Power Smart Control SL

SmartRCP
Technical manual

Power Smart Control S.L.

Version 2026.1

September 2026

Document revision history

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
2026-09-15	RVG	v2026.1 Beta: corporate template; new chapter 5 (driver IPs, incl. interrupt-timers IP); updated buck demonstrator (chapter 7); tool-version notices; figures cleaned and renumbered.



Contents

Document revision history **1**

List of acronyms **17**

1 Introduction **19**

2 Capabilities **21**

2.1 Processing power 21

2.2 Extreme tailor-made connectivity 21

2.3 Fully-programmable not black-box system 21

2.4 Networking and paralleling technology 22

2.5 Highlighted use-cases 22

3 Hardware and CarrierBoard details **24**

3.1 Mechanical design, dimensions, and weight 24

3.2 Operation requirements 24

3.3 Specification of connectivity and resources 25

3.4 Serial number and HW revision 25

3.5 KRIA K26 description and main features 26

3.6 Connector description: pin-out, function, voltages range and bandwidth 28

3.7 Constraints file 38

3.8 Ventilation 39

3.9 Booting modes 39

3.10 USB programming and debugging 39

3.11 In-depth description of functional blocks 39

3.11.1 RS-232 39

3.11.2 RS-485 40

3.11.3 User LEDs. 41
3.11.4 Ethernet ring communication 42
3.11.5 Analog output channels 42
3.11.6 Digital inputs 42
3.11.7 Digital outputs. 43
3.11.8 PWMs 44
3.11.9 Analog input channels 49
3.12 Connector shielding 51
4 Programming user guide 52
4.1 Development environment 52
4.1.1 Introduction to Vitis 52
4.1.2 Vitis download and installation instructions 53
4.2 Vivado software tools 54
4.2.1 How to create a project and the main configuration 54
4.2.2 Add an IP 57
4.2.3 Use the Constraints and create Ports 57
4.2.4 Generate Bitstream 58
4.2.5 How to use the RCP_TEMPLATE provided by PSC 59
4.3 Xilinx Vitis software tools 67
4.3.1 Introduction 67
4.3.2 Project creation in Vitis. 67
4.3.3 Procedure for loading and debugging the design on SmartRCP. 68
4.3.4 How to make multi-processor designs 70
5 Driver IPs included in the RCP_TEMPLATE 73
5.1 Vivado standard IP cores 74
5.2 Communication interfaces: RCP_Comm_Ctrl_V3 77
5.3 Digital inputs and outputs: DIGITAL_INOUT_DRIV_V1 83
5.4 PWM and 3.3 V digital outputs: RCP_PWM_DRIV_v1 87

5.5 Digital-to-analog converters: RCP_DAC_DRIV_v1 91
5.6 Analog-to-digital converters: SmartRCP_ADc_Driv 95
5.7 Write registers: OUTPUT_USER_REGISTERS 100
5.8 Read registers: INPUT_USER_REGISTERS 102
5.9 PL-DataLogger: RCP_DataLogger 104
5.10 RMS and average calculation: IP_BLOCK_RMS_AVG_v1 108
5.11 General-purpose interrupt timers: RCP_8_iNTERRUT_TIMERS 110
5.12 Integrated Logic Analyzer (ILA) 114

6 SmartRCP Test Mode 115

7 Synchronous Buck Converter demonstrator. 116

7.1 Hardware 116
7.2 Workflow for customizing the RCP_TEMPLATE. 120
7.3 Developing IPs for Vivado: C code and Vitis HLS. 122
7.3.1 Project introduction and functional blocks 122
7.3.2 IP generation with Vitis HLS: the DEMO_BUCK_PWM block 123
7.3.3 The DEMO_BUCK_PID block. 132
7.3.4 The DEMO_BUCK_DATA_PROCCESSING block. 134
7.4 RCP_TEMPLATE customization: adding and interconnecting user IPs 136
7.4.1 Project setup and renaming of the Vivado template 137
7.4.2 Introduction to the Vivado workspace environment. 137
7.4.3 Integrating user-developed IPs from Vitis HLS 141
7.4.4 Adding the ILA IP block for design debugging 149
7.4.5 Summary and final user considerations 152
7.5 Generating the bitstream: synthesis and implementation 157
7.6 RCP_TEMPLATE customization: configurations in Vitis IDE 159
7.6.1 Exporting files from Vivado to Vitis IDE. 159
7.6.2 Creating a new project in Vitis IDE 160
7.6.3 Generating a basic “Hello World” application 161

7.6.4 Considerations for parallel applications on multiple cores 163
7.7 Configuring the SmartRCP device: microSD and JTAG 167
7.7.1 Configuring the SmartRCP device via JTAG connection 167
7.7.2 Generating the boot file for the microSD card 170
7.8 Customization of the SmartRCP software interface. 173
7.9 Final results: control of the buck converter. 178
7.9.1 List of required materials and equipment connections. 178
7.9.2 Guided test for functionality verification 180
8 Terms of Use, Limitations and Warranty 189
8.1 Scope of SmartRCP (use, applications, and regulations) 189
8.2 Acceptance of goods (by the buyer) 190
8.3 Limited warranty for SmartRCP 190
8.4 Third parties. 191
9 Contact and Support 192
A SensingBoard design. 193
A.1 Designing a temperature sensor analog chain 193
A.2 Designing an isolated current sensor analog chain 195
A.3 Designing a DC isolated voltage sensor analog chain. 197
A.4 Designing an AC isolated voltage sensor analog chain. 199
References 202

List of figures

1 KRIA SOM module. 19

2 Networking main topologies. 22

3 SmartRCP main dimensions. 24

4 Serial QR codes and HW revision information. 26

5 KRIA K26 SOM comparison among other Xilinx devices. 26

6 Programmable-logic resources of the KRIA K26 SOM. 27

7 KRIA K26 main components and architecture. 27

8 Physical placement of the connectors on the CarrierBoard: communication side, top view and PWM side. 29

9 RS232 functional block chain (J15). 40

10 RS232 functional block chain (J16). 40

11 RS485 functional block chain (J10). 41

12 RS485 functional block chain (J11). 41

13 SmartRCP user LED position. 41

14 User LEDs functional block chain. 42

15 Ethernet ring functional block chain. 42

16 Analog outputs functional block chain. 43

17 0–24V Digital inputs functional block chain. 44

18 0–24V Digital outputs functional block chain. 45

19 Additional 0–3.3V functional block chain. 45

20 PWM functional block chain (J29 and J31). 45

21 PWM functional block chain (J33 and J35). 46

22 PWM functional block chain (J37 and J39). 46

23 PWM functional block chain (J41). 46

24 PWM functional block chain (J30 and J32). 46
25 PWM functional block chain (J34 and J36). 47
26 PWM functional block chain (J38 and J40). 47
27 PWM functional block chain (J42 and J44). 47
28 PWM functional block chain (J43). 47
29 PWM functional block chain. Enable signal detail. 48
30 PWM channels RJ45 connector pin-out. 48
31 Analog input channels front end. 49
32 Slow analog input functional block chain. 50
33 Quick analog input functional block chain. 51
34 Vitis installing options. 53
35 Web Installer entry on the AMD download page (screenshot of an earlier release; pick the 2023.2 entry). 53
36 Vivado project options. 55
37 Vivado project part definition. 55
38 Vivado project option configuration. 56
39 Incremental synthesis option in the Synthesis settings. 56
40 Implementation settings: optimization options to disable. 56
41 Designer Assistance banner with the Run Block Automation link. . . . 56
42 IP repository in Vivado project. 57
43 Creating an external port. 58
44 Changing name to an external port. 58
45 Creating HDL wrapper. 59
46 Generating output products. 59
47 Generate Output Products dialog with the settings to check. 60
48 Exporting HW. 60
49 Exporting HW with bitstream. 61
50 Vivado Project folder structure. 62

51 Contents of the RCP_TEST_uSD_FILES folder: the test-tool executable and the files to copy to the µSD card. 62

52 Contents of the RCP_VITIS_APP_USER_INTERFACE folder: the files to copy to the µSD card and the `.elf` application to run from Vitis. Screenshot of an earlier template; the current one names the folder `ADD_VITIS_A53_0` and the file `APP_RCP_USER_INTERFACE_A53_0.elf`, for core `psu_cortexa53_0` (chapter 5). 63

53 Unzipping the RCP_TEMPLATE file and renaming the folder to USER_GUIDE_EX. 64

54 Opening RCP_TEMPLATE.xpr from the RCP_VIVADO folder. 64

55 IP Integrator view of the RCP_TEMPLATE Vivado project. 65

56 Default MPSoC peripheral configuration (I/O unit of the Zynq UltraScale+ MPSoC). 66

57 Opening the Zynq UltraScale+ MPSoC configuration window from the Block Diagram. 67

58 Vitis architecture definition. 68

59 Vitis application names and resources. 69

60 Vitis OS or standalone mode selection. 69

61 Creating boot image in Vitis. 70

62 Adding additional application project in Vitis. 71

63 New processor added to the design in Vitis. 71

64 Selecting new processor in Vitis. 71

65 Linker memory positions in Vitis. 72

66 DDR memory available in KRIA K26. 72

67 The `VIVADO_STANDAR_IP_CORES` hierarchy of the template and its contents: Zynq UltraScale+ MPSoC (1), AXI Interconnect (2) and Processor System Reset (3), with its `peripheral_reset` (3.1) and `peripheral_aresetn` (3.2) outputs. 74

68 PS-PL configuration page of the Zynq UltraScale+ MPSoC block, where the AXI master interfaces from the processing system to the programmable logic are enabled. `AXI HPM0 LPD` is the one the template's driver IPs use; the screenshot predates the current template, in which `AXI HPM0 FPD` is also enabled and left free for the user. 76

69 The **Run Connection Automation** banner Vivado shows after a new IP block is added to the block design. 76

70 The RCP_Comm_Ctrl_V3 communications driver. Data ports are grouped by interface: Ethernet Ring, LEDs, RS-485 and UART; the PORT_* signals at the bottom are the physical ports. 77

71 Ethernet Ring: J18 receives and J17 transmits, and each connector carries two channels; the RJ-45 detail shows the pins of CH1 (1 and 2) and CH2 (7 and 8). 79

72 The two user LEDs on the front of the SmartRCP and the data input that drives each one; '1' switches the LED on. 79

73 RS-485 interfaces J10 and J11 and their data ports; the J11 detail marks the transmit and receive pairs on the connector. 80

74 UART interfaces over USB, UART1 on J15 and UART2 on J16, and the transmit and receive port of each one. 81

75 The DIGITAL_INOUT_DRIV_V1 digital I/O driver. On the left, the 32 data inputs that drive the board's outputs; on the right, the 32 data outputs carrying the state of its inputs and the status array DATA_DIGITAL_IN_OO_ARRAY. 83

76 Pin headers of the 32 digital inputs (left) and 32 digital outputs (right): each header carries four channels, and the channel number (*ID*) and header label are the ones that appear in the port names. 85

77 The RCP_PWM_DRIV_v1 PWM and 3.3 V digital output driver. From top to bottom on the left: clock, the 3.3 V output inputs, the PWM inputs and the physical ports; on the right, the fault outputs and the remaining physical ports. 87

78 The 14 auxiliary 3.3 V digital outputs: the channel number (*ID*) and the pin and header of each one appear in the port name; J55 also provides 3.3 V and ground. 88

79 Control interface of one PWM port: the port names carry the channel and the RJ-45 pins, and the 18 ports are located on the top of the unit; the pin order of the RJ-45 is CH1 on pins 1–2, enable on 3, CH2 on 4–5, error on 6, 5 V on 7 and ground on 8. 89

80 The RCP_DAC_DRIV_v1 digital-to-analog driver: 16 data inputs, one per channel, the physical ports on the right and a detail of the board's analog output connector. 92

81 The 16 analog outputs on the headers J64, J65, J66 and J54, and the driver input of each one; the channel number (*ID*) and the pin and header appear in the port name. 93

82 The `SmartRCP_ADc_Driv` analog-to-digital driver: physical ports and control signals on the left; on the right, from top to bottom, the general alarm, the fast channels with their alarm and identifier, the slow channels with theirs, and the remaining physical ports. 95

83 ADC channels on the board (fast on the left, slow on the right), the pinout of each input header — ground, signal −, signal +, 5 V — and the three outputs every channel offers for the same reading: raw, filtered and scaled. 97

84 Functional blocks of the ADC driver for one channel — conversion, scaling, low-pass filter and alarm — and the parameters of each one that are set from the SmartRCP Interface. 98

85 The `OUTPUT_USER_REGISTERS` write register block. The arrow gives the direction of travel, from the computer into the SmartRCP: this is the block the remote application presents as *Input-Register*. 100

86 The `INPUT_USER_REGISTERS` read register block, with the arrow pointing the opposite way to the previous one: this is the block the remote application presents as *RCP Output-Registers*. 103

87 The `RCP_DataLogger` block. On the left, from top to bottom: the `S00_AXI` interface, the `RST` reset, the 32 floating-point probes `SIGNAL_INPUT_01` to `SIGNAL_INPUT_32`, and `ap_rst` and `clk`; on the right, the debugging outputs `ENABLE`, `Pause` and `signal_Sychronism`, whose name is reproduced exactly as it appears in the template. 105

88 The `RCP_DataLogger` entry in Vivado's **Address Editor**, at `0x8000_1000` with a 4 K range. 106

89 The `IP_BLOCK_RMS_AVG_v1` RMS and average block: 24 floating-point inputs on the left, 24 average and 24 RMS outputs on the right. 108

90 The `RCP_8_iNTERRUT_TIMERS` block: eight AXI interfaces, one per timer, clock and reset on the left, and the 8-bit interrupt bus `PL_PS_IRQ0[7:0]` on the right. 111

91 `RCP_Test_Tool` Windows application main window. 115

92 Buck Converter topology included in RCPBuckBoard 117

93 RCPBuckBoard. 118

94 Connection among RCPBuckBoard and SmartRCP. 118

95 RCPBuckBoard schematic. 119

96 RCPBuckBoard connection to a desktop oscilloscope: PWM, output voltage and sensing voltage. 120

97 RCPBuckBoard connection to a desktop oscilloscope: PWM pulses of both transistors. 120

98 Recommended workflow for customizing the RCP_TEMPLATE to a new project. The loops on the right return to stage 3 when a connection is wrong and to stage 2 when an IP block has to be corrected. 121

99 Functional blocks of the buck converter control: user IPs (PWM, PID, data processing, multiplexer) and the SmartRCP drivers they connect to. 123

100 Sawtooth carrier, modulating signal and the resulting PWM signal. . . 124

101 C++ source of the DEMO_BUCK_PWM block: interface, internal state variables, carrier generation, comparison and output assignment. . . . 125

102 Vitis HLS project creation, steps 1 to 7: launching Vitis HLS, naming the project and its location, and skipping the design and testbench file pages. 126

103 Vitis HLS project creation, steps 8 to 15: clock period of the RCP_TEMPLATE main clock, selection of the xck26 part and the new project workspace. 127

104 Source file creation, steps 1 to 6: adding a new source file to the project and naming it with the `.cpp` extension. 128

105 Source file creation, steps 7 to 15: pasting the descriptive code, saving, and selecting the top function in the project settings. 128

106 Interface directive, steps 1 to 7: selecting all the interface parameters in the Directive tab and choosing the INTERFACE directive. 130

107 Interface directive, steps 8 to 12: mode `ap_none`, register option set to `true`, and the resulting directive line under each port. 131

108 C synthesis, steps 1 and 2: the Run C Synthesis button and the solution dialog, with the clock period and the selected part. 131

109 Synthesis summary of DEMO_BUCK_PWM: latency for a 200 MHz clock and FPGA resources used. 132

110 Implementation, steps 1 to 3: the Implementation entry of the run menu and the Run Implementation dialog. 132



111 Implementation report of DEMO_BUCK_PWM: resource usage and final timing. 133

112 The DEMO_BUCK_PWM block in Vivado: AP_CTRL_HS interface, user-defined inputs and user-defined outputs. 133

113 The `ap_ctrl_hs` interface: meaning of `ap_start`, `ap_idle`, `ap_ready` and `ap_done`, and their timing over one execution. 134

114 Block diagram of the DEMO_BUCK_PID code: proportional, integral and derivative terms with coefficients A0, A1 and A2, and the output limiter. 134

115 The DEMO_BUCK_PID block in Vivado. 135

116 DEMO_BUCK_PID: latency for a 200 MHz clock and FPGA resources used. 135

117 The DEMO_BUCK_DATA_PROCC block in Vivado. 136

118 DEMO_BUCK_DATA_PROCCESSING: latency for a 200 MHz clock and FPGA resources used. 136

119 Project setup, steps 1 to 6: unzipping the RCP_TEMPLATE under the new name and copying the IP_HLS_SRC folder into it. 138

120 Project setup, step 7: the two ways of opening the project, from the file explorer or from the Vivado Quick Start page. 138

121 The Vivado window with the RCP_TEMPLATE open: Flow Navigator, Workspace, Output section, Sources and Properties. 139

122 Adding the IP repository, steps 1 to 5: IP Catalog, Add Repository and selection of the IP_HLS_SRC directory. 141

123 Adding the IP repository, step 6: the three IPs detected in the directory. 142

124 Adding the IPs to the block design, steps 1 to 6: Open Block Design, the add-IP button and the search for the three Vitis HLS IPs. 142

125 The three user IP blocks after being added to the block design. 143

126 Connection automation, steps 1 to 3: the clock and reset of the three blocks connected automatically. 143

127 Connections between the write registers driver and the DEMO_BUCK_DATA_PROCC block. 144

128 Connections from DEMO_BUCK_DATA_PROCC to the PID and PWM blocks. 145



129 The filtered output of ADC QUICK 1 connected to the `data_in` input of the PID block. 145

130 Creating a VHDL source, steps 1 to 8: Add Sources, Create File with the name MUX_32bit, and the Define Module dialog. 146

131 Creating a VHDL source, steps 9 to 12: the empty entity opened from the Sources area, the multiplexer code, save, and return to the block design. 147

132 Connections of the MUX_32bit block: open-loop reference from the data processing block, closed-loop reference from the PID, selector from the write registers and output to the PWM block. 147

133 Connections of the AND_OP block: USER_ENABLE from the write registers, ALARM_GENERAL from the ADC driver, output to the enable inputs of the user blocks. 148

134 Connecting a constant to unused inputs, steps 1 to 8: adding a Constant block, setting its width (16 bits for the DAC) and value (0), and connecting it to the unused DAC inputs. 150

135 Adding the ILA, steps 1 to 10: search and configuration of the ILA (8 probes, 1024 samples), connection automation of its clock and connection of the probes to the signals to be debugged. 151

136 Creating a hierarchy, steps 1 to 6: area selection of the user blocks, Create Hierarchy in the context menu and the name of the new cell. 152

137 The block design before and after grouping the user logic into the USER_DEVELOPMENTS_DEMO_BUCK hierarchy. 153

138 The two ways of viewing the contents of a hierarchy: expanded in place, or opened in a new workspace. 154

139 Bitstream generation, steps 1 to 8: Validate Design, Create HDL Wrapper and Generate Output Products from the Sources area. 157

140 Bitstream generation, steps 9 and 10: progress of the output product generation, then Generate Bitstream and the Launch Runs dialog. . . . 158

141 Bitstream generation: progress of the synthesis and implementation runs in the Design Runs tab. 158

142 Bitstream generation, step 11: the message that confirms that the bitstream has been generated. 159

143 Export Hardware, steps 1 to 10: File, Export Hardware, Include bitstream, and the name and folder of the `.xsa` file. 160



144 Vitis IDE project, steps 1 to 8: launching Vitis IDE with the new workspace, Create Application Project and the platform created from the exported `.xsa`. 161

145 Vitis IDE project, steps 9 to 12: application and system project names and the selection of the target core. 162

146 Vitis IDE project, steps 13 to 15: domain, the Hello World template and the resulting development environment. 162

147 Hello World, steps 1 to 4: the platform and application folders in the Project Explorer, and the `src` folder with the C sources and the linker script. 163

148 Hello World, steps 5 to 8: the modified `helloworld.c`, save, and the build button of the tool area. 163

149 Core allocation of the demonstrator: the Hello World application on A53_0 and the interface application on R5_0. 165

150 Memory regions consumed by the application that communicates with the software interface, and their place in the address map. 166

151 Memory configuration, steps 1 to 5: the linker script `lscript.ld`, the base address and size of the DDR and OCM regions, save and build. . 167

152 Connections for JTAG programming: USB cable to the computer, microSD card inserted, and the Ethernet connection either direct or through a router. 168

153 JTAG programming, steps 3 to 5: opening the XSCT console, the four commands typed one by one, and the console output when the reconfiguration fails or succeeds. 168

154 JTAG programming, steps 6 to 16: Run Configurations, Single Application Debug, and the Application tab where the interface application is assigned to core R5_0. 169

155 Run configuration with only the interface application selected, on core R5_0. 170

156 XSCT console after a successful download: the A53 and R5 targets report Running. 170

157 Boot image, steps 1 to 5: Create Boot Image from the system project, the output folders and the partition list. 171

158 Boot image, steps 6 to 9: adding the interface application partition for core R5_0 and creating the image with the four files. 172



159 Interface customization, steps 1 to 4: the ADC configuration window with the two ADC QUICK channels and the Update button. 174

160 Interface customization, steps 5 to 11: enabling registers 1 to 11 with their data types, then the names and initial values of the write registers. 175

161 Interface customization, steps 12 to 14: the read registers enabled and named after the signals of Table 31. 176

162 Interface customization, steps 15 to 18: the PL-DataLogger channel names, consistent with Table 30. 176

163 Interface customization, steps 19 to 23: probe selection and time step of the PL-DataLogger. 177

164 Saving (top, steps 1 to 4) and loading (bottom, steps 1 to 5) an interface configuration: the save button and the file dialog; the COM configuration window, the Load button and the selection of the file. 177

165 Contents of the SmartRCP_Demo_Buck archive: the RCP_TEMPLATE default files and the files added by the buck converter demonstrator. . 179

166 Test setup: power supply, Ethernet link to the computer, USB cable for the ILA test, sensing cables, PWM trigger cable, USB power of the converter and the optional oscilloscope on the test points. 180

167 Direct connection, steps 1 to 4: the `PSC_TCP.txt` file with the manual IP configuration, copied with `BOOT.bin` to the microSD card and inserted into the SmartRCP. 181

168 Direct connection, steps 7 to 10: the COM configuration window, the IP address of the SmartRCP, loading the saved configuration and Update. 182

169 Guided test, steps 13 and 14: Connect and Enable, and the values that appear in the ADC readouts and the read registers once the converter is running. 182

170 Guided test, steps 15 and 16: the write registers window; the effect of the loop mode and reference changes on voltage and current. 183

171 Guided test, steps 17 to 19: register values of the test, Update and the Scope button. 183

172 Guided test, steps 19 to 21: PL-DataLogger configuration and the real-time capture of the filtered and unfiltered current, the filtered output voltage and the PWM carrier. 184



173 Inductor current measured by the PL-DataLogger and by a bench oscilloscope in open loop: top, 50 kHz and 32 % duty cycle; bottom, 100 kHz and 75 % duty cycle. 184

174 Alarm test, steps 1 to 3: current limit of ADC QUICK 2 set to 3.5 A and its filter to 5 kHz, then Update. 185

175 Alarm test, steps 4 to 6: PL-DataLogger channels for the current limit, the unfiltered and filtered current and the output voltage, while the closed-loop reference is raised. 185

176 Alarm test: current and voltage for reference voltages of 2, 3, 4 and 4.25 V; at 4.25 V the alarm trips, the PWM stops and voltage and current drop to zero. 186

177 Alarm test: the interface with the alarm active, its source in the alarm log, and the voltage and current recorded during the test. 186

178 ILA, steps 1 to 5: Open Hardware Manager, Open Target and Auto Connect. 187

179 The Hardware Manager: unique identifier of the SmartRCP, the captured-signal viewer, the capture controls and the trigger setup. . . . 187

180 ILA, step 6: a capture of the eight probes of the demonstrator. 188

181 NTC temperature sensing analog chain. 194

182 NTC temperature sensing schematic. 195

183 LEM current sensing analog chain. 196

184 LEM current sensing schematic. 197

185 DC voltage sensing analog chain. 197

186 DC voltage sensing schematic. 199

187 AC voltage sensing analog chain. 199

188 AC voltage sensing schematic. 201



List of acronyms

Acronym		Meaning
ADC	–	Analog to Digital converter
APP	–	Application
AVG	–	Average
AXI	–	Advanced eXtensible Interface
CAN	–	Controller Area Network
DAC	–	Digital to Analog Converter
DDR	–	Double Data Rate
DHCP	–	Dynamic Host Configuration Protocol
DSP	–	Digital Signal Processing (slice of the FPGA)
DT	–	Digital Twin
ELF	–	Executable and Linkable Format (`.elf`, a compiled program)
ENA	–	Enable
ERR	–	Error
ETH	–	Ethernet
FPD, LPD	–	Full and Low Power Domain of the MPSoC (`AXI HPM0 FPD`, `AXI HPM0 LPD`)
FPGA	–	Field Programmable Gate Arrays
FSBL	–	First Stage Boot Loader
FW	–	Firmware
GUI	–	Graphical User Interface
HDL	–	Hardware Description Language
HIL	–	Hardware in The Loop
HLS	–	High-Level Synthesis
HW	–	Hardware
I2C	–	Inter-Integrated Circuit bus
I/O	–	Input and Output
IDE	–	Integrated Development Environment
ILA	–	Integrated Logic Analyzer
IP	–	Intellectual Property (a packaged block of logic); also Internet Protocol, in addresses
IRQ	–	Interrupt Request
JTAG	–	Joint Test Action Group
kSPS	–	kilosamples per second
LED	–	Light Emitting Diode



Acronym		Meaning
LSB, MSB	–	Least and Most Significant Bit
LUT	–	Look-Up Table
MPSoC	–	Multiprocessor System on Chip
MUX	–	Multiplexer
OA	–	Operational Amplifier
OCM	–	On-Chip Memory
PCB	–	Printed Circuit Board
PID	–	Proportional-Integral-Derivative (controller)
PL	–	Programmable Logic
PL-DataLogger	–	Programmable Logic DataLogger, the embedded oscilloscope IP block (`RCP_DataLogger`)
PS	–	Processing System
PSC	–	Power Smart Control SL
PWM	–	Pulse Width Modulation
RCP	–	Rapid Control Prototyping
RMS	–	Root Mean Square
RTS	–	Real Time Simulator
SDK	–	Software Development Kit
SoC	–	System on Chip
SOM	–	System on Module
SPI	–	Serial Peripheral Interface
SW	–	Software
TCP/IP	–	Transmission Control Protocol / Internet Protocol
TVS	–	Transient Voltage Suppressor (diode)
UART	–	Universal Asynchronous Receiver Transmitter
URAM	–	UltraRAM, the large memory blocks of the FPGA
USB	–	Universal Serial Bus
VHDL	–	VHSIC Hardware Description Language
XSA	–	Xilinx Support Archive, the hardware platform file exported by Vivado
XSCT	–	Xilinx Software Command-line Tool



1 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

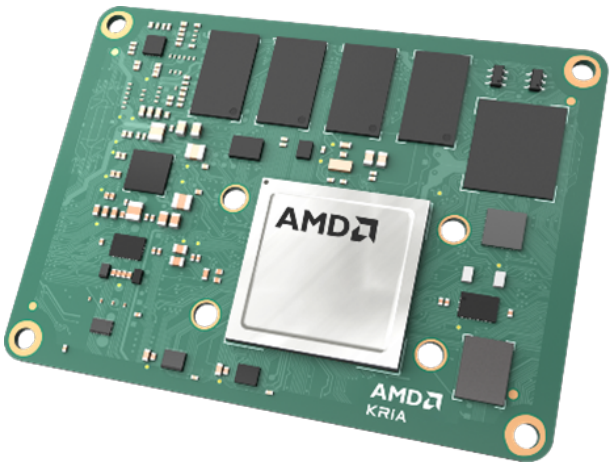


Figure 1: KRIA SOM module [1].

