fed.

R_a and R_b shall be selected to adapt the output value of the chain to the input span of CarrierBoard analog input channel (0-5V). OpAmp shall be selected to be rail-to-rail, with enough slew-rate to fulfill the application and with single-supply voltage for easiness.

An output TVS can be included to protect the next stage, which will be the CarrierBoard analog input. It should be selected to clamp at a voltage higher than the span and lower than the maximum rating of the next stage, a value of approximately 6V is suggested.

A common mode choke $L1$ can be included in case additional filtering is needed, since it will create a more EMI-resistant connection. It is usually recommended in case of long sensor wiring.

Application example:

- LEM model: Tamura L31s
- TVS model: SMAJ6.0A
- Operational amplifier model: OPA192
- Choke model ($L1$): Würth Elektronik 744235801
- Connector model: Molex 53375-0410
- $R_{load}=10k\Omega$
- $R_{in}=22\Omega$
- $C1=10nF$
- $R_a=10k\Omega$ / $R_b=30k\Omega$ / $V_{offset}=2.5V$
- $V_{out}(-200Amp) = 0V$
- $V_{out}(0Amp) = 2.5V$

- $V_{out} (200Amp) = 5V$
- $R_o = 100\Omega$
- Power is generated by additional power supply, 5V_analog is left unconnected.
- **Maximum output analog spam must be between 0V and 5V**

Figure 139 provides a detailed view of the schematic of a NTC sensor:

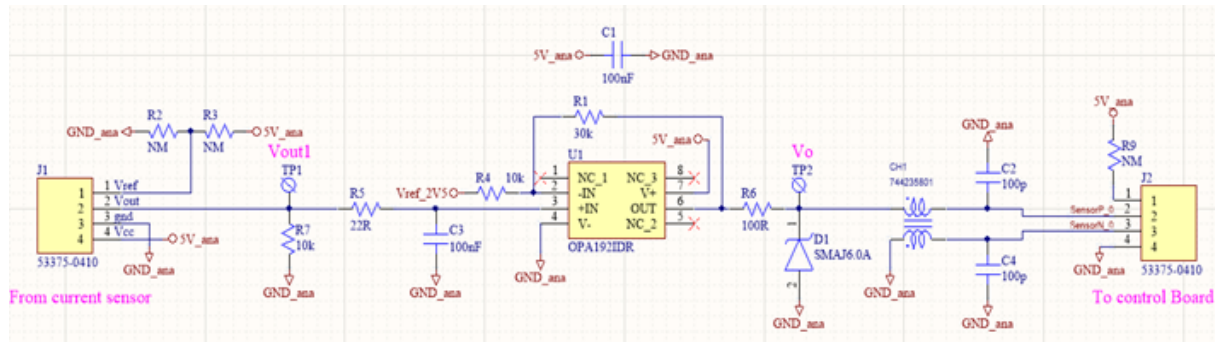


Figure 141: LEM current sensing schematic.

Designing a DC isolated voltage sensor analog chain

Typically, in power electronics applications all voltage measurements (being of a DC or AC nature) will be mandatory to be isolated from each other so no undesired ground/earth/neutral-point currents flow through the measurement stages.

To do so, an isolated voltage sensor is selected, typically a resistor with a differential isolated OpAmp, this is the case covered here. The next figure shows a simplified view of the analog chain.

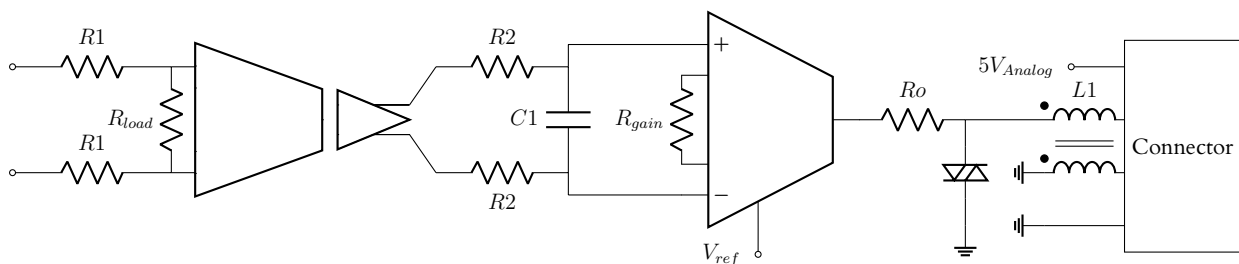


Figure 142: DC voltage sensing analog chain

R_1 and R_{load} shall be selected to drain the smallest amount of current (power) from the converter while ensuring the voltage drop on R_{load} is admissible by the isolated OpAmp input Spam. Due to dissipation or electrical security, R_1 resistors may be more than one single resistor and may need to be higher SMD packages and support higher voltage ratings than usual.