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 adaptions 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 1 Gbit/s Ethernet connector and an enormous powerful core it is also suited for AI applications related with converter controlling such as end-of-life estimation, predictive maintenance, etc.

By the use of the RCP_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.

2 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.

2.1 Processing power

The 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 and a rich set of configurable I/O capabilities.

Those capabilities make it possible to obtain PWM channels with 5ns time resolution, provides more than 200 user I/Os and integrates 6 high-end ARM microcontrollers.

2.2 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 1 Gbit/s Ethernet, 4x USB2.0
- 2x UART 2x RS-485 & 1x microSD

2.3 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.

2.4 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 5ns.

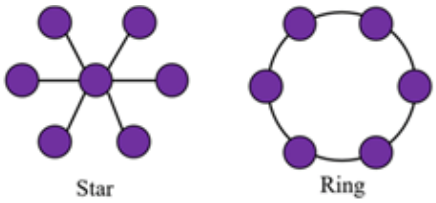


Figure 2: Networking main topologies.

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 5ns resolution channels, perfect synchronization of all the devices connected.

SmartRCP system also offers a 1 Gbit/s 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.

2.5 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 dips, etc.
- Its huge computing power makes it suitable to include supervising systems (Maybe based on AI algorithms) or even Digital Twins or Real Time Simulators.

We have developed SmartRCP to make the possibilities infinite.



3 Hardware and CarrierBoard details

This chapter describes the SmartRCP hardware: its mechanical and environmental data, the KRIA K26 SOM it is built on, every connector of the CarrierBoard with its pin-out, and the analog and digital chains that sit behind each connector.

3.1 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. The main dimensions are given in Figure 3.

Total weight is approximately 0.4kg.

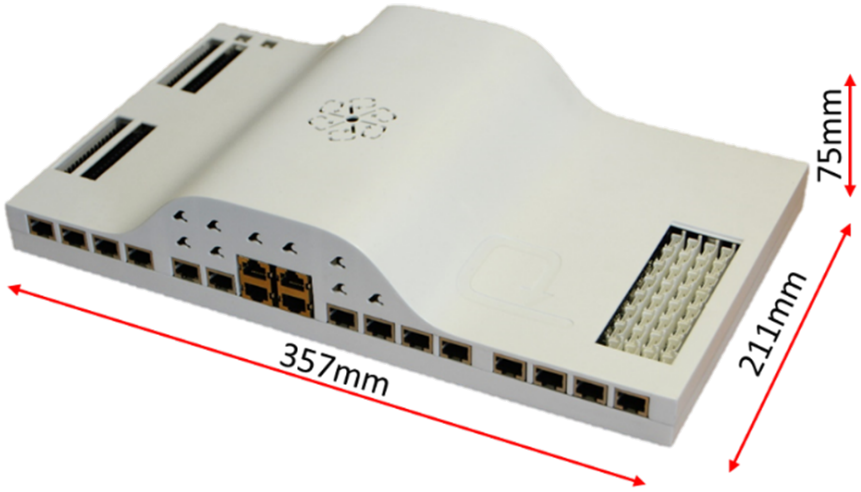


Figure 3: SmartRCP main dimensions.

3.2 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% – condensation should not be allowed.
- Salt ambient: Not supported for long periods.

3.3 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 – 5ns time resolution, RS-422, RJ45 connector
- 24x analog input channels (12x 500ksamples/s and 12x 3Msamples/s)
- 16x analog output channels (500ksamples/s)
- 1x 1 Gbit/s Ethernet (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

3.4 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).

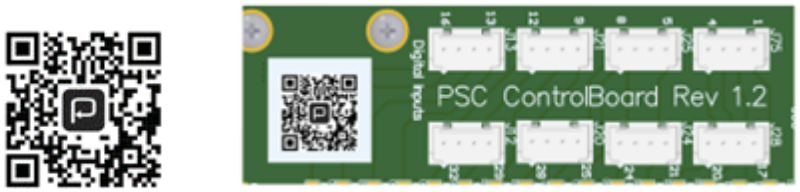


Figure 4: Serial QR codes and HW revision information.

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

3.5 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.

Figure 5 provides a view of the KRIA K26 SOM.

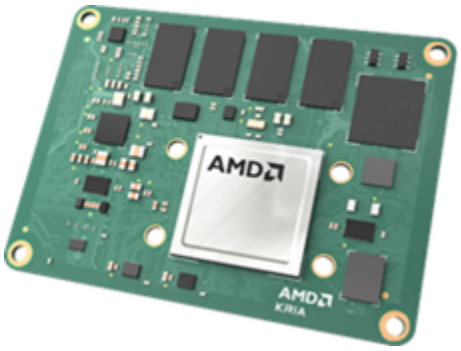


Figure 5: KRIA K26 SOM comparison among other Xilinx devices [1].

KRIA K26 SoC includes:

- Huge highly enriched FPGA (Figure 6)
- 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.)

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

Figure 6: Programmable-logic resources of the KRIA K26 SOM [2].

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). Figure 7 shows KRIA 26 architecture.

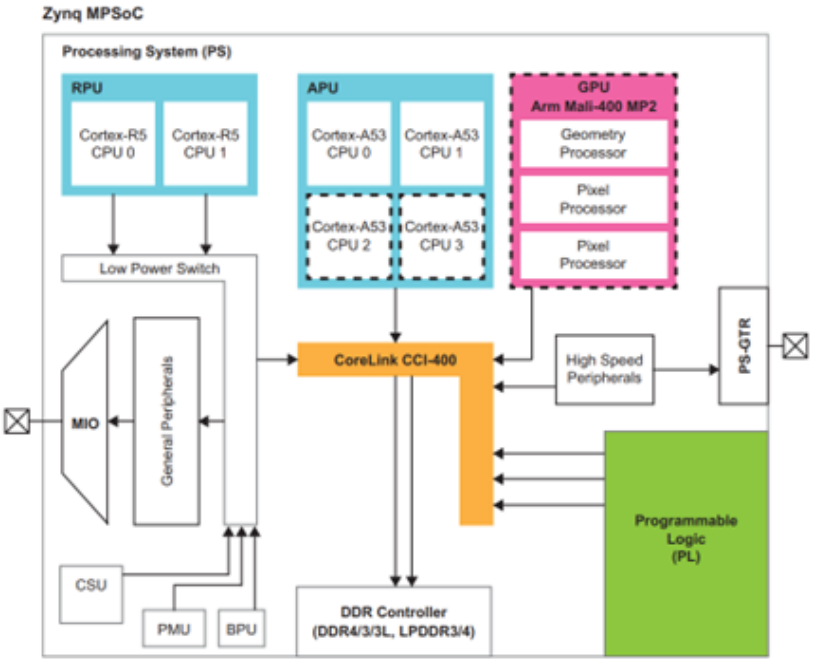


Figure 7: KRIA K26 main components and architecture [3].

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 Car-

rierBoard, offers the market the ultimate and most powerful solution for Rapid Control Prototyping.

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

Table 1 describes all connectors available in the CarrierBoard, its pin-out, and its bandwidth. The 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.

Figure 8 shows the physical placement of each connector inside the card.

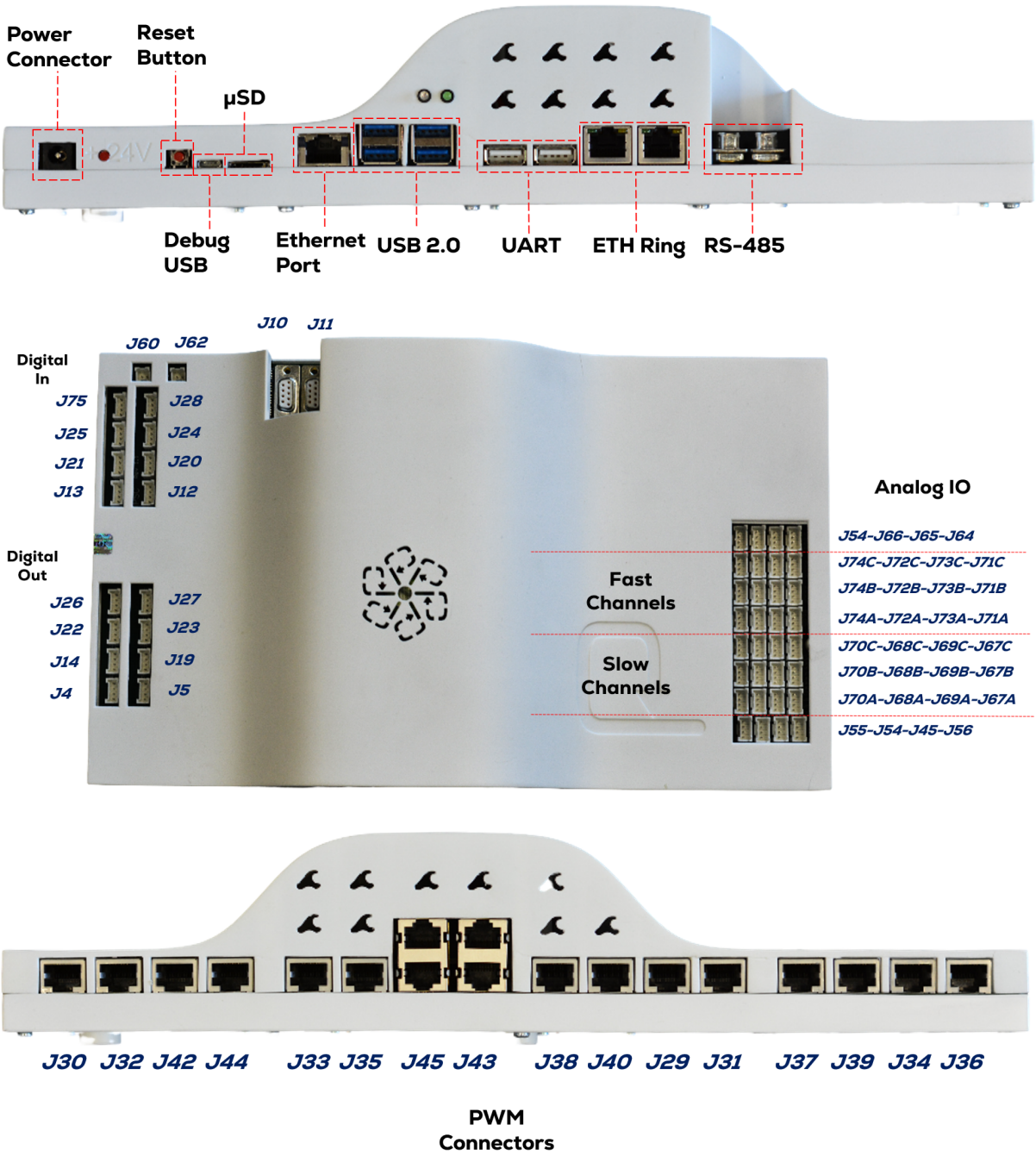


Figure 8: Physical placement of the connectors on the CarrierBoard: communication side, top view and PWM side.

Table 1: CarrierBoard connectors: function, pin-out, voltage range and bandwidth.

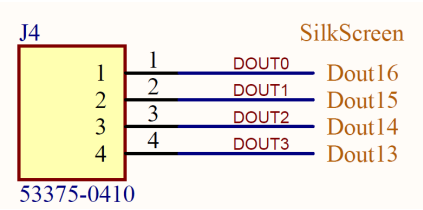
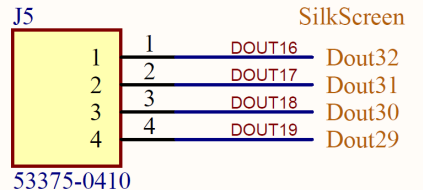
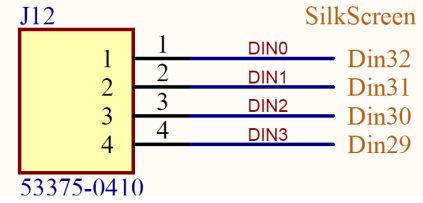
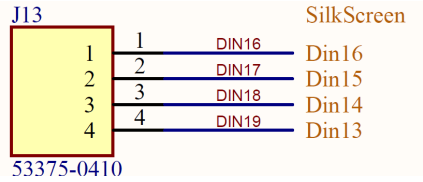
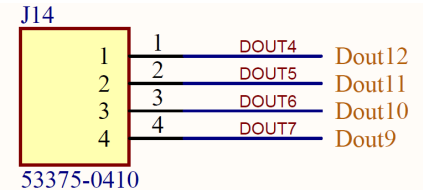
ID	Function	Pin-out	V	BW	Comment
J4	Digital outputs		0–24V	1.5kHz	Multiplexed fast i2c
J5	Digital outputs		0–24V	1.5kHz	Multiplexed fast i2c
J6	Programming & debugging	Female Micro USB connector	0–5V		Connect to a PC
J8	ETH	ETH RJ45 connector	0–3.3V	1 Gbit/s	Universal use
J9	Micro SD card	Micro SD card 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

Table 1: CarrierBoard connectors: function, pin-out, voltage range and bandwidth (continued).

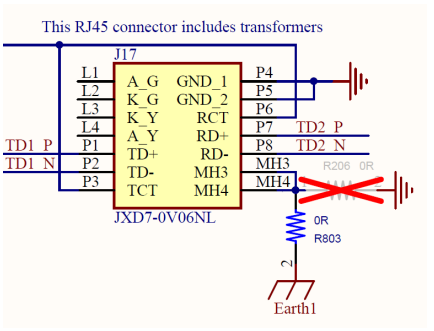
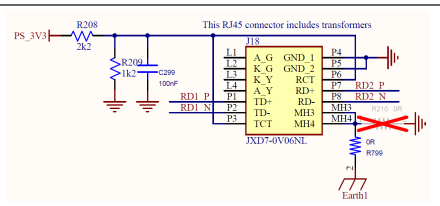
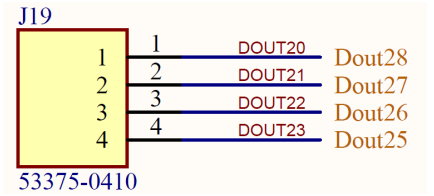
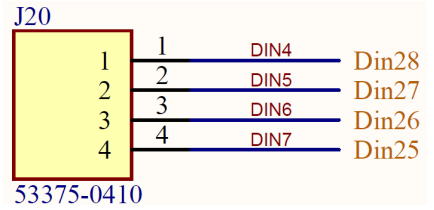
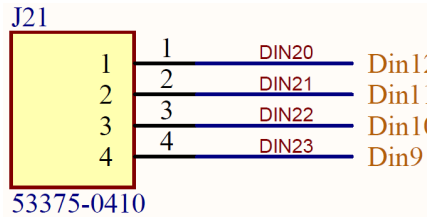
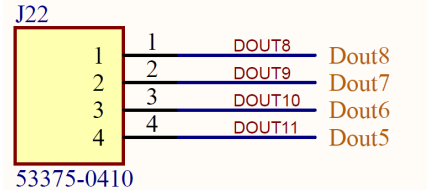
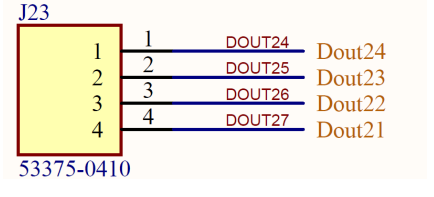
ID	Function	Pin-out	V	BW	Comment
J17	ETH Ring				
J18	ETH Ring				
J19	Digital outputs		0–24V	1.5kHz	Multiplexed fast i2c
J20	Digital inputs		0–24V	1.5kHz	Multiplexed fast i2c
J21	Digital inputs		0–24V	1.5kHz	Multiplexed fast i2c
J22	Digital outputs		0–24V	1.5kHz	Multiplexed fast i2c
J23	Digital outputs		0–24V	1.5kHz	Multiplexed fast i2c

Table 1: CarrierBoard connectors: function, pin-out, voltage range and bandwidth (continued).

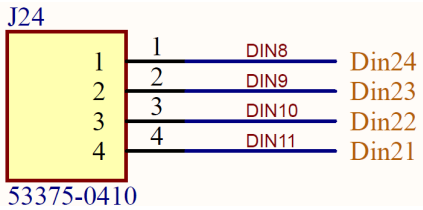
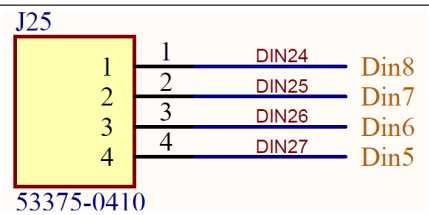
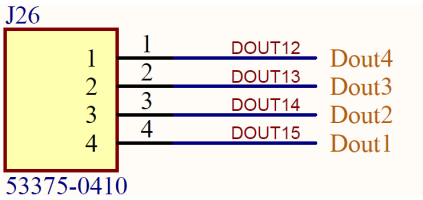
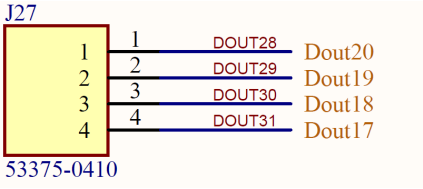
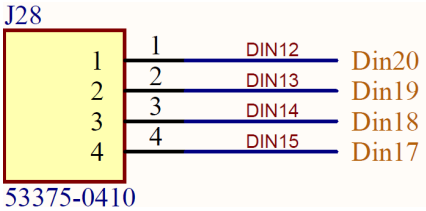
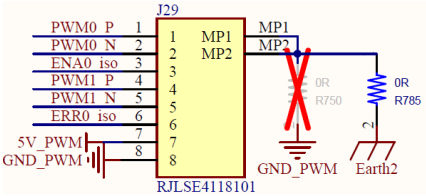
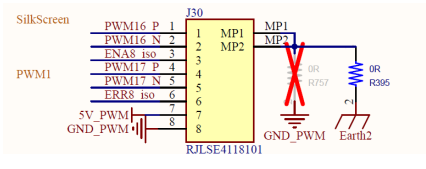
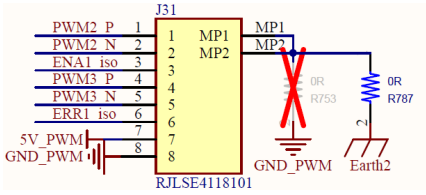
ID	Function	Pin-out	V	BW	Comment
J24	Digital inputs		0–24V	1.5kHz	Multiplexed fast i2c
J25	Digital inputs		0–24V	1.5kHz	Multiplexed fast i2c
J26	Digital outputs		0–24V	1.5kHz	Multiplexed fast i2c
J27	Digital outputs		0–24V	1.5kHz	Multiplexed fast i2c
J28	Digital inputs		0–24V	1.5kHz	Multiplexed fast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

Table 1: CarrierBoard connectors: function, pin-out, voltage range and bandwidth (continued).

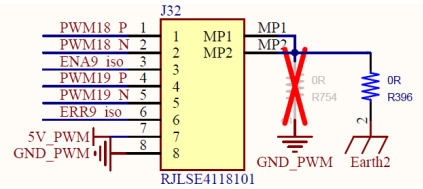
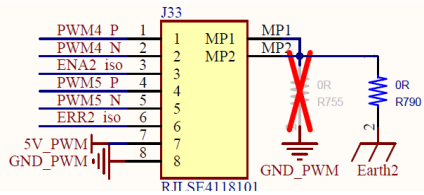
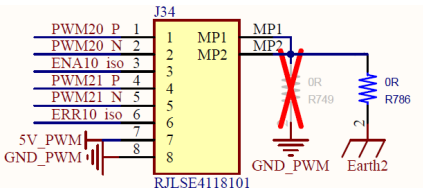
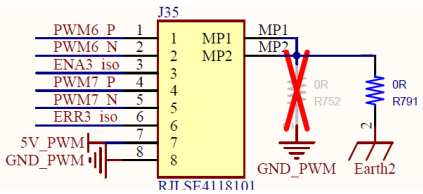
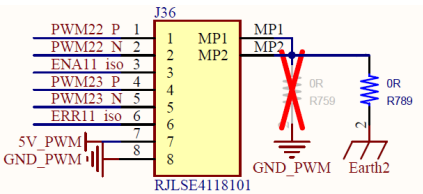
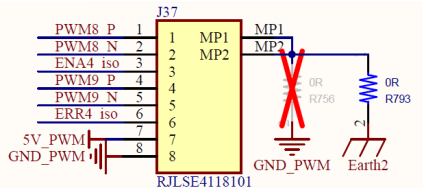
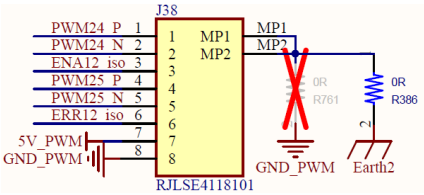
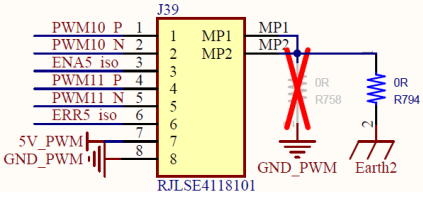
ID	Function	Pin-out	V	BW	Comment
J32	PWM		0–5V	5ns	RS-422 differential
J33	PWM		0–5V	5ns	RS-422 differential
J34	PWM		0–5V	5ns	RS-422 differential
J35	PWM		0–5V	5ns	RS-422 differential
J36	PWM		0–5V	5ns	RS-422 differential
J37	PWM		0–5V	5ns	RS-422 differential
J38	PWM		0–5V	5ns	RS-422 differential
J39	PWM		0–5V	5ns	RS-422 differential

Table 1: CarrierBoard connectors: function, pin-out, voltage range and bandwidth (continued).

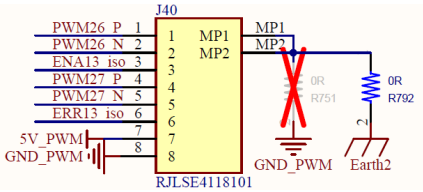
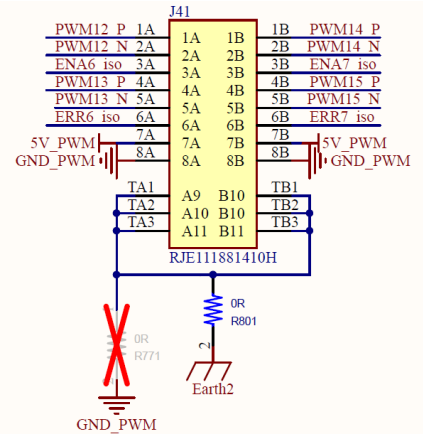
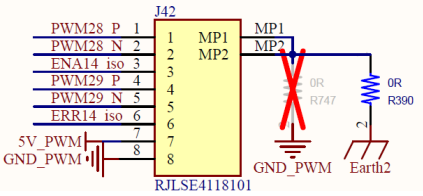
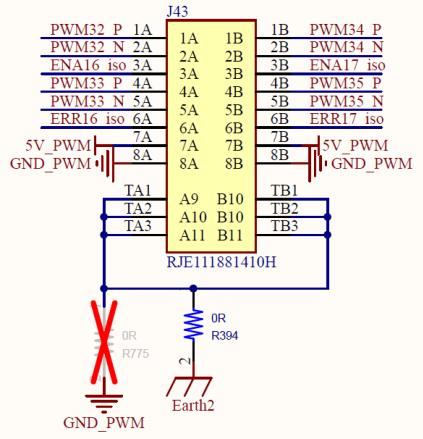
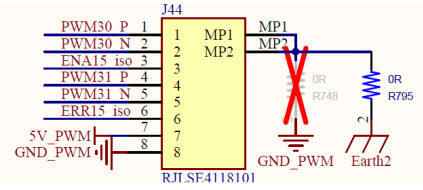
ID	Function	Pin-out	V	BW	Comment
J40	PWM		0–5V	5ns	RS-422 differential
J41	PWM		0–5V	5ns	RS-422 differential
J42	PWM		0–5V	5ns	RS-422 differential
J43	PWM		0–5V	5ns	RS-422 differential
J44	PWM		0–5V	5ns	RS-422 differential

Table 1: CarrierBoard connectors: function, pin-out, voltage range and bandwidth (continued).

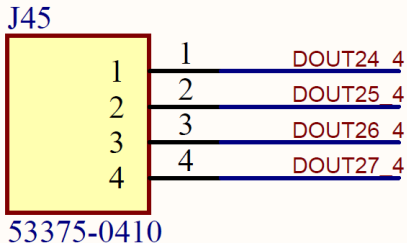
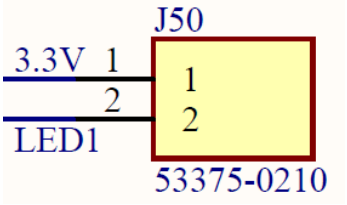
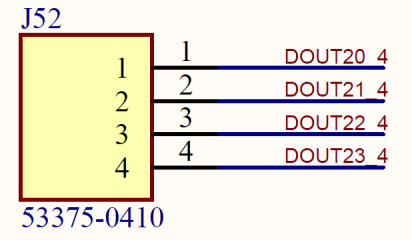
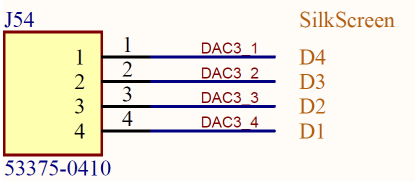
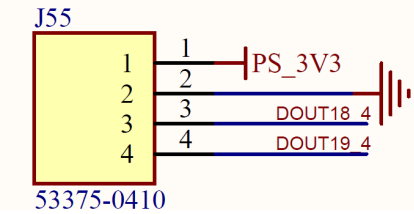
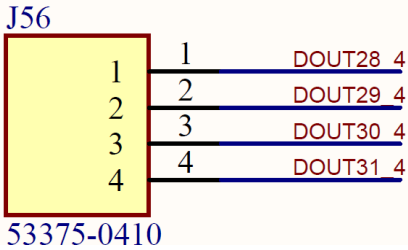
ID	Function	Pin-out	V	BW	Comment
J45	Digital outputs		0–3.3V	1.5kHz	Multiplexed fast i2c
J50	User LED		0–3.3V		Green panel LED
J52	Digital outputs		0–3.3V	1.5kHz	Multiplexed fast i2c
J54	DACs		0–3.3V	500 ks/s	Connector pin is the output of DAC IC, no analog chain included
J55	Digital outputs		0–3.3V	1.5kHz	Multiplexed fast i2c
J56	Digital outputs		0–3.3V	1.5kHz	Multiplexed fast i2c

Table 1: CarrierBoard connectors: function, pin-out, voltage range and bandwidth (continued).

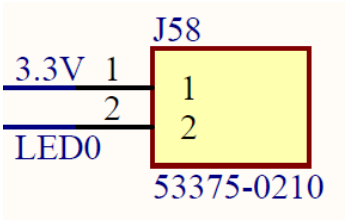
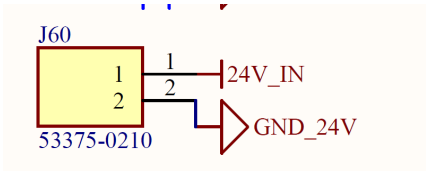
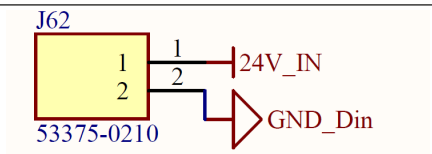
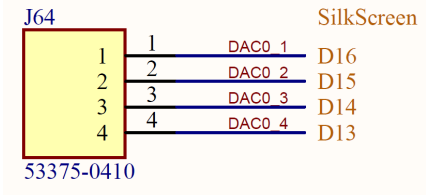
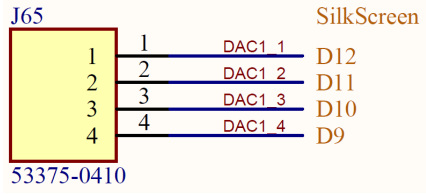
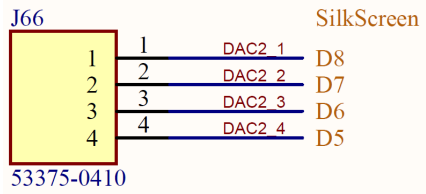
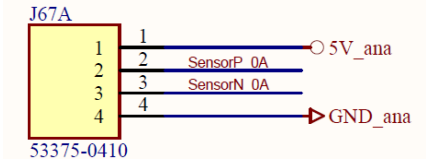
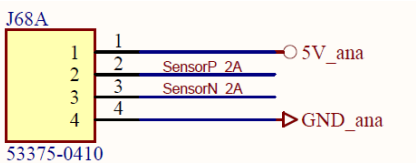
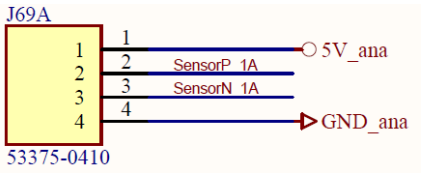
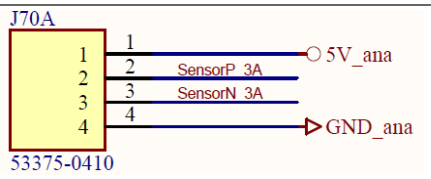
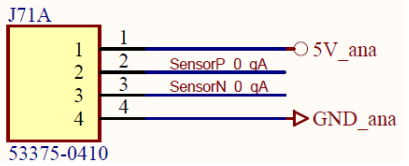
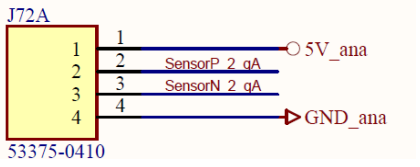
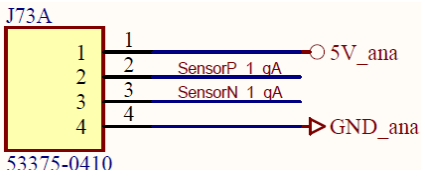
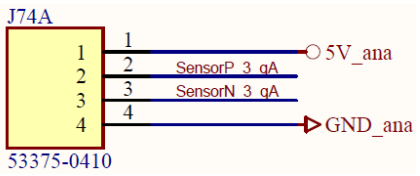
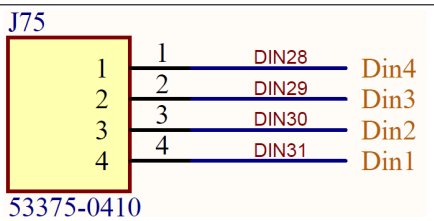
ID	Function	Pin-out	V	BW	Comment
J58	User LED		0–3.3V		Orange panel LED
J60	DOUT 24V power connector		0–24V	DC	Max total current 10mA per channel
J62	DIN 24V power connector		0–24V	DC	Max total current 10mA per channel
J64	DAC		0–3.3V	500 ks/s	No analog chain included
J65	DAC		0–3.3V	500 ks/s	No analog 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	

Table 1: CarrierBoard connectors: function, pin-out, voltage range and bandwidth (continued).

ID	Function	Pin-out	V	BW	Comment
J69ABC	Slow analog input		0–5V	500 ks/s	
J70ABC	Slow analog input		0–5V	500 ks/s	
J71ABC	Quick analog input		0–5V	3 Ms/s	
J72ABC	Quick analog input		0–5V	3 Ms/s	
J73ABC	Quick analog input		0–5V	3 Ms/s	
J74ABC	Quick analog input		0–5V	3 Ms/s	
J75	Digital inputs		0–24V	1.5kHz	Multiplexed fast i2c
U36	2x USB 2.0		0–5V		
U38	2x USB 2.0		0–5V		
SW2	Reset	Reset push button			
XSW1 / XSW2	Power connector	Power connector barrel style (Inner contact is positive terminal)			24V / 60W max

3.7 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 one of Listing 1:

Listing 1: **Syntax of the constraints file**

```
## SOM BANK 1-B - 3.3V
set_property PACKAGE_PIN J10 [get_ports PMOD1_1_1_P2];
set_property IOSTANDARD LVCMOS33 [get_ports PMOD1_1_1_P2];
set_property SLEW SLOW [get_ports PMOD1_1_1_P2];
set_property DRIVE 4 [get_ports PMOD1_1_1_P2];
# 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 RCP_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 RCP_TEMPLATE provided. All of them contain the name of the port and the name of the connector involved.



3.8 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.

3.9 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.

3.10 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.

3.11 In-depth description of functional blocks

3.11.1 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 Figure 9:

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 Figure 10:

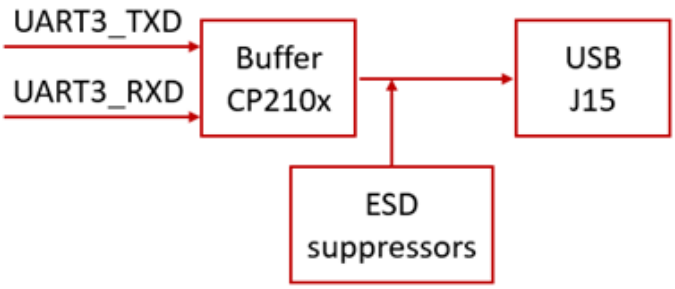


Figure 9: RS232 functional block chain (J15).

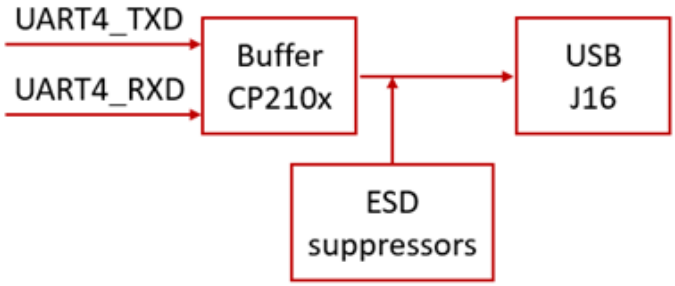


Figure 10: RS232 functional block chain (J16).

3.11.2 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.**

Figures 11 and 12 provide all the required information about the chain:

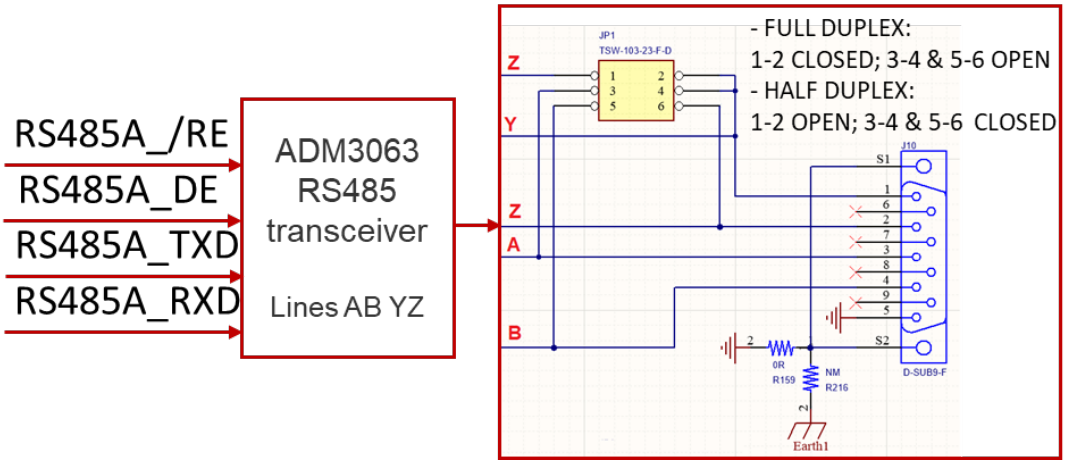


Figure 11: RS485 functional block chain (J10).

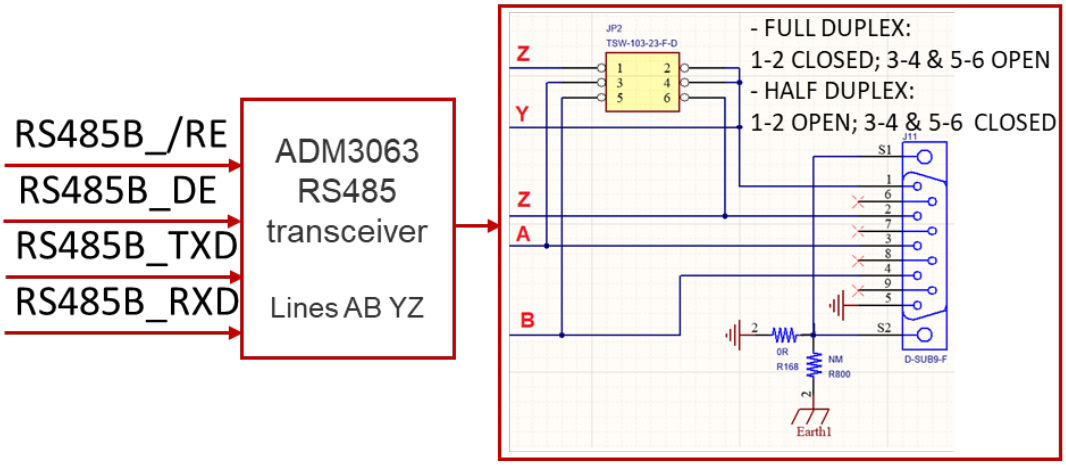


Figure 12: RS485 functional block chain (J11).

3.11.3 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 13; Figure 14 shows the chain of each LED.

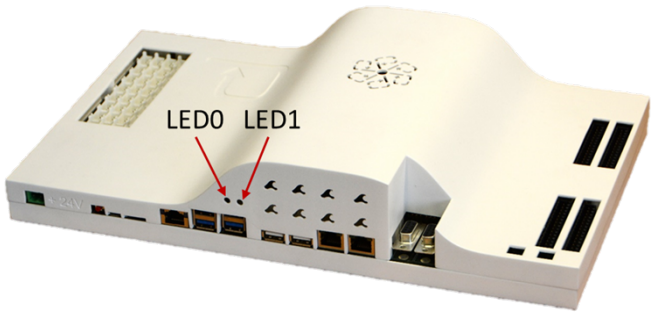


Figure 13: SmartRCP user LED position.

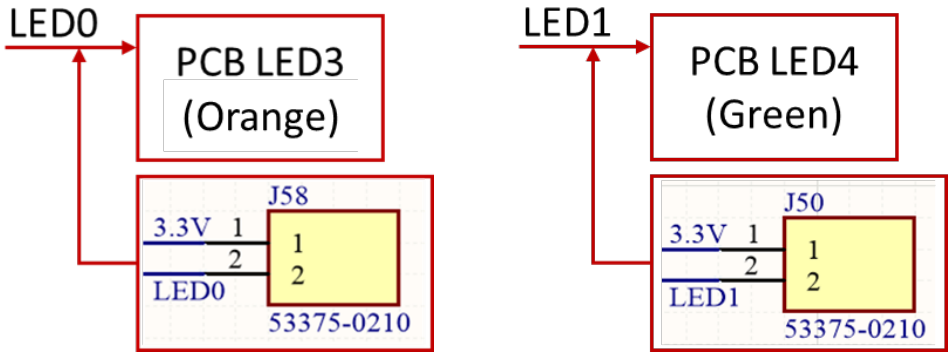


Figure 14: User LEDs functional block chain.

3.11.4 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 15 shows the chain.

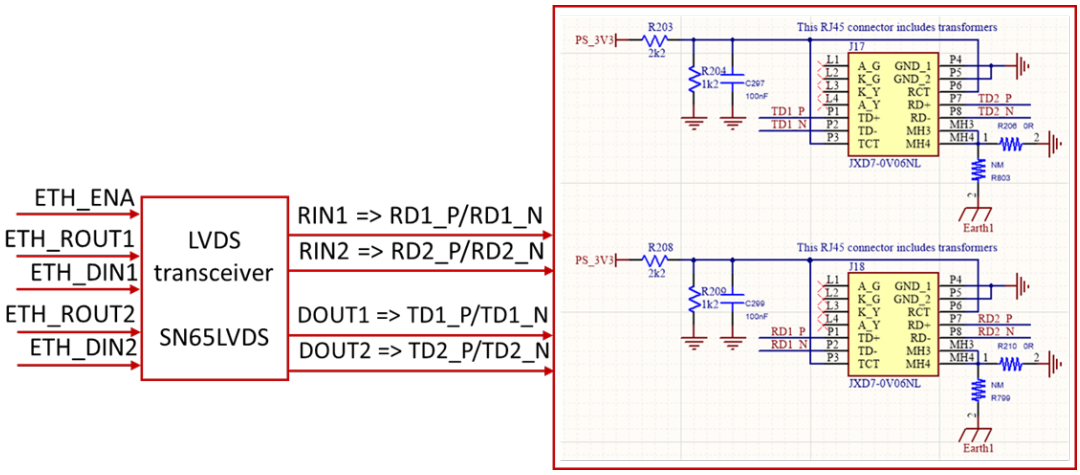


Figure 15: Ethernet ring functional block chain.

3.11.5 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 16 provides more information about the signals involved and the structure used.

3.11.6 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

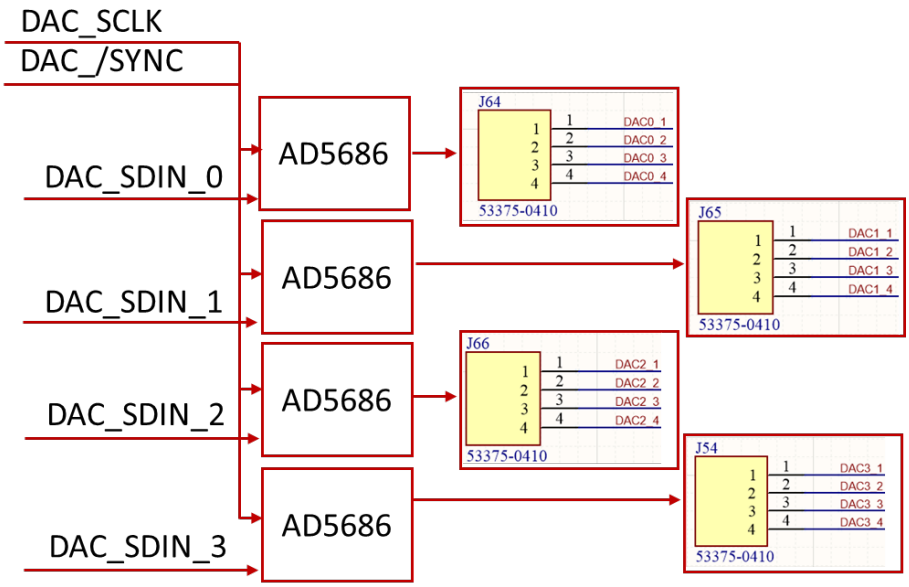


Figure 16: Analog outputs functional block chain.

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 Figure 17.

3.11.7 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 Figure 18.

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. Their chain is shown in Figure 19.

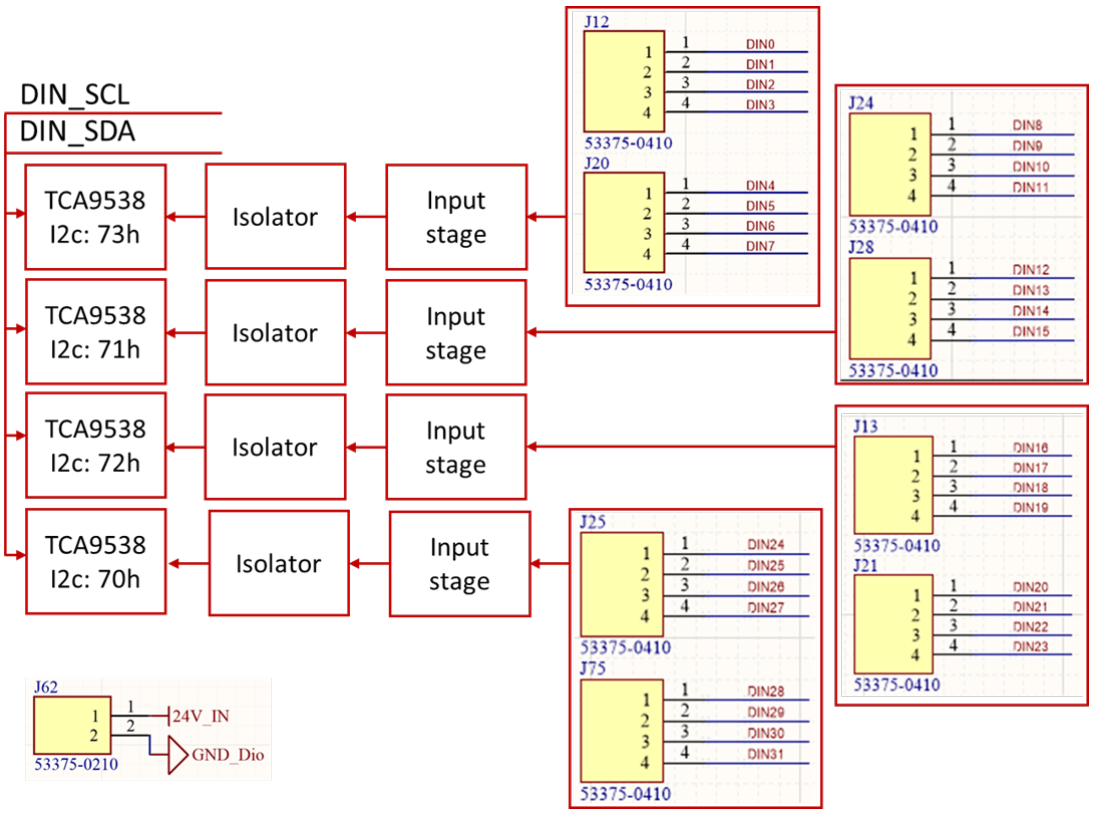


Figure 17: 0–24V Digital inputs functional block chain.

3.11.8 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.

Figures 20 to 29 provide more info about the different signals included.

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 Figure 30:

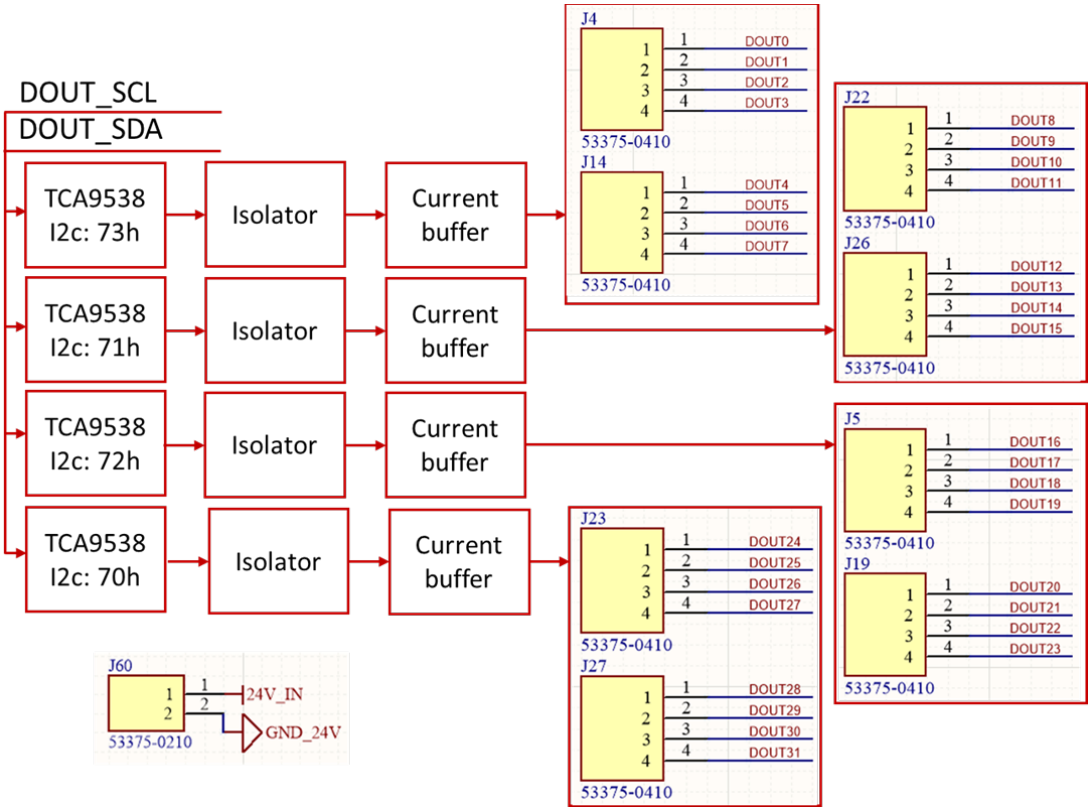


Figure 18: 0–24V Digital outputs functional block chain.

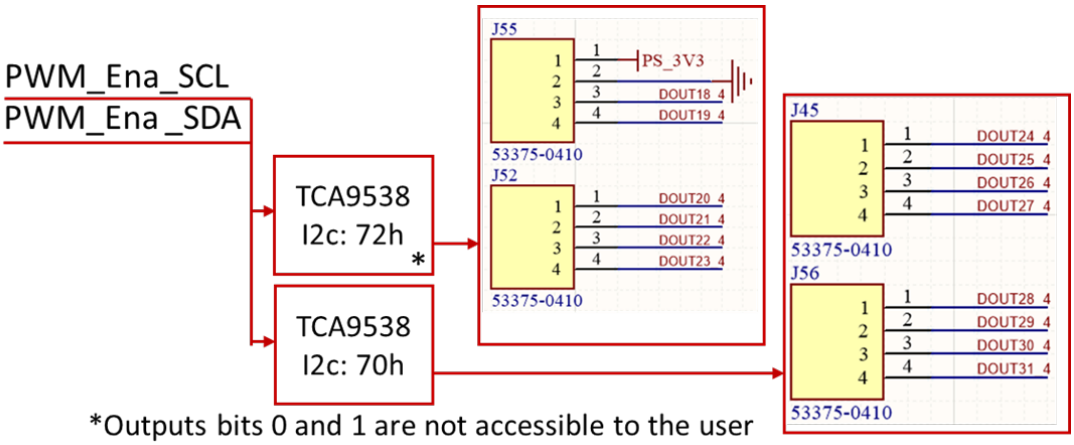


Figure 19: Additional 0–3.3V functional block chain.

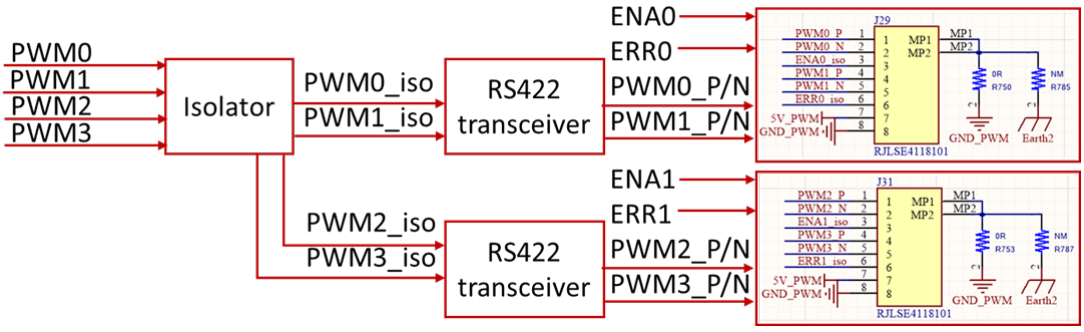


Figure 20: PWM functional block chain (J29 and J31).

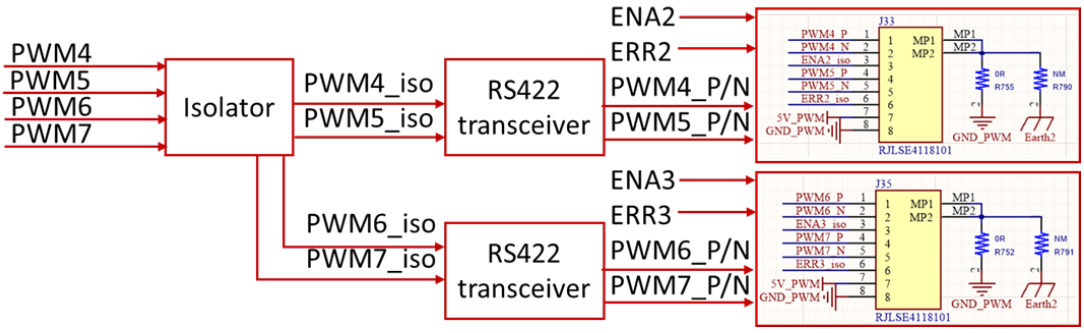


Figure 21: PWM functional block chain (J33 and J35).

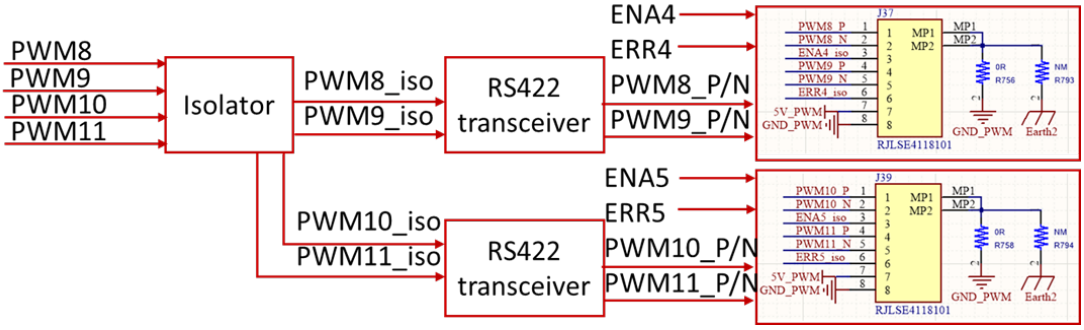


Figure 22: PWM functional block chain (J37 and J39).

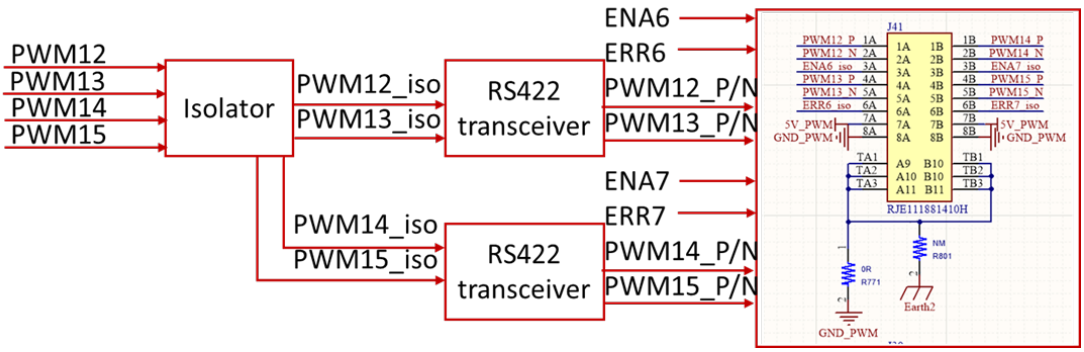


Figure 23: PWM functional block chain (J41).

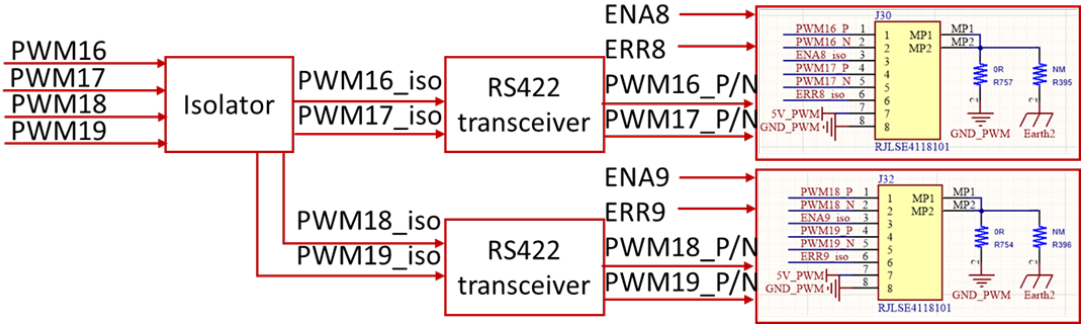


Figure 24: PWM functional block chain (J30 and J32).

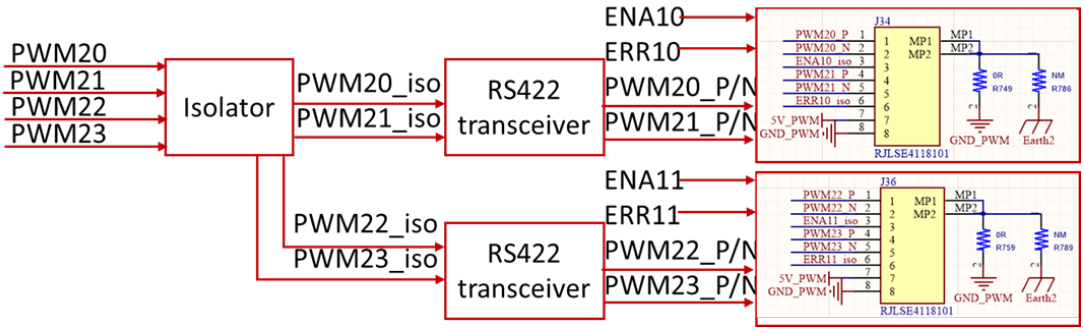


Figure 25: PWM functional block chain (J34 and J36).

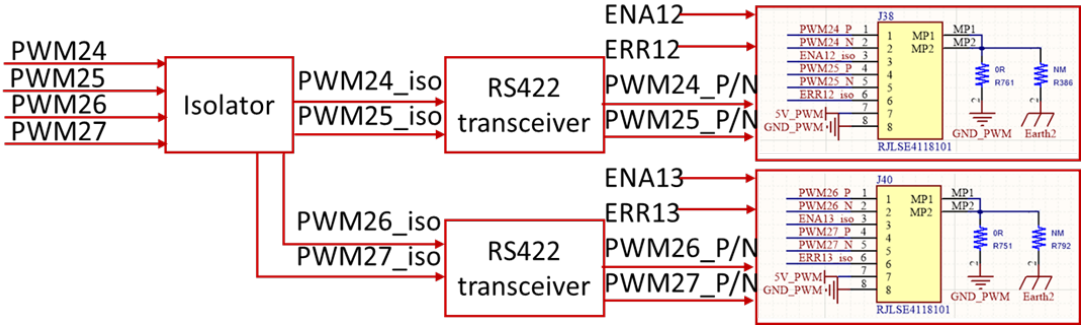


Figure 26: PWM functional block chain (J38 and J40).

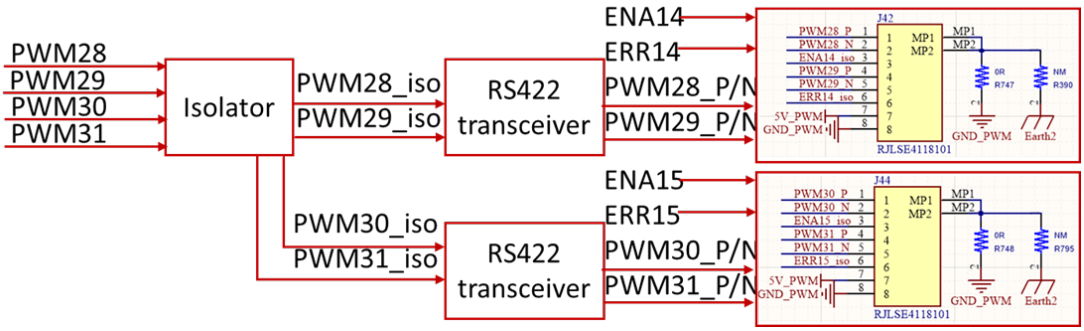


Figure 27: PWM functional block chain (J42 and J44).

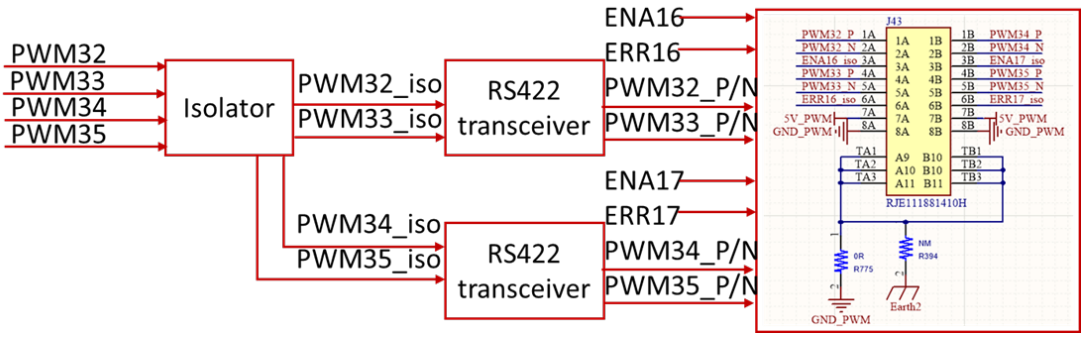


Figure 28: PWM functional block chain (J43).

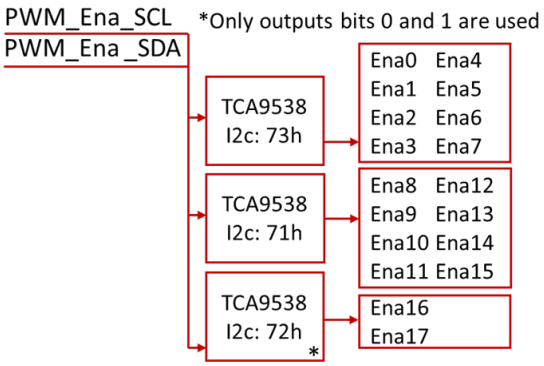


Figure 29: PWM functional block chain. Enable signal detail.

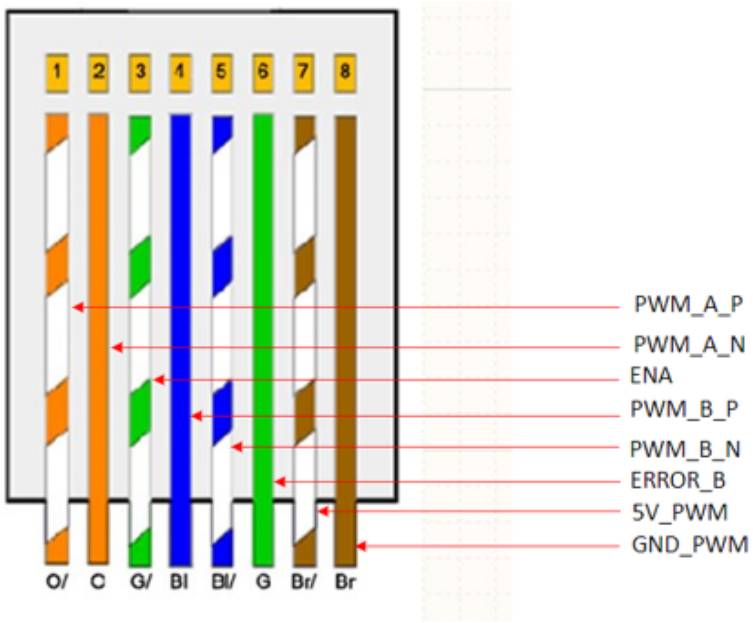


Figure 30: PWM channels RJ45 connector pin-out.

3.11.9 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. Figure 31 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 Equation 1:

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

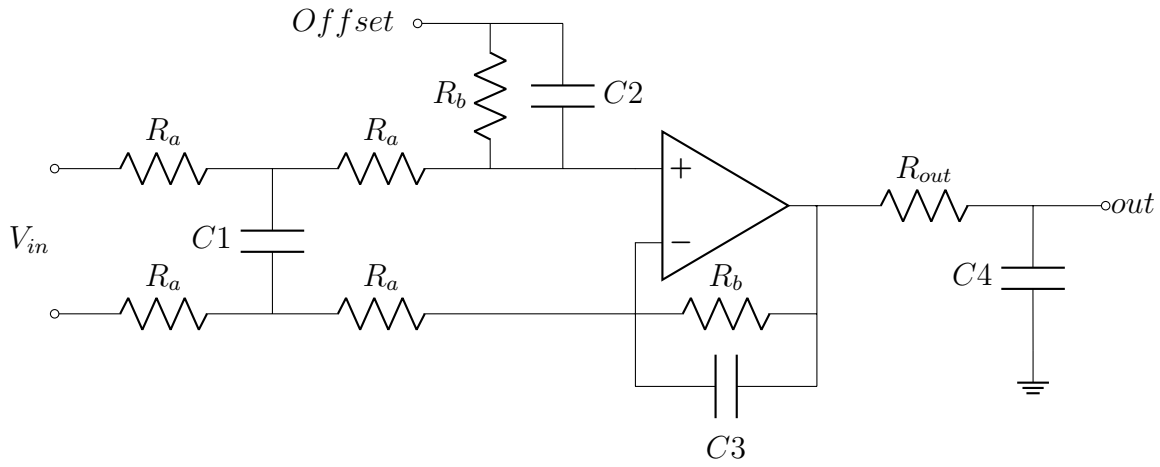


Figure 31: Analog input channels front end.

Slow analog channels

Those channels have the following details:

- Differential high impedance input
- Analog input voltage span: 0–5V
- Total gain of the analog chain: 1V/V (5V/5V=1V/V)
- Value of the components: Ra=5kOhm, Rb=10kOhm, C1=56pF, C2=56pF, C3=56pF, Rout=33Ohm, C4=10nF, Voffset=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 span: 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 32 provides more details about signals:

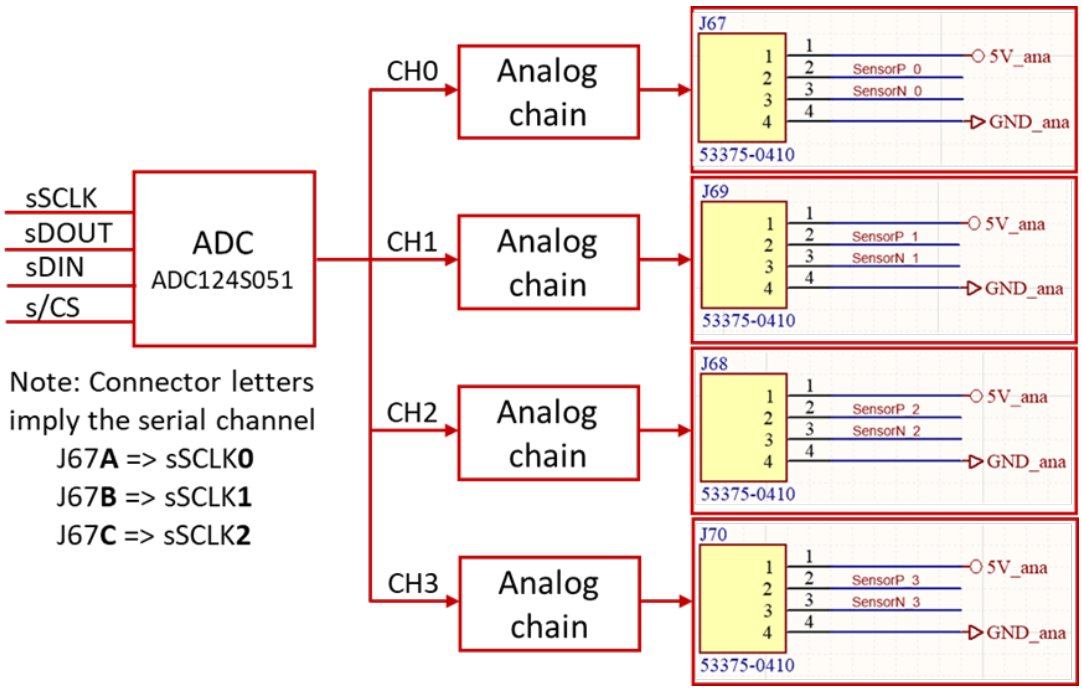


Figure 32: Slow analog input functional block chain.

Quick analog channels

Those channels have the following details:

- Differential high impedance input
- Analog input voltage span: 0–5V
- Total gain of the analog chain: 0.66V/V (3.3V/5V=0.66V/V)
- Value of the components: Ra=7.5kOhm, Rb=10kOhm, C1=10pF, C2=10pF, C3=10pF, Rout=33Ohm, C4=1nF, Voffset=0V (selectable among 0V and 1.65V)
- ADC features (AD7276A): 12bits per channel, 4 channels, 3Ms/s per channel, SPI
- ADC analog input span: 0–3.3V
- Maximum bandwidth of the analog input quick channel (−3dB): 1MHz (limited by C1–C4 capacitors this BW can be modified by PSC under special request)

Figure 33 provides more detail about signals:

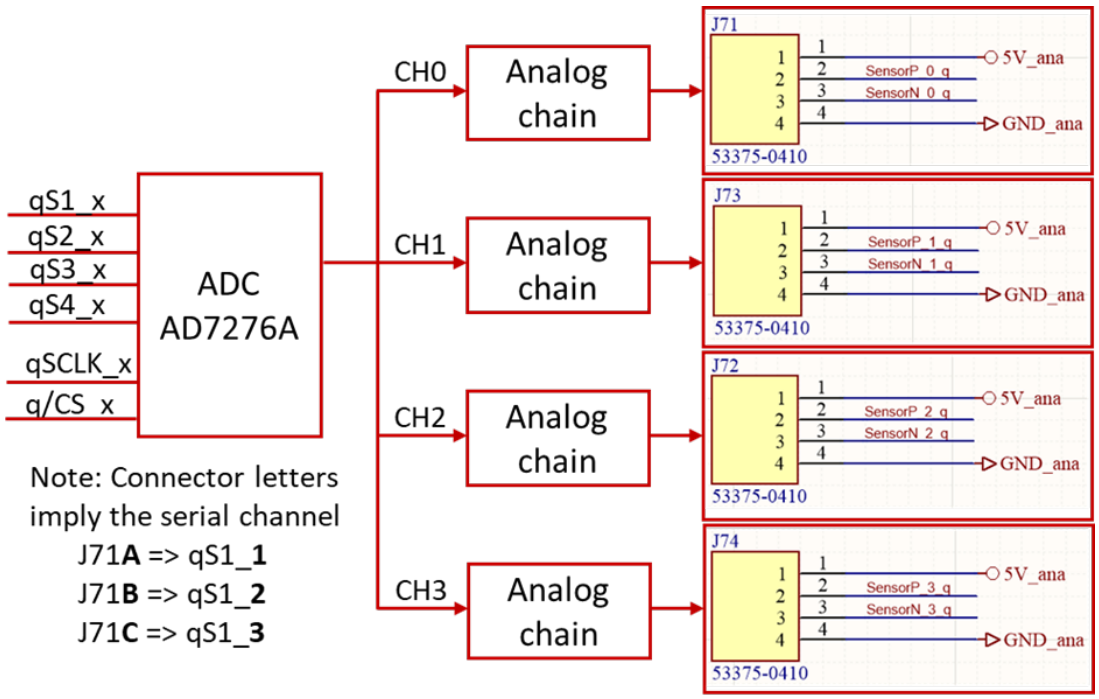


Figure 33: Quick analog input functional block chain.

3.12 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.

4 Programming user guide

This chapter describes the software environment used to program the SmartRCP: the AMD tools to install, the Vivado flow for the programmable logic together with the `RCP_TEMPLATE` project delivered with the board, and the Vitis flow for the processors of the Zynq UltraScale+ MPSoC.

4.1 Development environment

> **Tool versions used in this manual**
> The RCP_TEMPLATE project and every project delivered with the SmartRCP are built with AMD Vivado and Vitis **2023.2**. The procedures in this chapter are valid for that version and later ones: when a step names a version to download or install, use 2023.2 (or the version your template was delivered with) in its place.

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.

4.1.1 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.

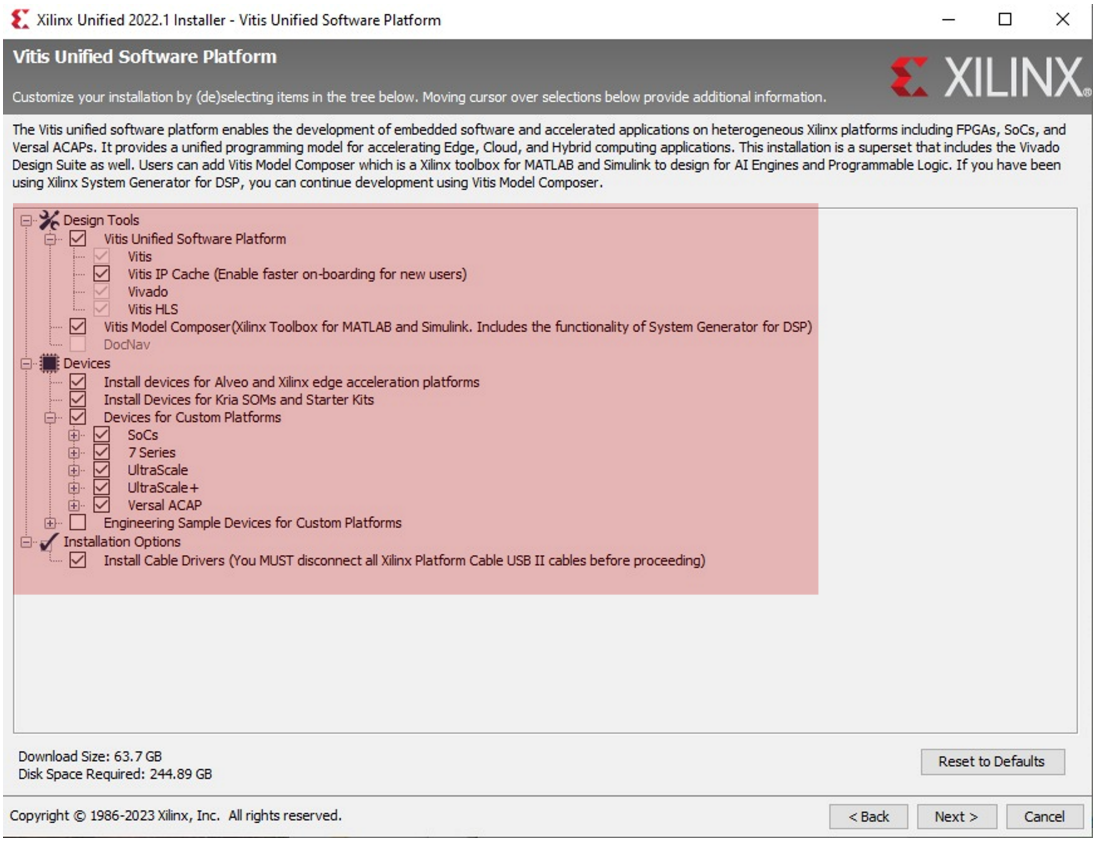


Figure 34: Vitis installing options.

4.1.2 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 2023.2. When selecting the version to download, the user must consider that the Vivado template can only be used properly with versions equal to or newer than the one it was created with. It is recommended to always select the latest version available at:

1. Click on the link below `https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2023-2.html`, select the Web Installer option that corresponds to your operating system (Figure 35). An AMD account must be created before you can install.



Figure 35: Web Installer entry on the AMD download page (screenshot of an earlier release; pick the 2023.2 entry).

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 options of Figure 34 and click next.
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.

4.2 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 RCP_TEMPLATE

4.2.1 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 options of Figure 36 checked and click on next.
4. In the new screen, go to **Boards** and select **Kria K26I SOM**. See Figure 37.
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 38).
    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 (Figure 39).

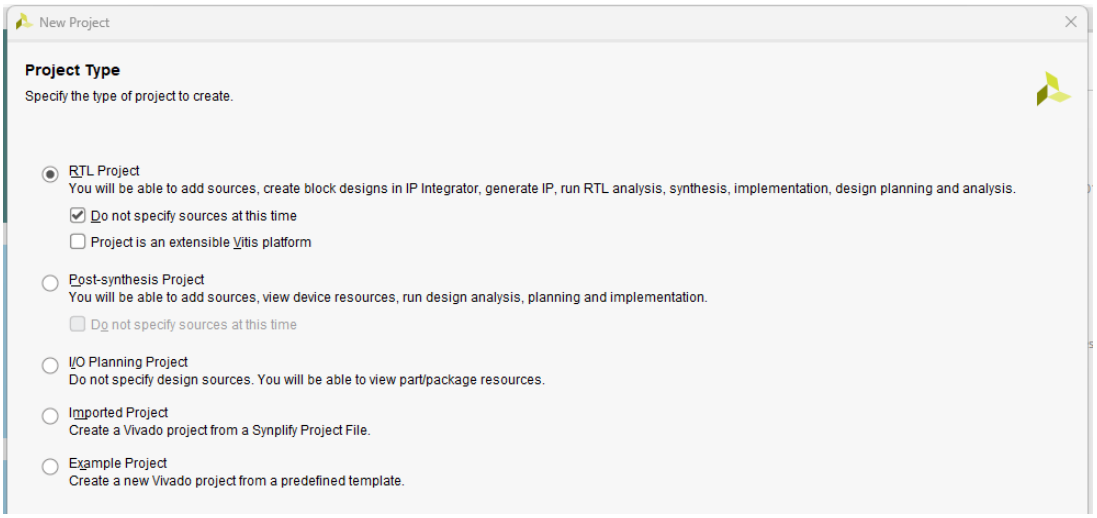


Figure 36: Vivado project options.

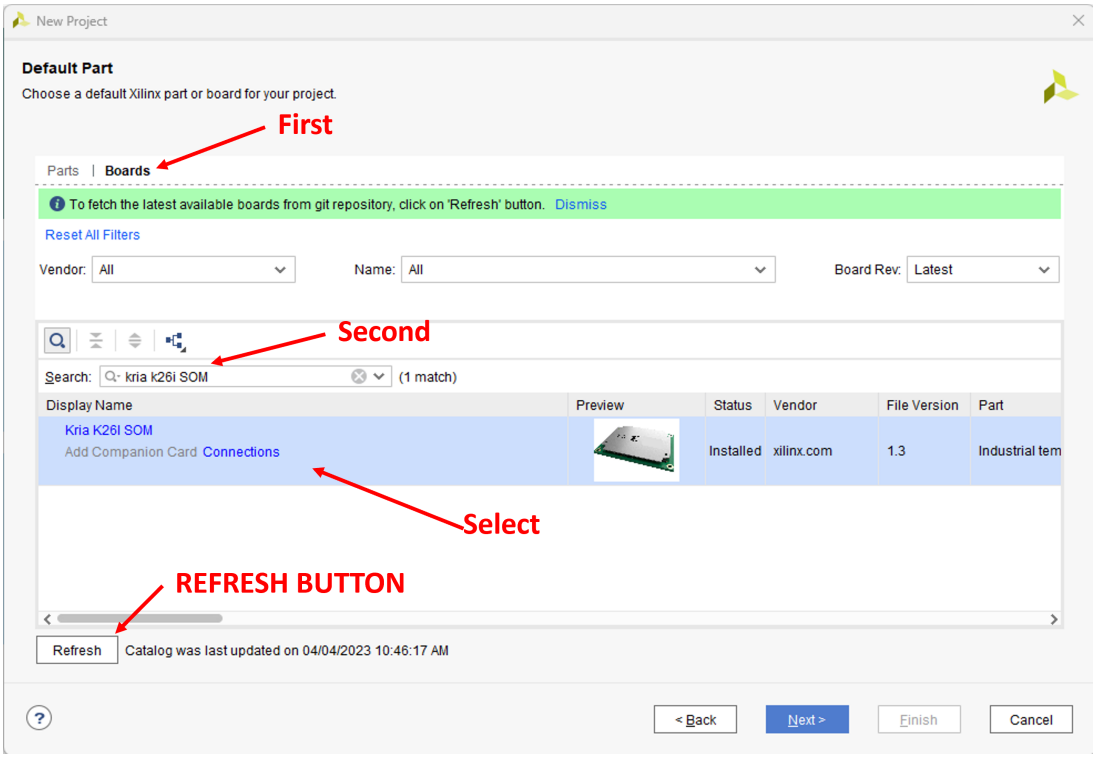


Figure 37: Vivado project part definition.

c. In **Implementation** go to settings and disable all optimization options (Figure 40).

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

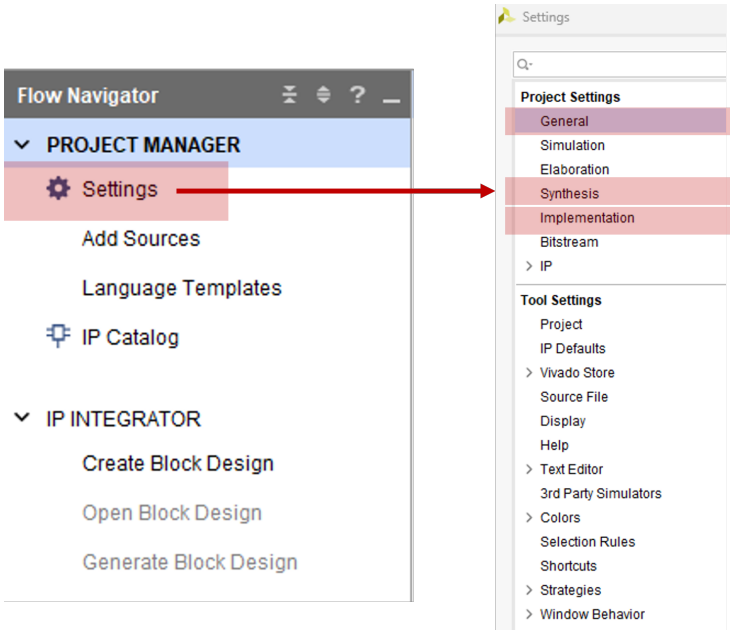