Isolated OpAmp has to be able to provide the required electrical isolation between Rload voltage and the other power supplies connected to it. It will require as well its primary side to be fed by an isolated power supply. R2 and C1 are selected to filter the output of the isolated OpAmp and to remove any possible noise.

OpAmp is required to have a differential input and its output range will be selected by setting the value of Rgain and Vref. Those values shall be selected to adapt the output value of the chain to the input span of CarrierBoard analog input channel (0-5V). Instrumentation OpAmp shall be selected to be rail-to-rail, with enough slew-rate to fulfill the application and with single-supply voltage for easiness.

An output TVS can be included to protect the next stage, which will be the CarrierBoard analog input. It should be selected to clamp at a voltage higher than the span and lower than the maximum rating of the next stage, a value of approximately 6V is suggested.

A common mode choke L1 can be included in case additional filtering is needed, since it will create a more EMI-resistant connection. It is usually recommended in case of long sensor wiring.

Application example:

- Optically isolated amplifier model: ACPL-C87A
- Input side of optically isolated amplifier is powered with an isolated PCB power supply like TBA_1-2421E
- TVS model: SMAJ6.0A
- Operational amplifier model: INA826
- Choke model (L1): Wurth Elektronik 744235801
- Connector model: Molex 53375-0410
- Measuring input range: 0-1300V
- $R1=3.28\text{M}\Omega$ (four 820K Ω resistor in serie)
- $R_{load}=10\text{k}\Omega$
- $R2=4.7\text{k}\Omega$
- $C1=10\text{nF}$
- $R_{gain}=33\text{k}\Omega$
- $V_{ref}=0\text{V}$ (ground connection)
- $R_o=100\Omega$
- Instrumentation OpAmp: INA826
- $V_o (V_{in}=0\text{V}) = 0\text{V}$

- $V_o (V_{in}=800V) = 3.02V$
- $V_o (V_{in}=1314V) = 5V$
- Power is generated by additional power supply, 5V_analog is left unconnected.
- **Maximum output analog spam must be between 0V and 5V**

Figure 141 provides a detailed view of the schematic for a DC isolated voltage sensor analog chain.

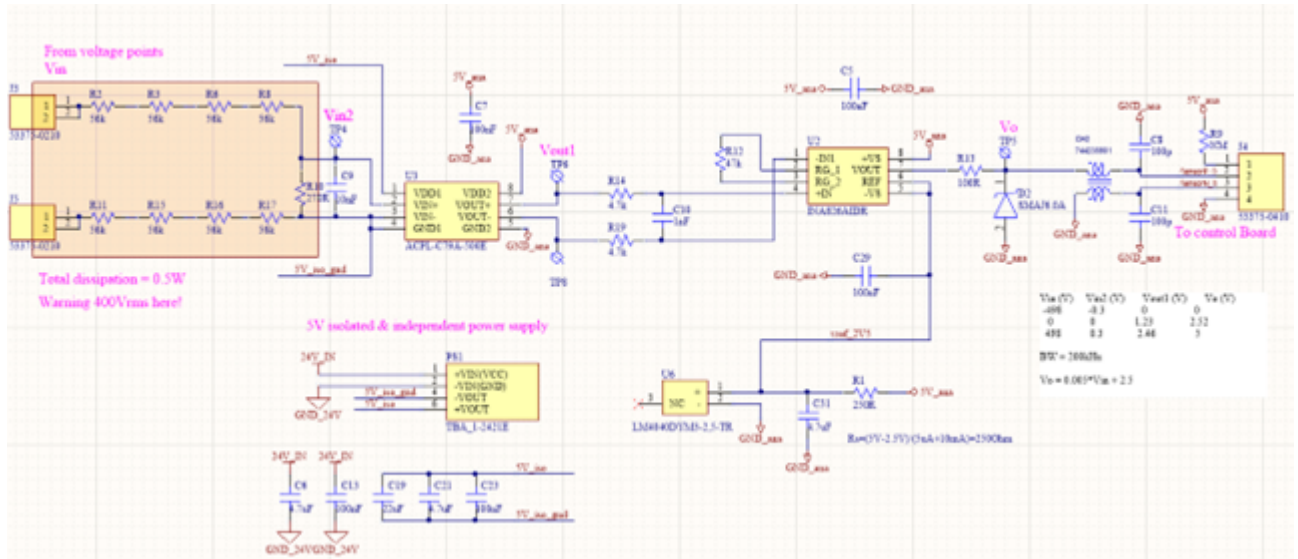


Figure 143: DC voltage sensing schematic.

Designing an AC isolated voltage sensor analog chain

Same precautions as in DC voltage sensing case shall be followed. Next figure provides a simplified schematic.