Figure 38: Vivado project option configuration.



Figure 39: Incremental synthesis option in the Synthesis settings.

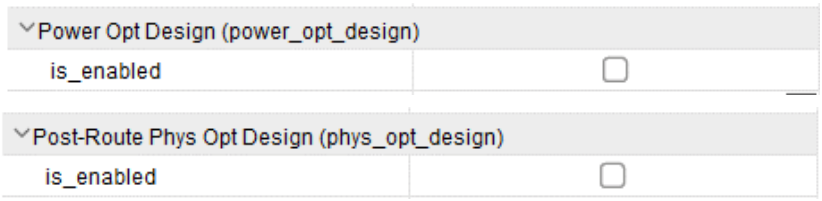


Figure 40: Implementation settings: optimization options to disable.

double-click on it. It will be added to your central window (the graphical interface of the design).

9. Click on **Run Block Automation** (Figure 41), a new window will pop up, click on OK.

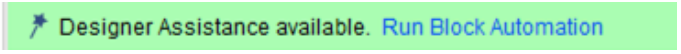


Figure 41: Designer Assistance banner with the Run Block Automation link.

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

4.2.2 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 42).

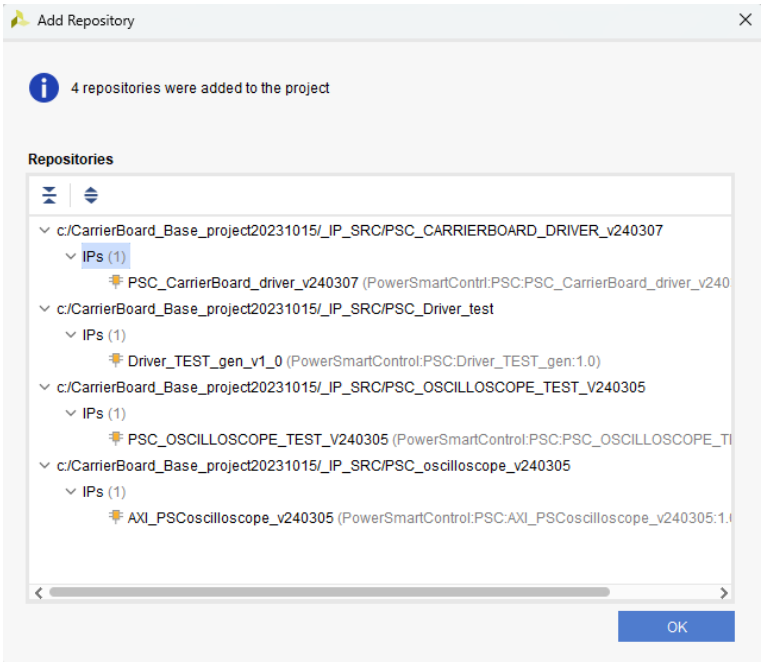


Figure 42: IP repository in Vivado project.

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.

4.2.3 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 43).

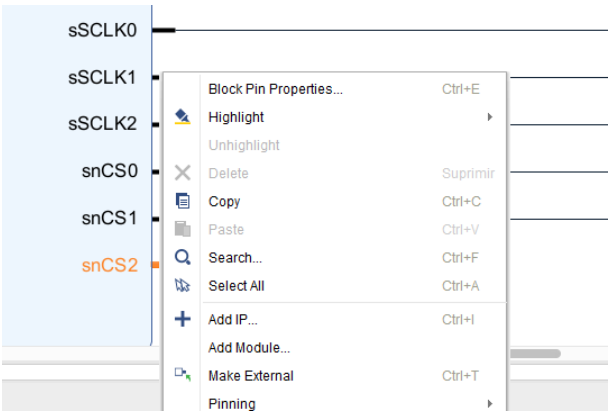


Figure 43: Creating an external port.

2. Change the port name to match the name of the output you want to use from the constraints (Figure 44).

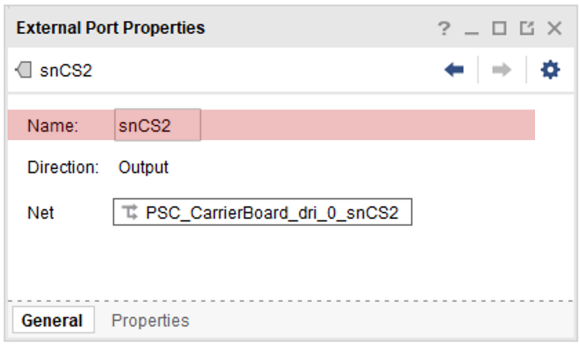


Figure 44: Changing name to an external port.

4.2.4 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 **Validate Design** or F6
2. Click on **Create HDL Wrapper** (Figure 45).

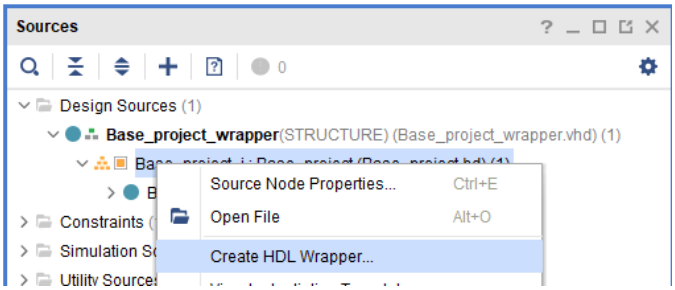


Figure 45: Creating HDL wrapper.

3. Click on **Generate Output Products** (Figure 46), a new window emerges, make sure the settings of Figure 47 are checked.

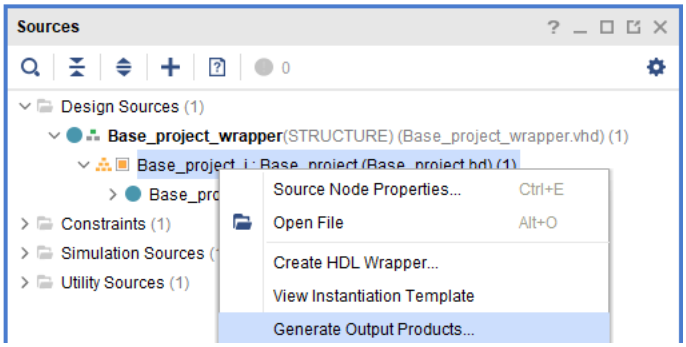


Figure 46: Generating output products.

4. Click on **Generate Bitstream** 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 48).
6. Click next in the tabs, make sure that in the second tab the option **Include bitstream** is checked (Figure 49).

4.2.5 How to use the RCP_TEMPLATE provided by PSC

As part of the resources delivered with the purchase of a SmartRCP, the user receives a set of files that includes a pre-configured Vivado project template[1], which can be modified according to the specific requirements of the project. This template greatly simplifies the integration and configuration of the SmartRCP, allowing the user to focus only on the logic of their design without worrying about low-level details.

[1]Earlier editions of this manual called it *SmartRCP_Template*; the delivered archive and the Vivado project are named `RCP_TEMPLATE`.

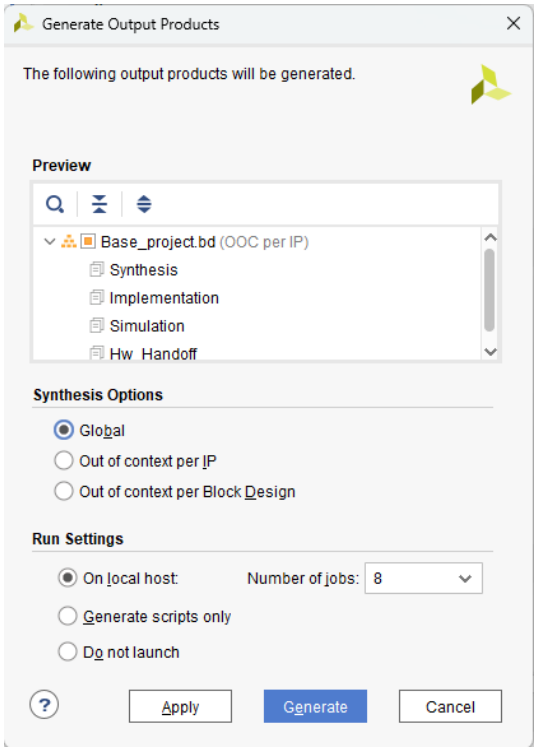


Figure 47: Generate Output Products dialog with the settings to check.

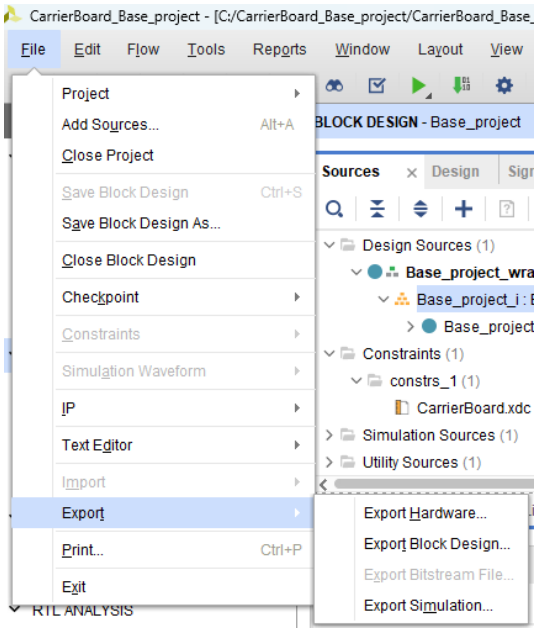


Figure 48: Exporting HW.

Vivado template components

1. **Vivado block design with all the necessary IP drivers:** The project contains a design that includes all the IP drivers required to access the SmartRCP peripherals (ADCs, DACs, etc.). These IP drivers are designed with a "plug-and-play" philosophy, so the user does not need to concern themselves with peripheral com-

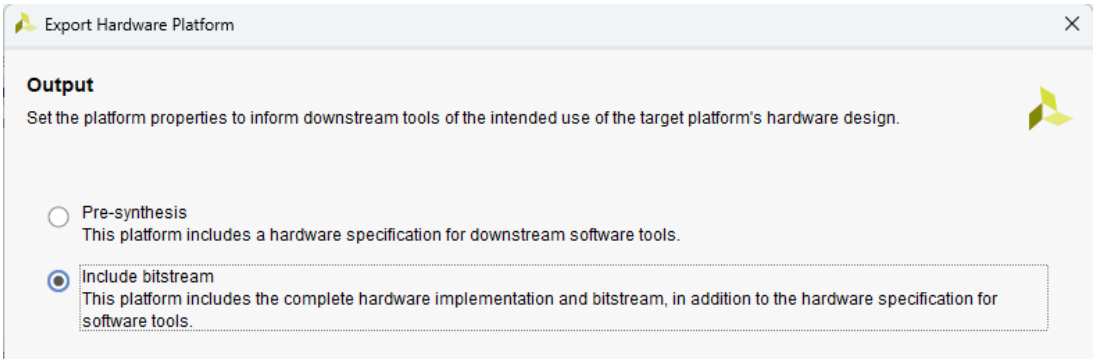


Figure 49: Exporting HW with bitstream.

munication protocols or internal configurations. The purpose is for the user to concentrate only on the logic of their own project.

2. **Basic predefined settings:** The template includes default settings for synthesis, implementation, and HDL file generation. By default, the project is configured to generate VHDL, although the user can change this option to Verilog if preferred.
3. **Zynq UltraScale+ MPSoC configuration:** The template handles basic configurations for the MPSoC, such as AXI bus connections, 200 MHz main clock configuration, and operating system-managed peripheral control. This configuration facilitates the interaction between the processor and the FPGA programmable logic. A second clock output, `pl_clk1` at 100 MHz, and the AXI PL-PS master interface `AXI HPM0 FPD` are enabled but left unconnected, reserved for the user's own blocks.
4. **IP blocks for the software interface:** IP blocks are included that allow the user to interact with the SmartRCP software interface, providing a much smoother user experience. These IP blocks allow performing actions such as:
   - Read and write variables on the FPGA.
   - Configure sensors and set enable signals.
   - Manage alarm signals based on ADC values.
   - Access to an oscilloscope to capture and visualize internal FPGA signals in real time (PL-DataLogger).
5. **Constraints file:** The template also includes a constraints file called `RCP_CONSTRAINTS.xdc`, which contains the necessary instructions to map the input and output ports of the design block to the physical pins of the K26 SoC. This ensures that the FPGA signals are correctly connected to peripherals and other external devices.

The Vivado project provided by PSC is `RCP_TEMPLATE.xpr`, in the `RCP_VIVADO` folder of the `RCP_TEMPLATE` archive.

The Vivado project provided with the SmartRCP is organized into several folders that allow the user to easily manage and modify the system's components. The structure of the `RCP_TEMPLATE` folder is shown in Figure 50 and described below:

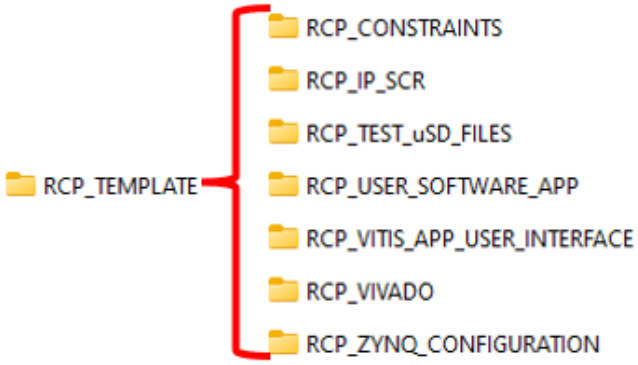


Figure 50: Vivado Project folder structure.

1. `RCP_CONSTRAINTS`: It contains the `RCP_CONSTRAINTS.xdc` file, which defines the relationship between the physical pins of the K26 SoC and the input and output ports of the design. This file is essential to ensure the correct mapping of physical signals to the pins.
2. `RCP_IP_SCR`: This folder contains the IP driver declarations and the IPs required to use the SmartRCP software interface. Inside this folder are the **drivers** that allow interaction with the SmartRCP peripherals, eliminating the need to worry about low-level details.
3. `RCP_TEST_uSD_FILES`: It includes the files needed to test the correct operation of the SmartRCP peripherals. These files, together with the software interface, must be copied to the µSD card to be inserted in the corresponding port of the SmartRCP to run the tests (Figure 51).

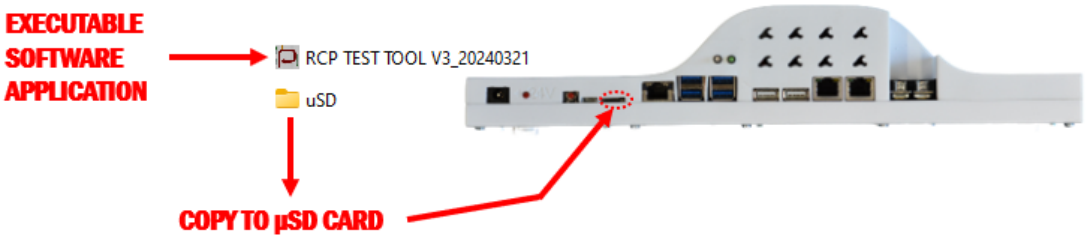


Figure 51: Contents of the RCP_TEST_uSD_FILES folder: the test-tool executable and the files to copy to the µSD card.

4. `RCP_USER_SOFTWARE_APP`: It contains the executable (`.exe`) file of the SmartRCP software interface application, which allows the user to interact with the system, configure sensors, view signals graphically, etc.

5. `RCP_VITIS_APP_USER_INTERFACE`: This folder contains the `.elf` file that the user must include during the software design stage in Vitis. This file is essential to establish the communication between the processor and the software interface, and to take advantage of all the features of the provided template. It also includes the files that must be copied to the µSD to set up the communication between the SmartRCP and the software interface (Figure 52).

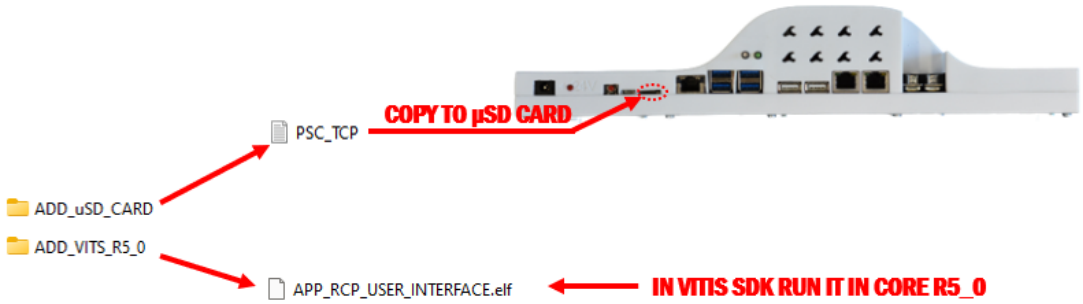


Figure 52: Contents of the RCP_VITIS_APP_USER_INTERFACE folder: the files to copy to the µSD card and the `.elf` application to run from Vitis. Screenshot of an earlier template; the current one names the folder `ADD_VITIS_A53_0` and the file `APP_RCP_USER_INTERFACE_A53_0.elf`, for core `psu_cortexa53_0` (chapter 5).

6. `RCP_VIVADO`: This is the base Vivado project that the user will use as a template. This project includes the predefined configurations and connections that the user can customize according to their design needs.
7. `RCP_ZYNQ_CONFIGURATION`: It includes the basic configuration of the Zynq UltraScale+ MPSoC, such as the configuration of AXI buses, clocks, and peripherals controlled by the operating system.

To take full advantage of the template provided with the SmartRCP, it is recommended to follow these guidelines:

1. Software requirements:
   - Make sure that you have installed Vitis 2023.2 or a later version. This will ensure full compatibility with the template files and settings.
2. Decompression of the file:
   - The SmartRCP template is delivered in a zipped file. When unzipping the file, the user can rename the top folder to fit their project.
   - It is advisable to decompress the archive on disk C: or in a directory with few hierarchies to avoid problems related to the length of the file paths, since Windows has limitations in this aspect.
3. Example of use:

- Figure 53 shows an example of how to unzip the RCP_TEMPLATE template file and rename it to "USER_GUIDE_EX".

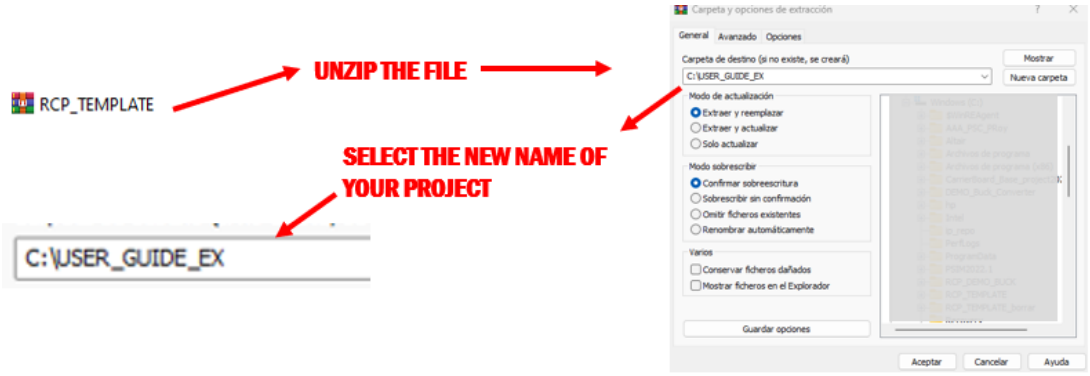


Figure 53: Unzipping the RCP_TEMPLATE file and renaming the folder to USER_GUIDE_EX.

Steps for using and customizing RCP_TEMPLATE according to the needs of each project

1. Decompression and renaming of the project file: Unzip the provided zip file containing the RCP_TEMPLATE. Once unzipped, rename the root file of the folder to match your project.
2. Opening of the project in Vivado: Using Vivado (version 2023.2 or later is recommended), open the Vivado project associated with the RCP_TEMPLATE. This file is named RCP_TEMPLATE.xpr and is located in the RCP_VIVADO folder within the unzipped structure. To open the project, follow the steps shown in Figure 54.

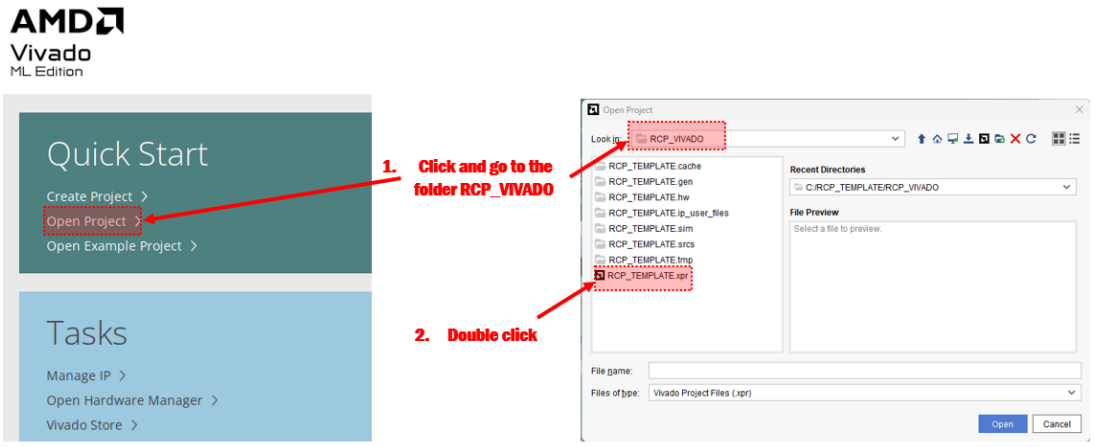


Figure 54: Opening RCP_TEMPLATE.xpr from the RCP_VIVADO folder.

3. Review and initial configuration of the project: Once the project is opened, Vivado will automatically load all the configuration. Since the RCP_TEMPLATE already includes all the necessary SoC (System on Chip) configuration, there is no need to modify the IP block of the Zynq UltraScale+ MPSoC processing system, unless

your design explicitly requires it. Such as changing the synthesis and implementation strategy to prioritize resource savings or propagation timing compliance, adding a new clock line, etc.

4. Customization of the design in the Block Diagram:

   (a) To start customizing the project, it is necessary to open the Block Diagram. To do this, select the **Open Block Design** option in the **Flow Navigator** column under the **IP INTEGRATOR** subsection.

   (b) When opening the Block Diagram, the default structure of the `RCP_TEMPLATE` will be displayed, as shown in Figure 55. The user will have to connect their custom IP blocks to the IPs included in the design.

   (c) These IPs provide access to the SmartRCP peripherals, to the software interface IPs, which allow various functionalities to be controlled from a computer, using the software application provided. In addition, an IP block is included that acts as a datalogger integrated into the FPGA, providing the functionality of an internal FPGA oscilloscope.

   (d) The unconnected signals in the Block Diagram are the ones that the user will be able to use to connect his own IP blocks, as required by his design. On the other hand, the signals that are already connected correspond to the ports that link the block design with the physical pins of the K26 SoC. It is important that the user does not delete these ports, nor modify their connections or names, as they are essential for the correct operation of the system.

   (e) In chapter 5, the characteristics, functionalities and use of each of these IPs will be explained in detail. In Figure 55, a brief introduction to their location within the template and their usefulness for the user is given.

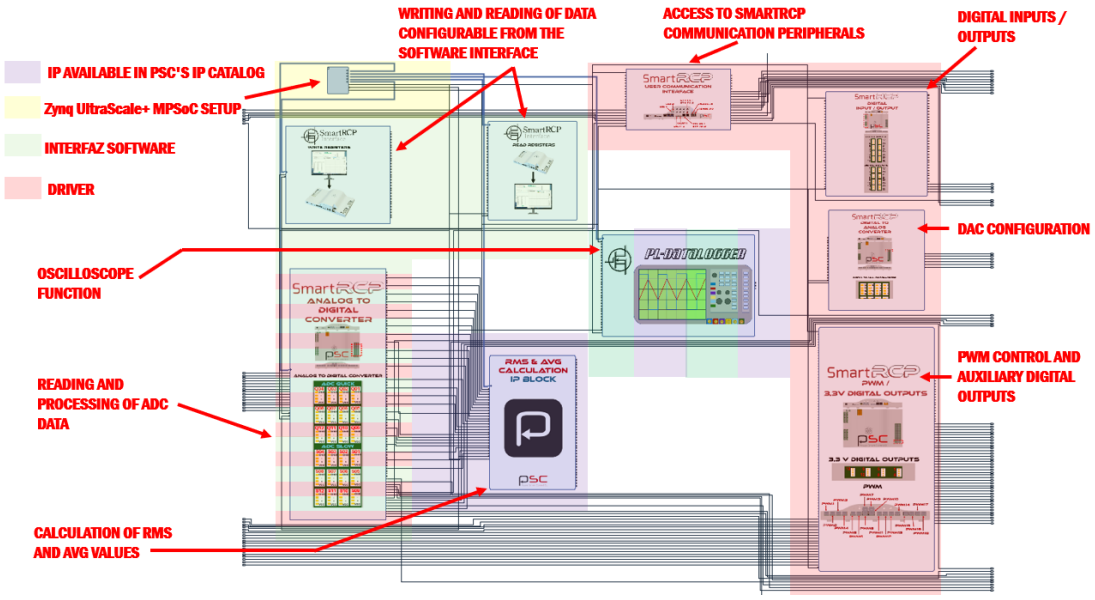


Figure 55: IP Integrator view of the RCP_TEMPLATE Vivado project.

The IP blocks are updated over time. The block symbols and port lists in the screenshots of this chapter and of chapter 5 may therefore differ in detail from the ones a newer template shows; their behaviour is the one documented here.

Advanced configurations on the SoC

Additionally, the user has the option to perform advanced configurations within the **Zynq UltraScale+ MPSoC SETUP** block. These configurations allow customizing various aspects of the K26 SoC, such as:

a) Adding new clock sources: The template provides the 200 MHz main clock used by every driver IP and a second clock output, `pl_clk1` at 100 MHz, together with the AXI PL-PS master interface `AXI HPM0 FPD`. Both are left unconnected on purpose, so that the user can attach their own blocks to them without touching the driver clock domain or its AXI interconnect. Further clock sources can still be added in the MPSoC block if the design requires it.

b) Modifying the characteristics of standard communications: The user has the ability to adjust parameters of interfaces controlled by the microprocessor, such as UART, SPI, CAN, among others. As well as enabling the drivers designed to control them. The default configuration of the `RCP_TEMPLATE` is shown in Figure 56.

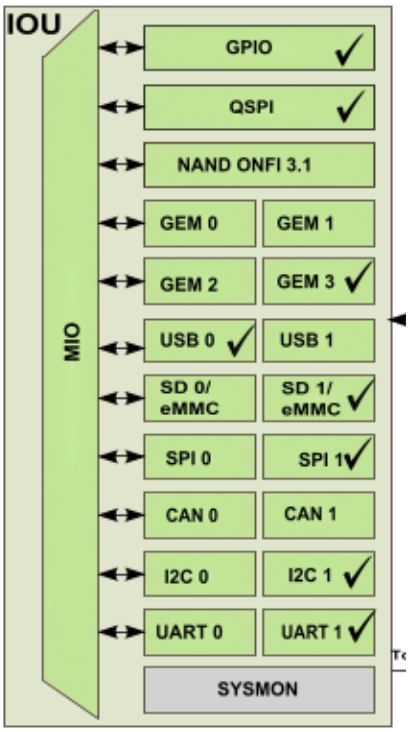


Figure 56: Default MPSoC peripheral configuration (I/O unit of the Zynq UltraScale+ MPSoC).

c) Adjusting advanced parameters of the communication protocols: For example, the Baud Rate of the UART interfaces managed by the SoC operating system can be modified.

These configurations can be made by accessing the properties of the **Zynq UltraScale+ MPSoC** block from the Block Diagram in Vivado (Figure 57).

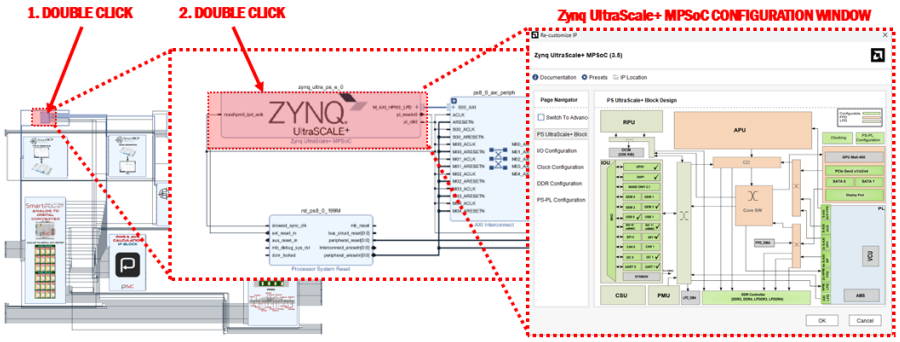


Figure 57: Opening the Zynq UltraScale+ MPSoC configuration window from the Block Diagram.

The driver IPs shipped in the template are documented in chapter 5.

4.3 Xilinx Vitis software tools

4.3.1 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.

4.3.2 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.
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 (Figure 58). Click next.

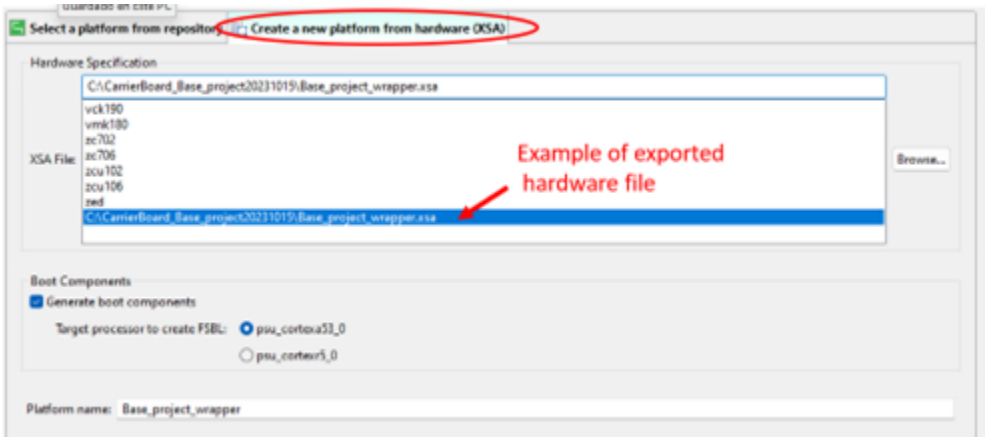


Figure 58: Vitis architecture definition.

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. Core `psu_cortexa53_0` should be avoided when the SmartRCP software interface is used: it is reserved for the interface application (chapter 5). In this example, a Cortex A53 processor has been used and the following names employed (Figure 59):

   The Application Project: APP_example.
   The System project: APP_example_userGuide.

   The screenshot predates the current template: it selects `psu_cortexa53_0`, the core the current release reserves. With the Interface 2024.2 release used in chapter 7 the reserved core was Cortex-R5_0 instead.

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 (Figure 60). Click next.
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.

4.3.3 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.

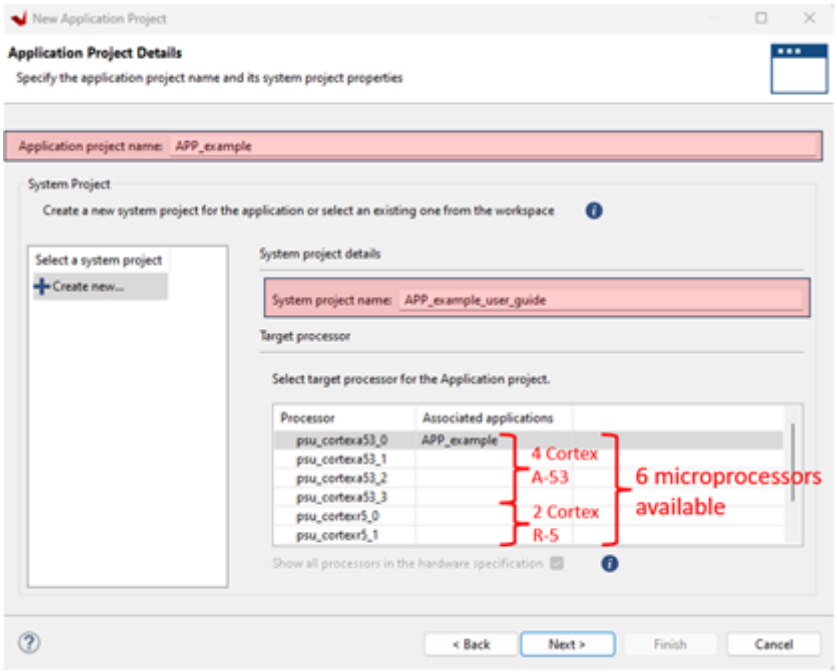


Figure 59: Vitis application names and resources.

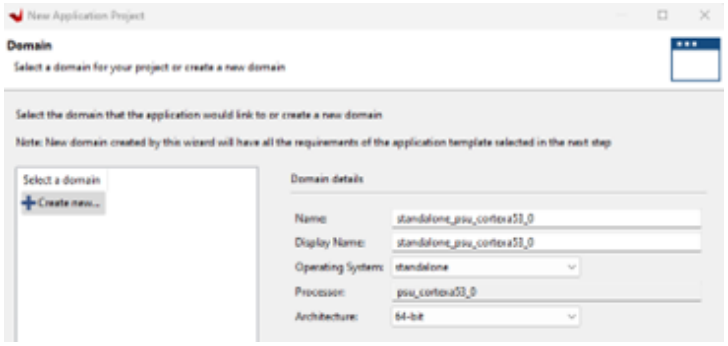


Figure 60: Vitis OS or standalone mode selection.

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 61).

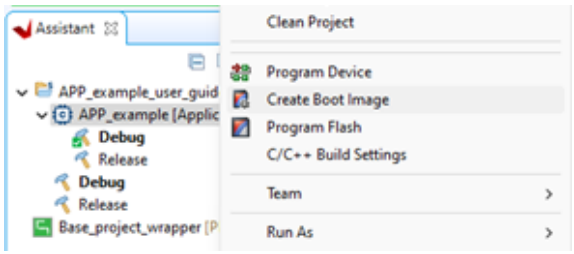


Figure 61: Creating boot image in Vitis.

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.

4.3.4 How to make multi-processor designs

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

1. Open the System Project Settings and click on add **Application Project** (Figure 62).
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 63).

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

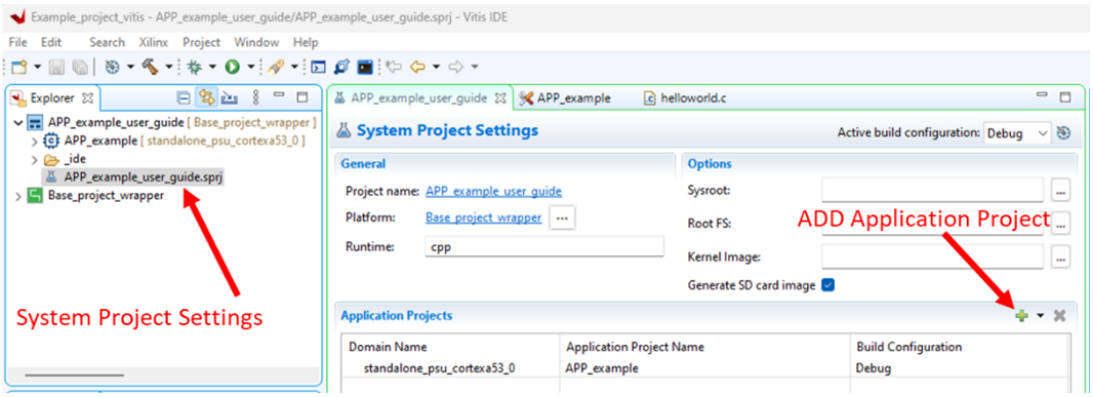


Figure 62: Adding additional application project in Vitis.

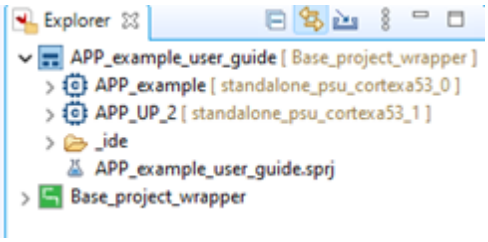


Figure 63: New processor added to the design in Vitis.

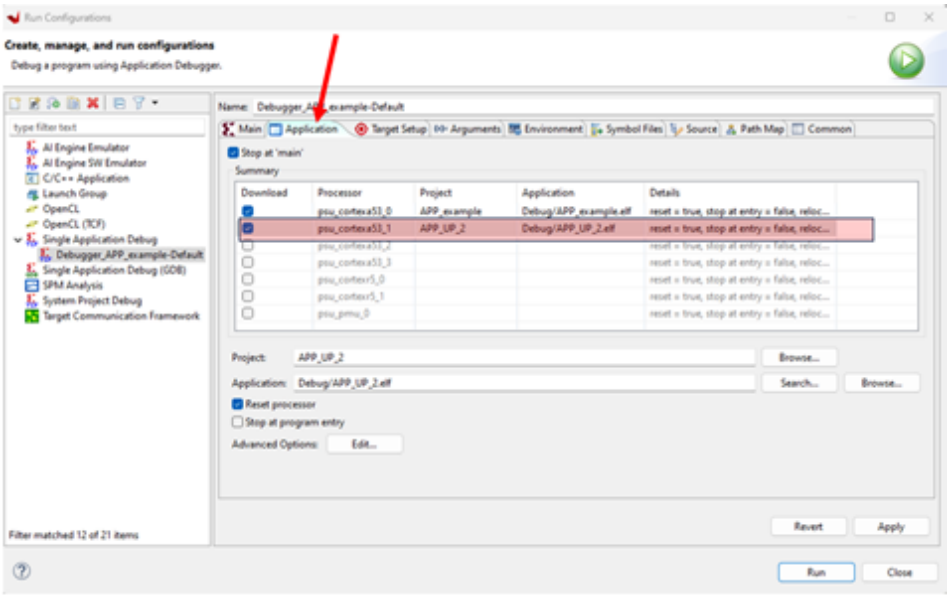


Figure 64: Selecting new processor in Vitis.

5. Go to **Application** and select the new microprocessor (Figure 64).

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 Figure 65.

The available DDR memory segments and its type is shown in Figure 66:

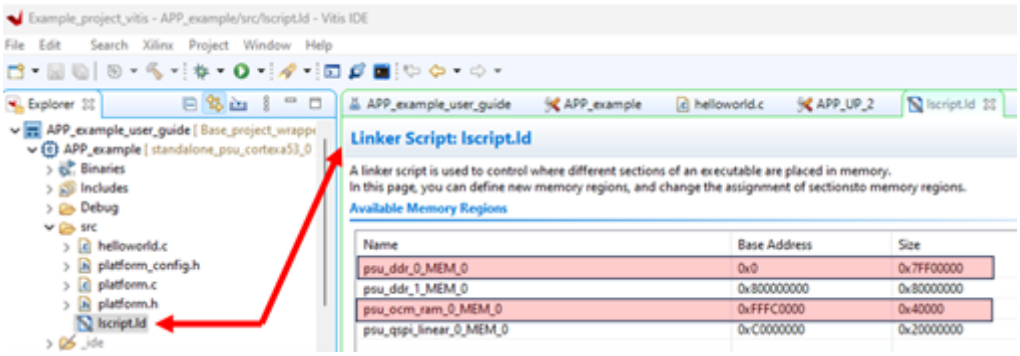


Figure 65: Linker memory positions in Vitis.

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

DDR

OCM

Figure 66: DDR memory available in KRIA K26.

If the user project includes the SmartRCP software interface, its application (`APP_RCP_USER_INTERFACE_A53_0.elf`) runs on `psu_cortexa53_0` and, in the current template, uses the following memory segments:

```
DDR => Base Address = 0x0FFFFFFF | Size = 0x0FFFFFFF
OCM => Base Address = 0xFFFC0000 | Size = 0x4000
```

These regions change between template revisions: check the ones in your own `lscript.ld` before reusing them. The Interface 2024.2 release ran the application on Cortex-R5_0 with DDR base `0x40000000`, size `0x7FE00000`, and the same OCM region (see section 7.6.4).

5 Driver IPs included in the RCP_TEMPLATE

This chapter gives the information a SmartRCP user needs in order to adapt the `RCP_TEMPLATE` project to a specific application: what each IP block delivered with the template does, how it is driven from the user's own logic and which ports it exposes. Every IP is presented with the same three subsections — *Description*, *How to use* and *Connections* — so that a reader who has understood one block finds the same things in the same place in all the others. The figures quoted here (refresh rates, resolutions, thresholds) correspond to the current release of the template.

Both the template and the IP blocks it contains have been designed to operate with a clock of 200 MHz. The use of any other clock value may result in incorrect operation of the IPs included in the template: the system must keep this frequency to avoid operational errors.

Physical ports and the constraints file. Every driver IP has two kinds of ports. **Data** ports (`DATA_*`) are the ones you connect to your own logic. **Physical** ports (`PORT_*`) link the IP to the pins of the K26 SoC through the constraints file `RCP_CONSTRAINTS.xdc`. Do not modify the connections of the `PORT_*` signals and do not rename any port defined by default in the template: those names are what the constraints file matches, and any alteration causes implementation errors or a broken link between the IP and the SoC. If you place one of these drivers in a project that does not start from the template, add `RCP_CONSTRAINTS.xdc` to the project, select the inputs and outputs identified as ports and press **Ctrl+T** (or right-click on them and choose **Make External**) so that they reach the pins. Every section below assumes this has been done.

How the ports are named. Port names follow one pattern, so reading one of them teaches you to read all of them. Take `DATA_DIGITAL_OUT_07_03_J22`: `DATA_` marks a data port (`PORT_` marks a physical one); `DIGITAL_OUT` is the peripheral and the direction of the signal, seen from the board; `07` is the channel number, always two digits starting at `01`; `03` is the position of the pin within its connector; and `J22` is the label silkscreened beside that connector on the board, so the port name tells you where the signal enters or leaves the equipment. Where a channel carries several signals, a middle field says which one (`CH1`, `ENA`, `AD`, `SF...`); each section lists its own. Groups with 32-bit bulk access use the number `00` and the `_ARRAY[31:0]` suffix, with bit 0 as channel 1.

AXI base addresses. Every IP block that talks to the processor has a base address, and Vivado shows it in the **Address Editor**. The ones in the template stay as they come: the remote application looks for each block at its own address. Table 2 lists them.

IP block	Base address	Section
ADC (`SmartRCP_ADc_Driv`)	`0x8000_0000`	5.6
DataLogger (`RCP_DataLogger`)	`0x8000_1000`	5.9
RMS/AVG (`IP_BLOCK_RMS_AVG_v1`)	`0x8000_2000`	5.10
Read registers (`INPUT_USER_REGISTERS`)	`0x8001_0000`	5.8
Write registers (`OUTPUT_USER_REGISTERS`)	`0x8002_0000`	5.7
Interrupt timer 1 (`AXI_Interrupt_TIMPER_1`)	`0x8008_0000`	5.11
…one per timer, 64 K apart…		
Interrupt timer 8 (`AXI_Interrupt_TIMPER_8`)	`0x800F_0000`	5.11

Table 2: AXI base addresses of the RCP_TEMPLATE, as they appear in Vivado's **Address Editor**.

5.1 Vivado standard IP cores

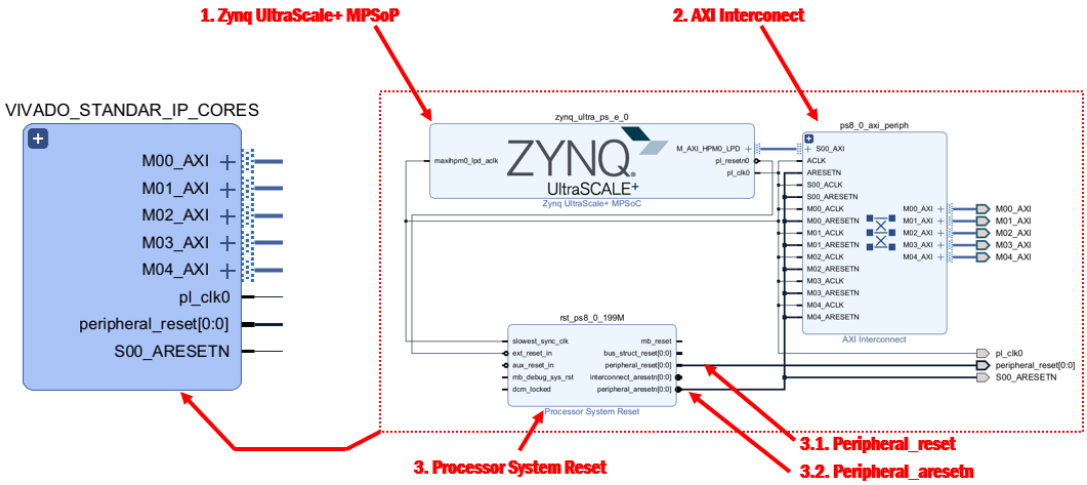


Figure 67: The `VIVADO_STANDAR_IP_CORES` hierarchy of the template and its contents: Zynq UltraScale+ MPSoC (1), AXI Interconnect (2) and Processor System Reset (3), with its `peripheral_reset` (3.1) and `peripheral_aresetn` (3.2) outputs.

5.1.1 Description

Figure 67 shows the minimum set of IP blocks required for any project developed in the Vivado ecosystem. In the template they are grouped in one hierarchy. Among the functions these blocks offer, the ones that matter to the user are:

1. **Reset management block.** It generates the reset signals that must be connected both to the user's IP blocks and to those included in the template, and ensures a correct restart and synchronisation of the system.
2. **Zynq UltraScale+ MPSoC.** It is where the parameters of the K26 SoC are customised: clocks, AXI connections and the peripherals associated with the processing system, among others.

These blocks are the functional basis of any project in the Vivado development environment, and they are what makes the device operable and customisable.

5.1.2 How to use

1. **AXI Interconnect.** This block needs no intervention or configuration by the user. Its function is to connect the processors to the programmable logic. Any new block that requires an AXI line automatically makes a new master output appear on this block, so no manual modification is needed.
2. **Processor System Reset.** This block conditions the reset signals generated by the processors. Two of its outputs are used, and they have opposite polarities:
   - `peripheral_reset` is active when its value is 1: no reset at 0, reset at 1. This polarity is the usual one in IP blocks generated with Vitis HLS.
   - `peripheral_aresetn` is active when its value is 0: no reset at 1, reset at 0. This polarity is the usual one in VHDL designs.
3. **Zynq UltraScale+ MPSoC.** The user can change the basic configuration of the project: modify the clock signal, add clock sources, add AXI interfaces (Figure 68) or include controllers for specific functions such as CAN or I2C. Double-clicking on the block opens the configuration interface. It is not recommended to modify the main clock source (200 MHz), so that the drivers keep working as specified. The template already exposes a second clock, `pl_clk1` at 100 MHz, and the AXI master interface `AXI HPM0 FPD`, both unconnected and free for the user's own blocks; chapter 4 gives the details. For more information on the configuration of this block, see the AMD product guide PG201, https://docs.amd.com/r/en-US/pg201-zynq-ultrascale-plus-processing-system/Features.

5.1.3 Connections

The connections of these blocks are generally handled automatically by Vivado, so no manual intervention is required in most cases. When a new IP block is added, the banner of Figure 69 offers to make the connections automatically; it is recommended to



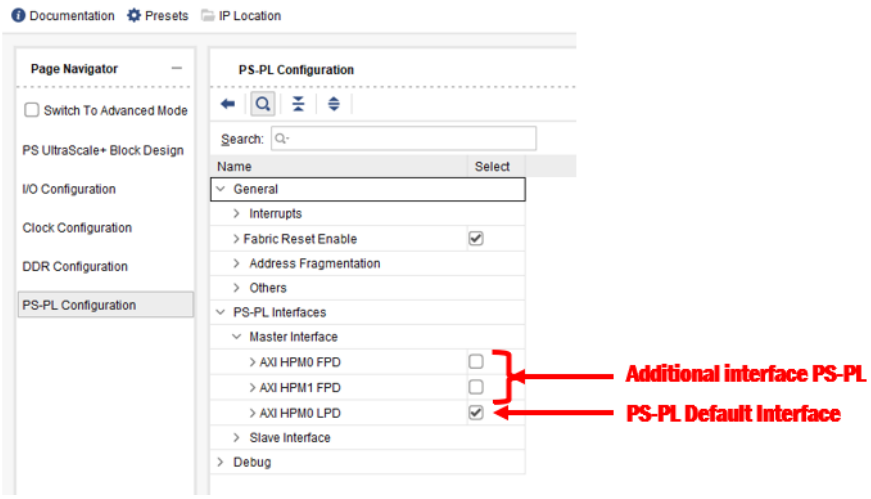


Figure 68: PS-PL configuration page of the Zynq UltraScale+ MPSoC block, where the AXI master interfaces from the processing system to the programmable logic are enabled. `AXI HPM0 LPD` is the one the template's driver IPs use; the screenshot predates the current template, in which `AXI HPM0 FPD` is also enabled and left free for the user.

accept it. In projects with several AXI interfaces or several clock sources, clicking on **Run Connection Automation** opens a window where the user chooses the AXI interface or clock source the new IP block is to be connected to. This procedure keeps the connections between IP blocks consistent.



Figure 69: The **Run Connection Automation** banner Vivado shows after a new IP block is added to the block design.

Table 3 lists what the hierarchy exposes to the rest of the design.

Name	Size	Type	Description	Use
`M00_AXI ... M04_AXI`	AXI	Master	AXI master interfaces of the interconnect, one per AXI slave it serves.	Template
`pl_clk0`	1 bit	Output	200 MHz clock that every IP block of the template runs on.	Template, user
`peripheral_reset[0:0]`	1 bit	Output	Active-high reset.	Template, user
`S00_ARESETN`	1 bit	Output	Active-low reset (`peripheral_aresetn`).	Template, user

Table 3: Ports of the `VIVADO_STANDAR_IP_CORES` hierarchy.

5.2 Communication interfaces: RCP_Comm_Ctrl_V3

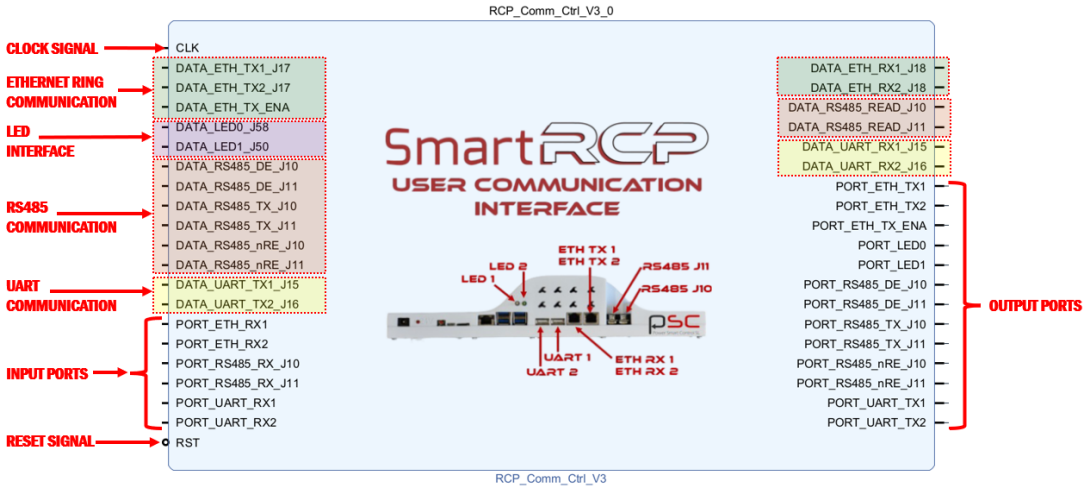


Figure 70: The `RCP_Comm_Ctrl_V3` communications driver. Data ports are grouped by interface: Ethernet Ring, LEDs, RS-485 and UART; the `PORT_*` signals at the bottom are the physical ports.

5.2.1 Description

The `RCP_Comm_Ctrl_V3` block groups all the communication interfaces of the SmartRCP that are driven straight from the programmable logic, with no protocol built on top (the standard interfaces managed directly by the processors — Ethernet, USB and µSD — are not part of it). It gives the user a high level of abstraction: the interfaces are used without managing low-level details such as the pin assignment of the K26 SoC or the logic levels, so the user's logic connects to them directly. As the block symbol in Figure 70 shows, it provides:

- **Ethernet Ring**: two RJ-45 connectors with 2 channels each, transmit on J17 and receive on J18. It is a direct FPGA-speed link, intended for chaining several SmartRCP units together.
- **Two RS-485 interfaces**, J10 and J11, with independent transmit and receive.
- **Two UART interfaces** with physical output over USB, J15 and J16.
- **Two LEDs** for any user-defined purpose.

The purpose of the block is to let the user implement whatever communication protocol the application needs without worrying about the physical management of the interface. Its ports fall into three groups:

1. **Control signals** (`CLK` and `RST`). Already connected in the template; in a new project they are connected automatically with **Run Connection Automation**.

2. **Physical ports** (`PORT_*`). They connect the physical interfaces of the K26 SoC to the logic through `RCP_CONSTRAINTS.xdc`, as explained at the start of the chapter. In the template these connections are already made.
3. **Data signals** (`DATA_*`). Inputs, on the left of the block, carry the data to transmit and the control signals; outputs, on the right, carry the data received from the partner.

Table 4 summarises the driver.

Driver	RCP_Comm_Ctrl_V3
Controls	2 RS-485 interfaces (J10, J11), Ethernet Ring (J17 Tx, J18 Rx), 2 UART outputs over USB (J15, J16) and 2 user LEDs (`LED0` on J58, `LED1` on J50).
Update rate	RS-485: up to 500 kbps in hardware (230,400 baud is the recommended UART maximum). Ethernet Ring: 400 Mbps theoretical per channel, or 12.5 MBaud with an AXI-UART style protocol at 200 MHz. UART over USB: 500,000 baud, 50 kB/s at 8N1.
Limits	RS-485 duplex is fixed in hardware by jumpers JP1/JP2, reachable only with the case off (both ports ship in full duplex). The top speed of the Ethernet Ring depends on the protocol you implement: the manufacturer publishes no reference figure.

Table 4: Communications driver.

5.2.2 How to use

The steps below describe each interface managed by the block. All of them assume the physical ports are left as the template delivers them.

Ethernet Ring

This interface assumes that the user already has a communication protocol that will use the RJ-45 connectors of the Ethernet Ring as its physical layer. Each connector has 2 channels: J17 has two independent channels for transmission and J18 two independent channels for reception. In both directions the channels can be used in parallel and independently of each other: both at once, only one, or none, as the design requires. The RJ-45 connectors are wired as Figure 71 shows: each channel takes two wires, `CH1` on pins 1 and 2 and `CH2` on pins 7 and 8.

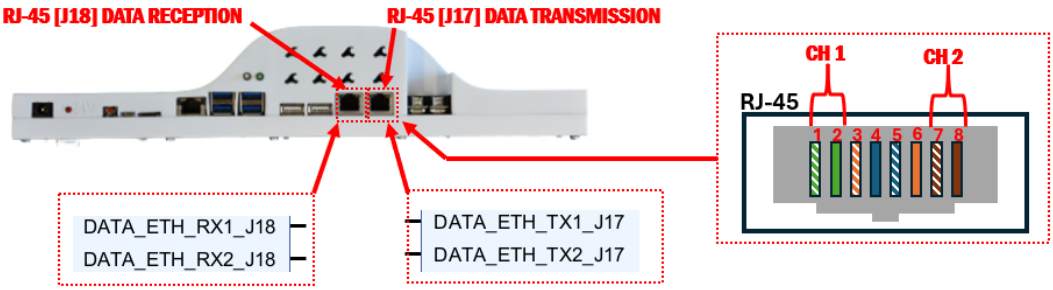
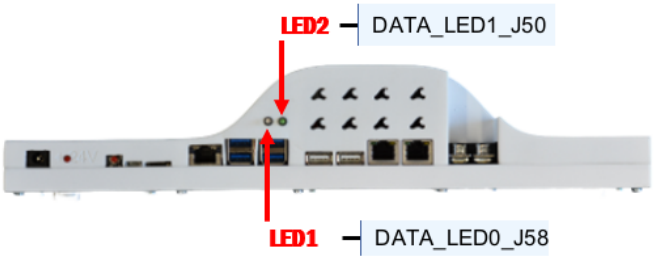


Figure 71: Ethernet Ring: J18 receives and J17 transmits, and each connector carries two channels; the RJ-45 detail shows the pins of `CH1` (1 and 2) and `CH2` (7 and 8).

1. **Enable transmission.** The user's IP must drive a signal that enables data transmission, or it can come from another IP block, or a constant can be connected to keep transmission always enabled. It goes to `DATA_ETH_TX_ENA`: `'1'` enables the transmitter, `'0'` disables it.
2. **Data to transmit.** Connect the data output of the user's IP block to `DATA_ETH_TX1_J17` or `DATA_ETH_TX2_J17`. The driver acts as the intermediary between the user's logic and the physical interface.
3. **Data received.** Connect `DATA_ETH_RX1_J18` or `DATA_ETH_RX2_J18` to the data input of the user's IP block, so that the received data reaches the design.

User LEDs

The LEDs are controlled directly: the user connects the signal that is to drive each LED to the corresponding input of the driver, `DATA_LED0_J58` or `DATA_LED1_J50` (Figure 72). Each LED is independent and lights when its signal is `'1'`.



Signal Value	Status
0	OFF
1	ON

Figure 72: The two user LEDs on the front of the SmartRCP and the data input that drives each one; `'1'` switches the LED on.

RS-485

Either of the two RS-485 interfaces can be used independently. The protocol the user implements must handle the four signals of each interface (Figure 73):

- `nRE` (receiver enable), an input to the driver generated by the user's IP. Active low: `'0'` enables the receiver, `'1'` disables it.
- `DE` (driver enable), an input generated by the user's IP that controls the differential outputs A and B of the transceiver. `'1'` enables them and allows transmission; `'0'` puts them in high impedance.
- `TX`, the data to transmit, generated by the user's IP and connected to the driver input.
- `READ`, the data received. It is an output of the driver and must be connected to the user's IP.

Full or half duplex is selected in hardware with jumpers JP1 and JP2.

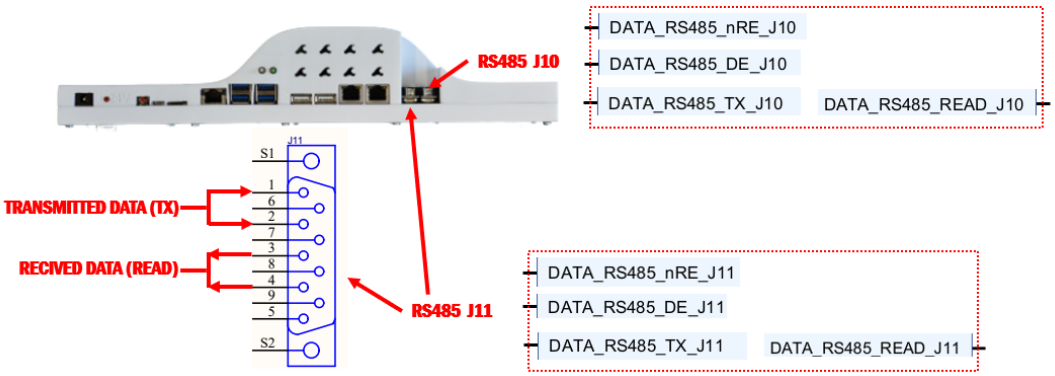


Figure 73: RS-485 interfaces J10 and J11 and their data ports; the J11 detail marks the transmit and receive pairs on the connector.

UART

The SmartRCP offers two UART interfaces, physically brought out through two USB connectors: `UART1` on J15 and `UART2` on J16 (Figure 74). They can be used in parallel and independently — both, one or none — with the user's own serial protocol or through Vivado's `AXI UART LITE` block. For each UART in use:

1. **Transmission (TX).** Connect the user's transmit signal to `DATA_UART_TX1_J15` or `DATA_UART_TX2_J16`.

2. **Reception (RX).** Connect `DATA_UART_RX1_J15` or `DATA_UART_RX2_J16` to the user's receive input.

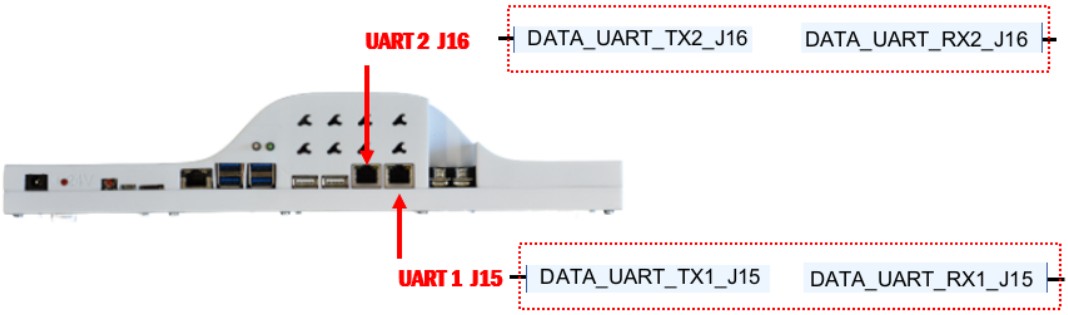


Figure 74: UART interfaces over USB, `UART1` on J15 and `UART2` on J16, and the transmit and receive port of each one.

5.2.3 Connections

Table 5 lists the data and control ports of the block, grouped by interface, and Table 6 its physical ports.

Name	Size	Type	Description	Use
DATA_ETH_TX1_J17, DATA_ETH_TX2_J17	1 bit	Input	Ethernet Ring: data to transmit, one per channel of J17.	User
DATA_ETH_TX_ENA	1 bit	Input	Ethernet Ring: transmit enable, shared by both channels.	User
DATA_ETH_RX1_J18, DATA_ETH_RX2_J18	1 bit	Output	Ethernet Ring: data received, one per channel of J18.	User
DATA_LED0_J58, DATA_LED1_J50	1 bit	Input	User LEDs: LED data input ('1' means on).	User
DATA_RS485_DE_J10, DATA_RS485_DE_J11	1 bit	Input	RS-485: driver enable of each transceiver.	User
DATA_RS485_TX_J10, DATA_RS485_TX_J11	1 bit	Input	RS-485: data to transmit on each connector.	User
DATA_RS485_nRE_J10, DATA_RS485_nRE_J11	1 bit	Input	RS-485: receiver enable of each transceiver, active low.	User
DATA_RS485_READ_J10, DATA_RS485_READ_J11	1 bit	Output	RS-485: data received on each connector.	User
DATA_UART_TX1_J15, DATA_UART_TX2_J16	1 bit	Input	UART: data to transmit on each interface.	User
DATA_UART_RX1_J15, DATA_UART_RX2_J16	1 bit	Output	UART: data received on each interface.	User
CLK, RST	1 bit	Input	200 MHz clock and reset, already connected in the template.	Template

Table 5: Data and control ports of the RCP_Comm_Ctrl_V3 driver.

Name	Size	Type	Description	Use
PORT_ETH_RX1, PORT_ETH_RX2	1 bit	Input	Ethernet Ring, to the SoC pins.	HW port
PORT_ETH_TX1, PORT_ETH_TX2, PORT_ETH_TX_ENA	1 bit	Output	Ethernet Ring, to the SoC pins.	HW port
PORT_LED0, PORT_LED1	1 bit	Output	LEDs, to the SoC pins.	HW port
PORT_RS485_RX_J10, PORT_RS485_RX_J11	1 bit	Input	RS-485, to the SoC pins.	HW port
PORT_RS485_DE_J10, PORT_RS485_DE_J11, PORT_RS485_TX_J10, PORT_RS485_TX_J11, PORT_RS485_nRE_J10, PORT_RS485_nRE_J11	1 bit	Output	RS-485, to the SoC pins.	HW port
PORT_UART_RX1, PORT_UART_RX2	1 bit	Input	UART, to the SoC pins.	HW port
PORT_UART_TX1, PORT_UART_TX2	1 bit	Output	UART, to the SoC pins.	HW port

Table 6: Physical ports of the RCP_Comm_Ctrl_V3 driver.



5.3 Digital inputs and outputs: DIGITAL_INOUT_DRIV_V1

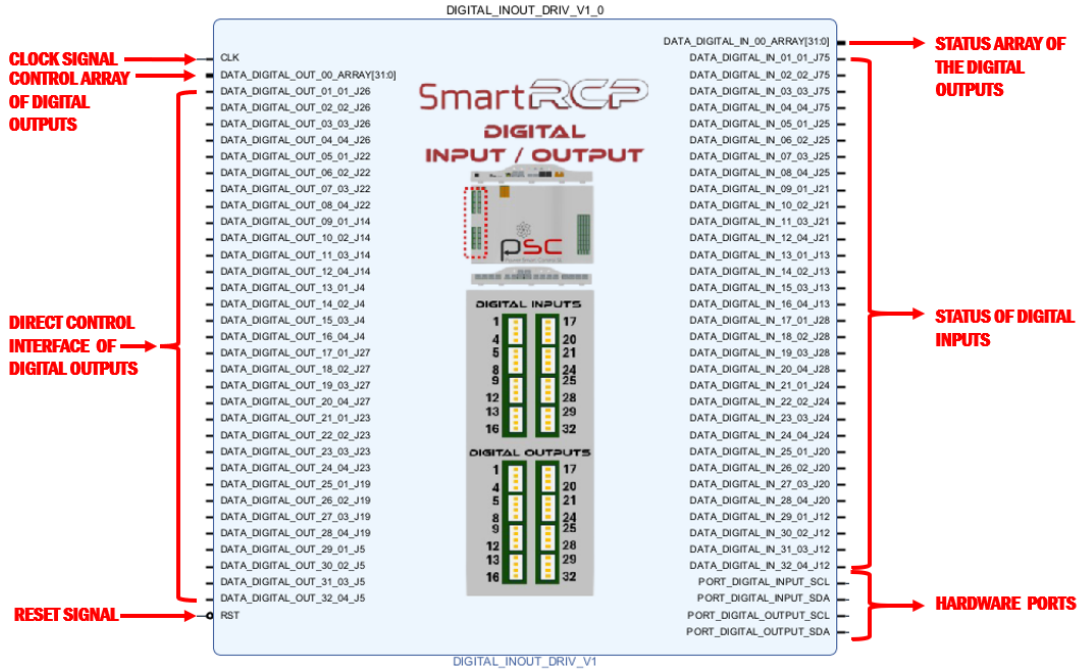


Figure 75: The `DIGITAL_INOUT_DRIV_V1` digital I/O driver. On the left, the 32 data inputs that drive the board's outputs; on the right, the 32 data outputs carrying the state of its inputs and the status array `DATA_DIGITAL_IN_00_ARRAY`.

5.3.1 Description

The `DIGITAL_INOUT_DRIV_V1` driver gives a simple interface to drive the digital outputs and read the digital inputs of the SmartRCP without managing the low-level details of the hardware, so that the user concentrates on the logic of the application. The SmartRCP has 32 isolated digital inputs and 32 isolated digital outputs, all at 24 V and all independent of one another.

- **Refresh rate.** Inputs are read at 4 kHz (every 250 µs); outputs are updated at 2 kHz (every 500 µs).
- **Clock requirement.** The driver requires a 200 MHz clock. Other frequencies cause timing problems.
- **Input signals.** An input reads 1 when the pin is high and 0 when it is low. The high level is guaranteed from 21 V (24 V logic). Every input carries a first-order 1 kHz low-pass filter and a TVS diode: it is that filter, not the 4 kHz sampling, that sets the real response time.
- **Output signals.** The output voltage is 0 V for a logic 0 and 24 V for a logic 1, with a maximum output current of 40 mA per channel.

Table 7 summarises the driver.

Driver	DIGITAL_INOUT_DRIV_V1
Controls	32 independent 24 V digital inputs and 32 digital outputs, plus 32-bit bulk access to each group.
Update rate	Inputs: 4 kHz (250 µs). Outputs: 2 kHz (500 µs).
Limits	High level guaranteed from 21 V. Maximum output current 40 mA per channel. Every input carries a first-order 1 kHz low-pass filter and a TVS diode. Writes through the 32-bit word and through the individual pin are combined with a logical OR.

Table 7: 24 V digital I/O driver.

5.3.2 How to use

The physical ports of this driver are the four I2C lines listed in the Connections subsection; leave them as the template delivers them. Figure 76 shows how the channel numbers of the port names map onto the pin headers of the board: four channels per connector, in the order printed on the housing.