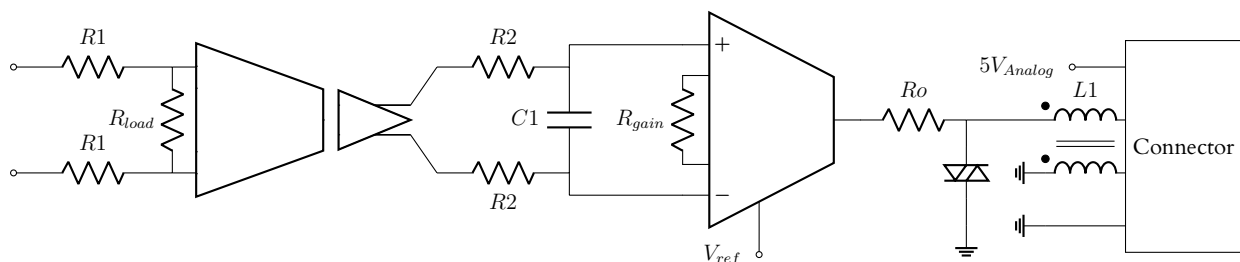


Figure 144: AC voltage sensing analog chain

R1 and Rload shall be selected to draw the smallest amount of current (power) from the converter while ensuring the voltage drop on Rload is admissible by the isolated OpAmp input Spam. Due to dissipation or electrical security, R1 resistors may be more than one

single resistor and may need to be higher SMD packages and supports higher voltage ratings than usual.

Isolated OpAmp has to be able to provide the required electrical isolation between Rload voltage and the other power supplies connected to it. It will require as well its primary side to be feeded by an isolated power supply. R2 and C1 are selected to filter the output of the isolated OpAmp and to remove any possible noise.

Instrumentation OpAmp is required to have a differential input and its output range will be selected by setting the value of Rgain and Vref. Those values shall be selected to adapt the output value of the chain to the input span of CarrierBoard analog input channel (0-5V). Instrumentation OpAmp shall be selected to be rail-to-rail, with enough slew-rate to fulfil the application and with single-supply voltage for easiness.

An output TVS can be included to protect next stage, which will be the CarrierBoard analog input. It should be selected to clamp at a voltage higher than the span and lower than the maximum rating of the next stage, a value of approximately 6V is suggested.

A common mode choke L1 can be included in case additional filtering is needed, since it will create a more EMI resistant connection. It is usually recommended in case of long sensor wiring.

Application example:

- Optically isolated amplifier model: ACPL-C79A
- Input side of optically isolated amplifier is powered with an isolated PCB power supply like TBA_1-2421E
- TVS model: SMAJ6.0A
- Operational amplifier model: INA826
- Choke model (L1): Wurth Elektronik 744235801
- Connector model: Molex 53375-0410
- Measuring input range: -400Vrms to 400Vrms
- R1=224kOhm (four 56KOhm resistor in serie)
- Rload=270Ohm
- R2=4.7kOhm
- C1=10nF
- Rgain=Not Mounted
- Vref= 2.5V (generated with LM4040)
- Ro=100Ohm

- Instrumentation OpAmp: INA826
- $V_o (V_{in}=-498V) = 0V$
- $V_o (V_{in}=0V) = 2.52V$
- $V_o (V_{in}=498V) = 5V$
- Power is generated by additional power supply, 5V_analog is left unconnected.
- **Maximum output analog spam must be between 0V and 5V**

Figure 143 provides a detailed schematic of an AC isolated voltage sensor analog chain.

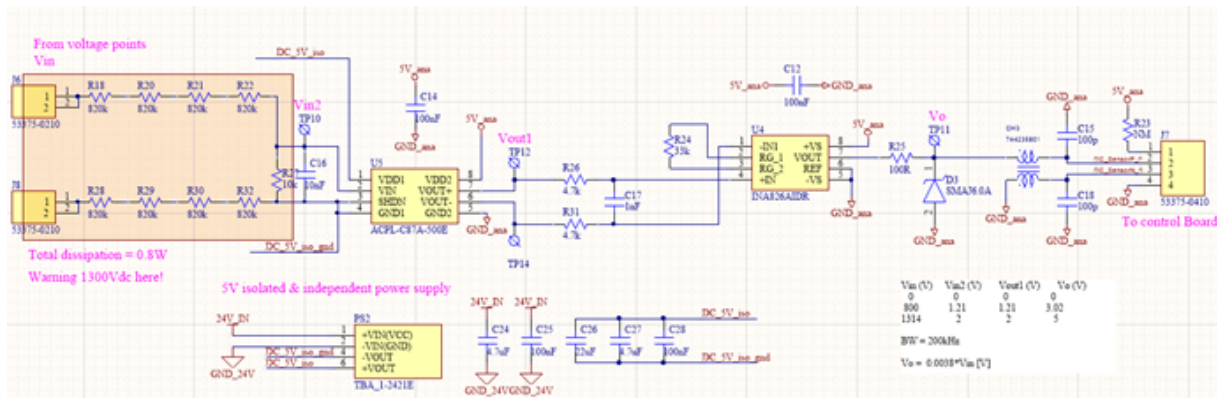


Figure 145: AC voltage sensing schematic.