Digital inputs (0–24 V). The user has two ways of reading the state of the input pins:

1. **32-bit word**, `DATA_DIGITAL_IN_00_ARRAY[31:0]`. It delivers the state of the 32 inputs in one register; the least significant bit is input 1 and the most significant bit is input 32.
2. **One output per pin**, `DATA_DIGITAL_IN_<ID>_<NN>_J<xx>`, a 1-bit output per input that reads 0 while the pin is low and 1 while it is high (guaranteed from 21 V). `<ID>` is the channel number as printed in Figure 76, `<NN>` the position of the pin within its header and `<xx>` the header it belongs to.

Digital outputs (0–24 V). There are also two ways of setting the outputs, and both can be used at the same time. If they are, the final value of each pin is the logical OR of what the two methods assign to it.

1. **32-bit word**, `DATA_DIGITAL_OUT_00_ARRAY[31:0]`. All 32 outputs are set at once; the least significant bit drives output 1 and the most significant bit drives output 32. A logic 1 sets 24 V on the pin and a logic 0 sets 0 V.
2. **One input per pin**, `DATA_DIGITAL_OUT_<ID>_<NN>_J<xx>`, with the same meaning of the fields as for the inputs. A logic 1 gives 24 V and a logic 0 gives 0 V.

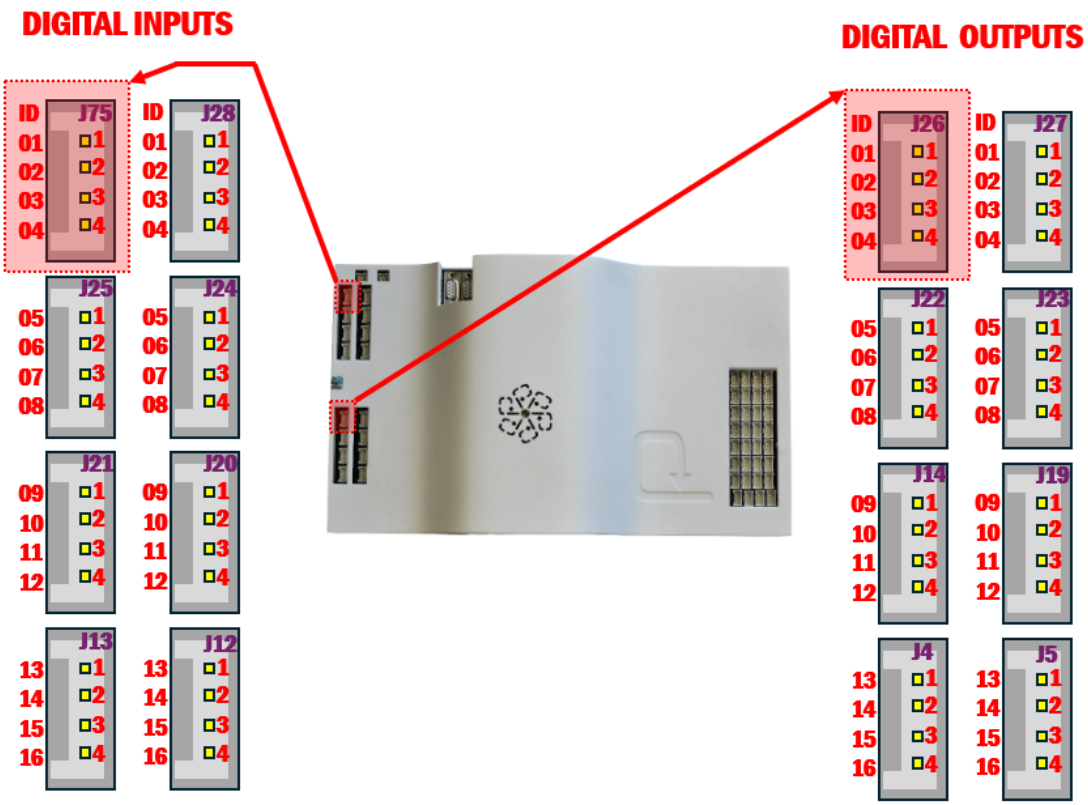


Figure 76: Pin headers of the 32 digital inputs (left) and 32 digital outputs (right): each header carries four channels, and the channel number (*ID*) and header label are the ones that appear in the port names.

An output that is not going to be used can be left unconnected, but synthesis and implementation then raise warnings that can mask real ones. Tie unused inputs of the driver to a constant 0 instead.

5.3.3 Connections

Table 8 lists the port groups of the block and Table 9 the full name of each channel.

Name	Size	Type	Description	Use
DATA_DIGITAL_OUT_00_ARRAY[31:0]	32 bit	Input	Sets all the digital outputs at once.	User
DATA_DIGITAL_OUT_01_01_J26 ... DATA_DIGITAL_OUT_32_04_J5	1 bit	Input	Sets one digital output; 32 ports.	User
DATA_DIGITAL_IN_00_ARRAY[31:0]	32 bit	Output	State of all the digital inputs at once.	User
DATA_DIGITAL_IN_01_01_J75 ... DATA_DIGITAL_IN_32_04_J12	1 bit	Output	State of one digital input; 32 ports.	User
PORT_DIGITAL_INPUT_SCL, PORT_DIGITAL_INPUT_SDA, PORT_DIGITAL_OUTPUT_SCL, PORT_DIGITAL_OUTPUT_SDA	1 bit	I2C	Connection to the physical pins of the SoC.	HW port
CLK, RST	1 bit	Input	200 MHz clock and reset, already connected in the template.	Template

Table 8: Connections of the DIGITAL_INOUT_DRIV_V1 driver.

ID	Output data port (input to the driver)	Input data port (output of the driver)
01	DATA_DIGITAL_OUT_01_01_J26	DATA_DIGITAL_IN_01_01_J75
02	DATA_DIGITAL_OUT_02_02_J26	DATA_DIGITAL_IN_02_02_J75
03	DATA_DIGITAL_OUT_03_03_J26	DATA_DIGITAL_IN_03_03_J75
04	DATA_DIGITAL_OUT_04_04_J26	DATA_DIGITAL_IN_04_04_J75
05	DATA_DIGITAL_OUT_05_01_J22	DATA_DIGITAL_IN_05_01_J25
06	DATA_DIGITAL_OUT_06_02_J22	DATA_DIGITAL_IN_06_02_J25
07	DATA_DIGITAL_OUT_07_03_J22	DATA_DIGITAL_IN_07_03_J25
08	DATA_DIGITAL_OUT_08_04_J22	DATA_DIGITAL_IN_08_04_J25
09	DATA_DIGITAL_OUT_09_01_J14	DATA_DIGITAL_IN_09_01_J21
10	DATA_DIGITAL_OUT_10_02_J14	DATA_DIGITAL_IN_10_02_J21
11	DATA_DIGITAL_OUT_11_03_J14	DATA_DIGITAL_IN_11_03_J21
12	DATA_DIGITAL_OUT_12_04_J14	DATA_DIGITAL_IN_12_04_J21
13	DATA_DIGITAL_OUT_13_01_J4	DATA_DIGITAL_IN_13_01_J13
14	DATA_DIGITAL_OUT_14_02_J4	DATA_DIGITAL_IN_14_02_J13
15	DATA_DIGITAL_OUT_15_03_J4	DATA_DIGITAL_IN_15_03_J13
16	DATA_DIGITAL_OUT_16_04_J4	DATA_DIGITAL_IN_16_04_J13
17	DATA_DIGITAL_OUT_17_01_J27	DATA_DIGITAL_IN_17_01_J28
18	DATA_DIGITAL_OUT_18_02_J27	DATA_DIGITAL_IN_18_02_J28
19	DATA_DIGITAL_OUT_19_03_J27	DATA_DIGITAL_IN_19_03_J28
20	DATA_DIGITAL_OUT_20_04_J27	DATA_DIGITAL_IN_20_04_J28
21	DATA_DIGITAL_OUT_21_01_J23	DATA_DIGITAL_IN_21_01_J24
22	DATA_DIGITAL_OUT_22_02_J23	DATA_DIGITAL_IN_22_02_J24
23	DATA_DIGITAL_OUT_23_03_J23	DATA_DIGITAL_IN_23_03_J24
24	DATA_DIGITAL_OUT_24_04_J23	DATA_DIGITAL_IN_24_04_J24
25	DATA_DIGITAL_OUT_25_01_J19	DATA_DIGITAL_IN_25_01_J20
26	DATA_DIGITAL_OUT_26_02_J19	DATA_DIGITAL_IN_26_02_J20
27	DATA_DIGITAL_OUT_27_03_J19	DATA_DIGITAL_IN_27_03_J20
28	DATA_DIGITAL_OUT_28_04_J19	DATA_DIGITAL_IN_28_04_J20
29	DATA_DIGITAL_OUT_29_01_J5	DATA_DIGITAL_IN_29_01_J12
30	DATA_DIGITAL_OUT_30_02_J5	DATA_DIGITAL_IN_30_02_J12
31	DATA_DIGITAL_OUT_31_03_J5	DATA_DIGITAL_IN_31_03_J12
32	DATA_DIGITAL_OUT_32_04_J5	DATA_DIGITAL_IN_32_04_J12

Table 9: Data ports of the 32 digital channels, as named on the block.



5.4 PWM and 3.3 V digital outputs: RCP_PWM_DRIV_v1

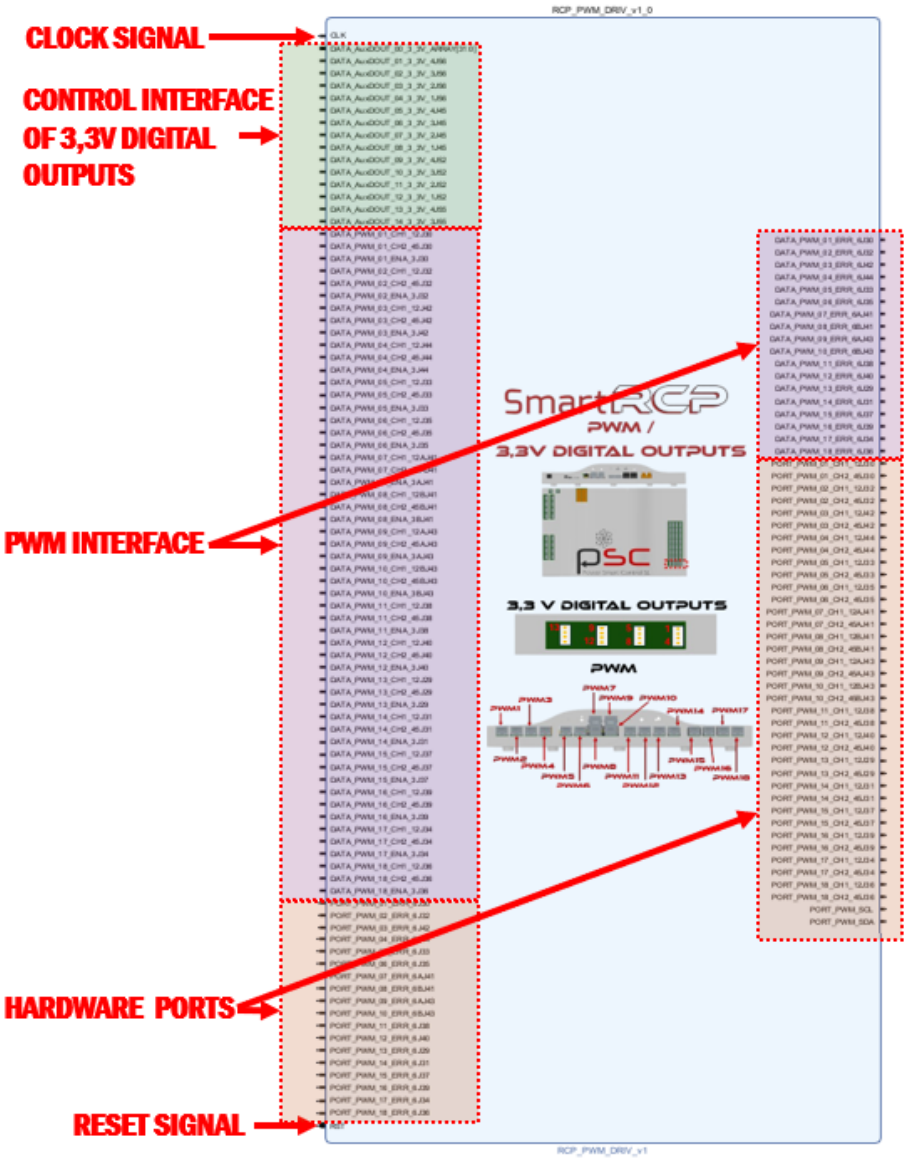


Figure 77: The `RCP_PWM_DRIV_v1` PWM and 3.3 V digital output driver. From top to bottom on the left: clock, the 3.3 V output inputs, the PWM inputs and the physical ports; on the right, the fault outputs and the remaining physical ports.

5.4.1 Description

This driver gives a high-level interface to the PWM outputs of the SmartRCP — signal generation, output enable and fault detection — and to its auxiliary 3.3 V digital outputs, so that the user concentrates on the application logic. Both groups share a ground isolated from the rest of the board.

- **3.3 V digital outputs.** The driver gives access to 14 digital outputs with levels 0–3.3 V (logic 0 and 1). They are updated at 2 kHz (every 500 µs). These outputs come straight off the FPGA, with no analog front end and no short-circuit protection. Figure 78 shows how the driver inputs correspond to the pin headers of the board.
- **PWM.** The SmartRCP has 18 RJ-45 PWM ports, each with 2 independently controllable PWM signals: 36 PWM signals in all. Each port is meant to drive one half-bridge, so besides the two channels it carries an **enable** signal towards the gate driver, updated at 2 kHz (500 µs) because it is multiplexed, and an **error** signal coming back from it. If the driver under control has protections and a fault output, the error signal reflects its state: it reads high while the gate driver reports a fault, it arrives directly on a programmable-logic pin, and its resolution inside the FPGA is therefore 5 ns, with a 50 ns minimum readable pulse. Figure 79 summarises the control interface of each PWM port and the pin order of its RJ-45.
- **Clock requirement.** The driver must be fed with the 200 MHz clock; another frequency alters the timing of every signal.

The template presents 18 PWM ports while the board labels 16 connector identifiers, J29 to J44: J41 and J43 are dual RJ-45 housings with sub-ports A and B, and the port names carry that letter (`AJ41`, `BJ41`, `AJ43`, `BJ43`).

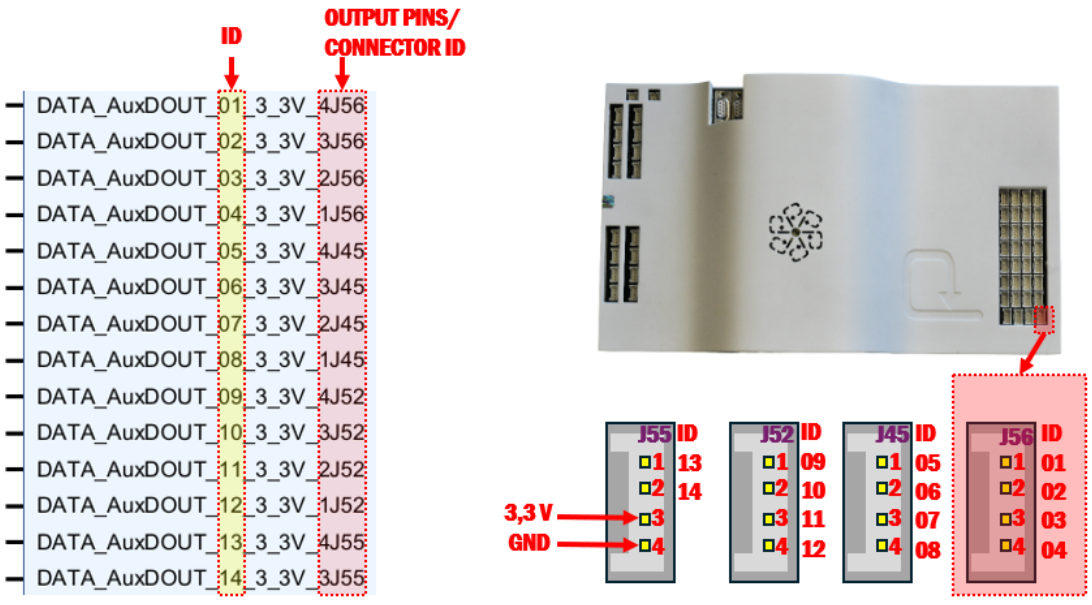


Figure 78: The 14 auxiliary 3.3 V digital outputs: the channel number (*ID*) and the pin and header of each one appear in the port name; J55 also provides 3.3 V and ground.

Table 10 summarises the driver.

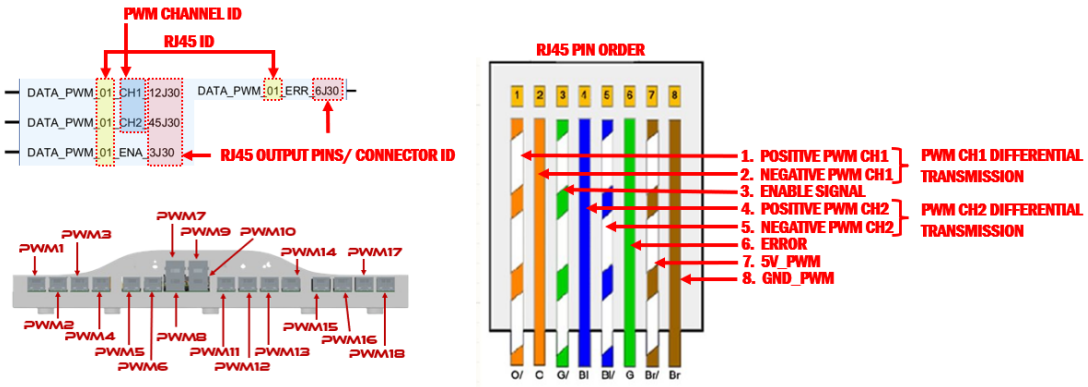


Figure 79: Control interface of one PWM port: the port names carry the channel and the RJ-45 pins, and the 18 ports are located on the top of the unit; the pin order of the RJ-45 is `CH1` on pins 1–2, enable on 3, `CH2` on 4–5, error on 6, 5 V on 7 and ground on 8.

Driver	RCP_PWM_DRIV_v1
Controls	18 PWM outputs (half-bridge, 2 channels each) and 14 auxiliary 3.3 V digital outputs.
Update rate	PWM timing resolution: 5 ns. Enable and 3.3 V outputs: 2 kHz (500 µs). Fault signal: 5 ns resolution inside the FPGA, with a 50 ns minimum readable pulse.
Limits	Minimum usable PWM pulse width: 50 ns, set by the isolation chain and the distortion of the differential driver. Isolation: 4242 Vpk transient, 6000 Vpk surge, 1414 Vpk repetitive, over 8 mm of creepage and clearance. **The 3.3 V outputs come straight off the FPGA**, with no analog front end and no short-circuit protection.

Table 10: PWM and 3.3 V digital output driver.

5.4.2 How to use

The physical ports of this driver are the `PORT_PWM_*` signals; leave them as the template delivers them.

3.3 V digital outputs. There are two ways of setting these outputs, and both can be used at the same time; if they are, the final value of each pin is the logical OR of what the two methods assign to it.

1. **32-bit word**, `DATA_AuxDOUT_00_3_3V_ARRAY[31:0]`. All outputs are set at once; the least significant bit drives output 1 and bit 13 (`DATA_AuxDOUT_00_3_3V_ARRAY[13]`) drives output 14. A logic 1 sets 3.3 V on the pin and a logic 0 sets 0 V.

2. **One input per pin**, `DATA_AuxDOUT_<ID>_3_3V_<N>J<xx>`, where `<ID>` is the channel number of Figure 78, `<N>` the pin within the header and `<xx>` the header. A logic 1 gives 3.3 V and a logic 0 gives 0 V.

Unused outputs can be left unconnected, at the cost of synthesis warnings that can mask real ones; tie them to a constant 0 instead.

PWM. The user's own IP generates the PWM signals and connects them to the RJ-45 output ports through the driver inputs. For each port in use:

- **PWM generation.** Implement in the user's IP the logic that generates the two PWM signals of the port, and connect them to `DATA_PWM_<ID>_CH1_12J<xx>` and `DATA_PWM_<ID>_CH2_45J<xx>`. The digits before the `J` are the RJ-45 pins the signal leaves on. The minimum usable pulse width is 50 ns.
- **Enable.** If the gate driver connected to the RJ-45 needs an enable signal, generate it in the user's IP and connect it to `DATA_PWM_<ID>_ENA_3J<xx>`.
- **Error handling.** If the hardware under control (the driver that fires the semiconductor) has a fault detection mechanism, connect the user's fault handling logic to `DATA_PWM_<ID>_ERR_6J<xx>`, the output through which the driver delivers that signal.

5.4.3 Connections

Table 11 lists the port groups of the block and Table 12 the full name of the four data ports of every PWM port.

Name	Size	Type	Description	Use
DATA_AuxDOUT_00_3_3V_ARRAY[31:0]	32 bit	Input	Sets the 14 auxiliary 3.3 V outputs at once (bits 0 to 13).	User
DATA_AuxDOUT_01_3_3V_4J56 ... DATA_AuxDOUT_14_3_3V_3J55	1 bit	Input	Sets one auxiliary 3.3 V output; 14 ports (Figure 78).	User
DATA_PWM_<ID>_CH1_12J<xx>, DATA_PWM_<ID>_CH2_45J<xx>, DATA_PWM_<ID>_ENA_3J<xx>	1 bit	Input	PWM channels and enable of each of the 18 ports (Table 12).	User
DATA_PWM_<ID>_ERR_6J<xx>	1 bit	Output	Fault signal returned by the gate driver of each port.	User
PORT_PWM_<ID>_ERR_6J<xx>	1 bit	Input	Connection to the physical pins of the SoC; 18 ports.	HW port
PORT_PWM_<ID>_CH1_12J<xx>, PORT_PWM_<ID>_CH2_45J<xx>	1 bit	Output	Connection to the physical pins of the SoC; 36 ports.	HW port
PORT_PWM_SCL, PORT_PWM_SDA	1 bit	I2C	Connection to the physical pins of the SoC.	HW port
CLK, RST	1 bit	Input	200 MHz clock and reset, already connected in the template.	Template

Table 11: Connections of the RCP_PWM_DRIV_v1 driver.

ID	Channel 1	Channel 2	Enable	Error
01	DATA_PWM_01_CH1_12J30	DATA_PWM_01_CH2_45J30	DATA_PWM_01_ENA_3J30	DATA_PWM_01_ERR_6J30
02	DATA_PWM_02_CH1_12J32	DATA_PWM_02_CH2_45J32	DATA_PWM_02_ENA_3J32	DATA_PWM_02_ERR_6J32
03	DATA_PWM_03_CH1_12J42	DATA_PWM_03_CH2_45J42	DATA_PWM_03_ENA_3J42	DATA_PWM_03_ERR_6J42
04	DATA_PWM_04_CH1_12J44	DATA_PWM_04_CH2_45J44	DATA_PWM_04_ENA_3J44	DATA_PWM_04_ERR_6J44
05	DATA_PWM_05_CH1_12J33	DATA_PWM_05_CH2_45J33	DATA_PWM_05_ENA_3J33	DATA_PWM_05_ERR_6J33
06	DATA_PWM_06_CH1_12J35	DATA_PWM_06_CH2_45J35	DATA_PWM_06_ENA_3J35	DATA_PWM_06_ERR_6J35
07	DATA_PWM_07_CH1_12AJ41	DATA_PWM_07_CH2_45AJ41	DATA_PWM_07_ENA_3AJ41	DATA_PWM_07_ERR_6AJ41
08	DATA_PWM_08_CH1_12BJ41	DATA_PWM_08_CH2_45BJ41	DATA_PWM_08_ENA_3BJ41	DATA_PWM_08_ERR_6BJ41
09	DATA_PWM_09_CH1_12AJ43	DATA_PWM_09_CH2_45AJ43	DATA_PWM_09_ENA_3AJ43	DATA_PWM_09_ERR_6AJ43
10	DATA_PWM_10_CH1_12BJ43	DATA_PWM_10_CH2_45BJ43	DATA_PWM_10_ENA_3BJ43	DATA_PWM_10_ERR_6BJ43
11	DATA_PWM_11_CH1_12J38	DATA_PWM_11_CH2_45J38	DATA_PWM_11_ENA_3J38	DATA_PWM_11_ERR_6J38
12	DATA_PWM_12_CH1_12J40	DATA_PWM_12_CH2_45J40	DATA_PWM_12_ENA_3J40	DATA_PWM_12_ERR_6J40
13	DATA_PWM_13_CH1_12J29	DATA_PWM_13_CH2_45J29	DATA_PWM_13_ENA_3J29	DATA_PWM_13_ERR_6J29
14	DATA_PWM_14_CH1_12J31	DATA_PWM_14_CH2_45J31	DATA_PWM_14_ENA_3J31	DATA_PWM_14_ERR_6J31
15	DATA_PWM_15_CH1_12J37	DATA_PWM_15_CH2_45J37	DATA_PWM_15_ENA_3J37	DATA_PWM_15_ERR_6J37
16	DATA_PWM_16_CH1_12J39	DATA_PWM_16_CH2_45J39	DATA_PWM_16_ENA_3J39	DATA_PWM_16_ERR_6J39
17	DATA_PWM_17_CH1_12J34	DATA_PWM_17_CH2_45J34	DATA_PWM_17_ENA_3J34	DATA_PWM_17_ERR_6J34
18	DATA_PWM_18_CH1_12J36	DATA_PWM_18_CH2_45J36	DATA_PWM_18_ENA_3J36	DATA_PWM_18_ERR_6J36

Table 12: Data ports of the 18 PWM ports, as named on the block. The physical ports PORT_PWM_* repeat the channel 1, channel 2 and error names with the PORT_ prefix.

5.5 Digital-to-analog converters: RCP_DAC_DRIV_v1



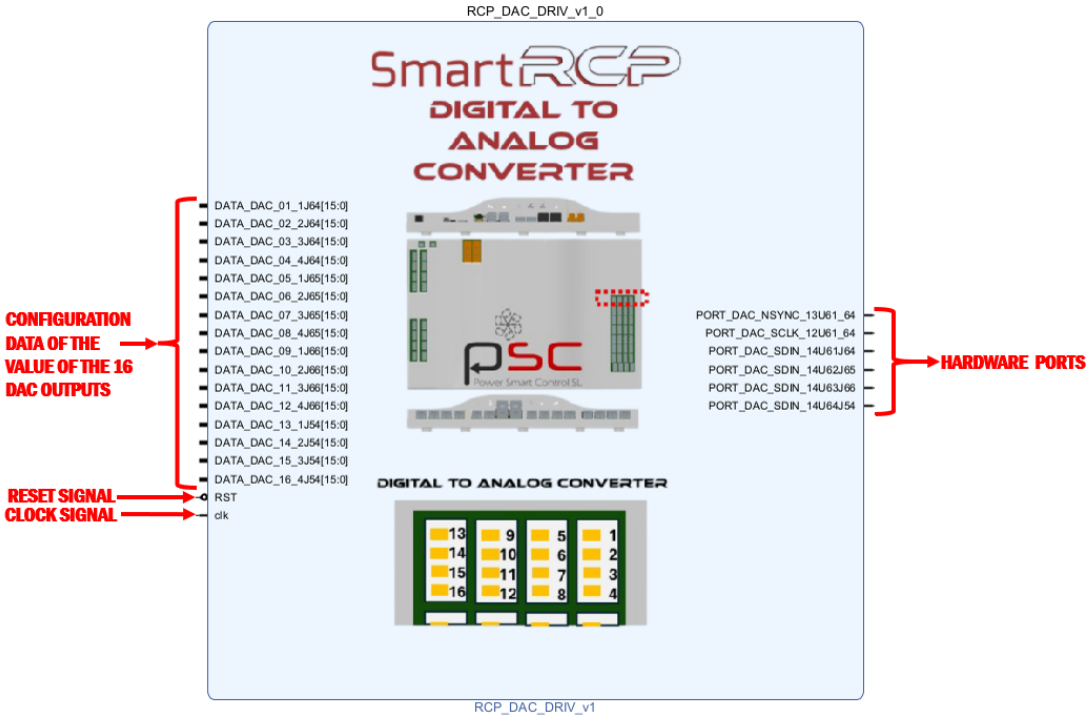


Figure 80: The `RCP_DAC_DRIV_v1` digital-to-analog driver: 16 data inputs, one per channel, the physical ports on the right and a detail of the board's analog output connector.

5.5.1 Description

The SmartRCP has 16 analog outputs, driven and configured through the `RCP_DAC_DRIV_v1` IP block. The block gives a high-level interface that hides the communication with the converters and their internal programming, so that the user concentrates on the application logic. Its main characteristics are:

1. **Clock requirement.** The IP requires the 200 MHz clock. A different frequency compromises the operation and accuracy of the DACs.
2. **Output voltage range.** The analog outputs generate voltages in the 0–5 V range, with an analog ground isolated from both the digital ground and the PWM ground.
3. **Resolution.** 16 bits: each step of the conversion is an increment of 76.294 µV.
4. **Refresh rate.** Each channel is updated every 4.48 µs (223.214 kHz per channel).

Table 13 summarises the driver.

Driver	RCP_DAC_DRIV_v1
Controls	16 analog outputs, 0–5 V, 16 bits, with an analog ground isolated from both the digital ground and the PWM ground.
Update rate	4.48 µs per channel (223.214 kHz per channel).
Limits	Minimum step (1 LSB): 76.294 µV. The analog ground reference is taken from one of the ADC grounds.

Table 13: Digital-to-analog converter driver.

5.5.2 How to use

The physical ports of this driver are the `PORT_DAC_*` signals; leave them as the template delivers them. To drive a channel, connect the 16-bit output of the user's design to the driver input of that channel, `DATA_DAC_<ID>_<N>J<xx>[15:0]`, where `<ID>` is the channel number, `<N>` the pin within the header and `<xx>` the header, as Figure 81 shows.

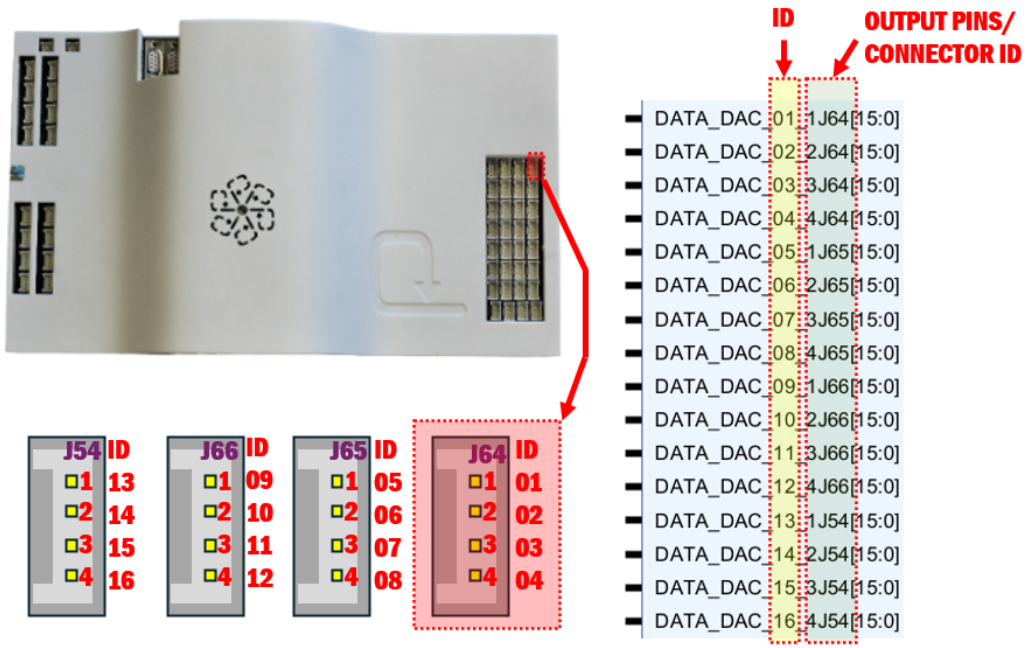


Figure 81: The 16 analog outputs on the headers J64, J65, J66 and J54, and the driver input of each one; the channel number (ID) and the pin and header appear in the port name.

The value the user's design must generate for a given output voltage comes from Equation 2, where $Data$ is the 16-bit value written to the driver input, V_{out} the voltage wanted at the output and V_{ref} the reference of the DAC.

$$Data = (2^{16} - 1) \times \frac{V_{out}}{V_{ref}}, \qquad V_{ref} = 5\,\text{V} \tag{2}$$

Two limitations follow from it:

- The DAC only generates voltages between 0 V and 5 V, so V_{out} must always lie within that range.
- If the data of the design lies outside that range, it must be preprocessed before it reaches the driver. To generate a sinusoidal signal, for instance, the signal must first be made entirely positive, since the DAC cannot handle negative values: add an offset that brings the whole signal into the positive range and, if the output range is to be used in full, apply a gain so that the peak of the sinusoid corresponds to 5 V at the output.

5.5.3 Connections

Table 14 lists the port groups of the block and Table 15 the full name of each channel.

Name	Size	Type	Description	Use
`DATA_DAC_01_1J64[15:0]` … `DATA_DAC_16_4J54[15:0]`	16 bit	Input	Value to program on each DAC output; 16 ports (Table 15).	User
`PORT_DAC_NSYNC_13U61_64`, `PORT_DAC_SCLK_12U61_64`, `PORT_DAC_SDIN_14U61J64`, `PORT_DAC_SDIN_14U62J65`, `PORT_DAC_SDIN_14U63J66`, `PORT_DAC_SDIN_14U64J54`	1 bit	Output	Connection to the physical pins of the SoC.	HW port
`clk`, `RST`	1 bit	Input	200 MHz clock and reset, already connected in the template.	Template

Table 14: Connections of the `RCP_DAC_DRIV_v1` driver.

ID	Data port	Header, pin	ID	Data port	Header, pin
01	`DATA_DAC_01_1J64[15:0]`	J64, 1	09	`DATA_DAC_09_1J66[15:0]`	J66, 1
02	`DATA_DAC_02_2J64[15:0]`	J64, 2	10	`DATA_DAC_10_2J66[15:0]`	J66, 2
03	`DATA_DAC_03_3J64[15:0]`	J64, 3	11	`DATA_DAC_11_3J66[15:0]`	J66, 3
04	`DATA_DAC_04_4J64[15:0]`	J64, 4	12	`DATA_DAC_12_4J66[15:0]`	J66, 4
05	`DATA_DAC_05_1J65[15:0]`	J65, 1	13	`DATA_DAC_13_1J54[15:0]`	J54, 1
06	`DATA_DAC_06_2J65[15:0]`	J65, 2	14	`DATA_DAC_14_2J54[15:0]`	J54, 2
07	`DATA_DAC_07_3J65[15:0]`	J65, 3	15	`DATA_DAC_15_3J54[15:0]`	J54, 3
08	`DATA_DAC_08_4J65[15:0]`	J65, 4	16	`DATA_DAC_16_4J54[15:0]`	J54, 4

Table 15: Data ports of the 16 DAC channels, as named on the block.



5.6 Analog-to-digital converters: SmartRCP_ADc_Driv

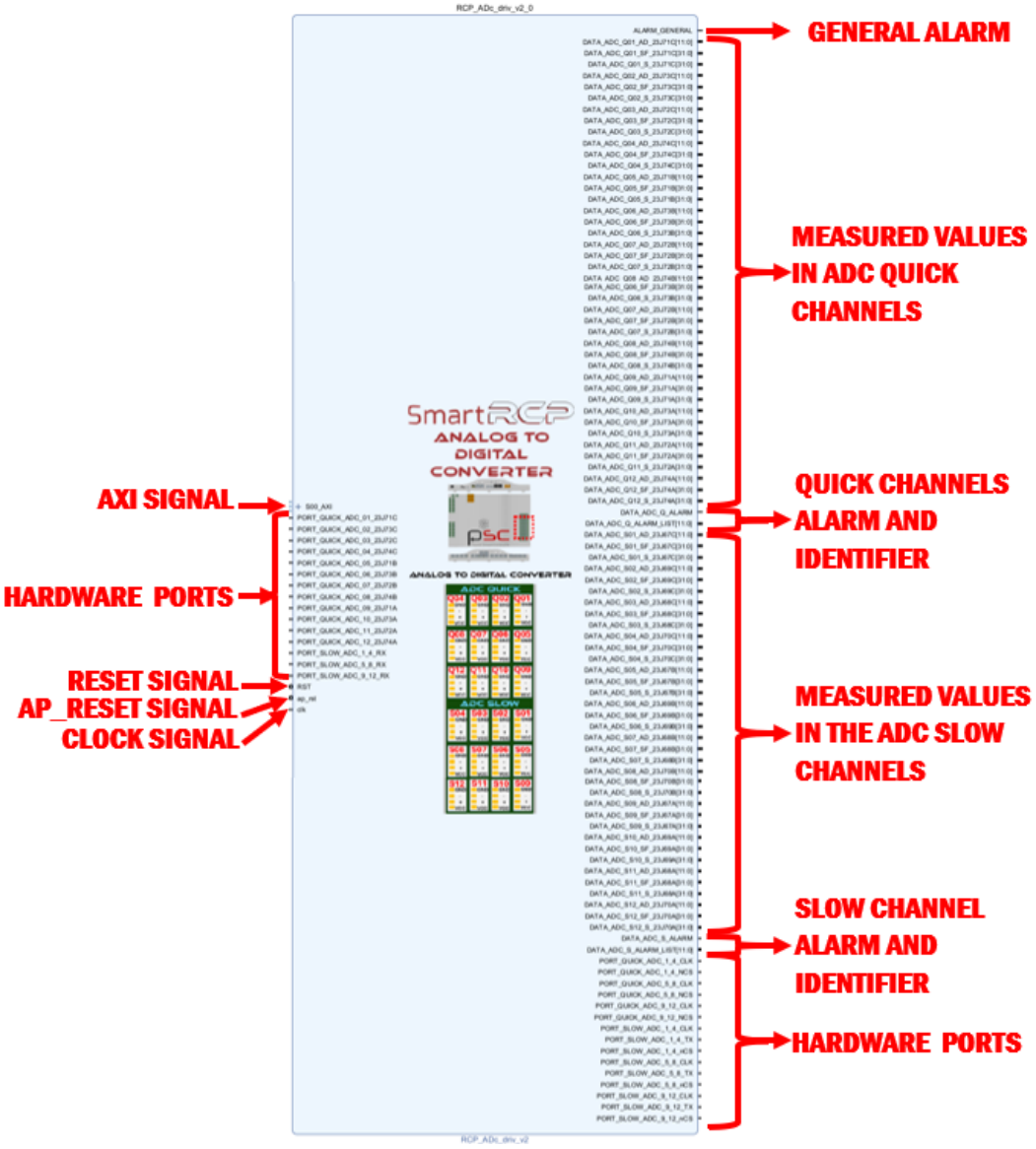


Figure 82: The `SmartRCP_ADc_Driv` analog-to-digital driver: physical ports and control signals on the left; on the right, from top to bottom, the general alarm, the fast channels with their alarm and identifier, the slow channels with theirs, and the remaining physical ports.

5.6.1 Description

The SmartRCP has 24 analog-to-digital converter (ADC) inputs, split by sampling rate into 12 fast channels and 12 slow channels, 0 to 5 V with 12-bit resolution. The `SmartRCP_ADc_Driv` driver gives access to them: it runs on its own from the moment the board powers up and delivers its readings continuously. Used together with the SmartRCP software interface, it adds:

- conversion of the sampled value to floating point;
- offset and gain correction;
- low-pass filtering with adjustable cut-off frequency;
- alarms, with a maximum value, a minimum value and a dwell time that validates the alarm.

If the driver is used in a project that does not run the `APP_RCP_USER_INTERFACE.elf` application, only the raw ADC readings are available and the user is responsible for any further processing. The technical specifications are:

- **Clock requirement.** The driver must be connected to the 200 MHz clock.
- **Memory address.** `0x8000_0000`, already set in the template.
- **Measurement range.** 0 V to 5 V. The analog front end has a gain of 1 V/V on the slow channels and 0.66 V/V on the fast ones.
- **Resolution.** 12 bits, a step of 1.2207 mV per LSB.
- **Sampling rate.** Fast channels: 625 kSPS (1.6 µs between samples). Slow channels: 121.21 kSPS (8.25 µs between samples). Processing adds 0.8 µs (offset and gain) plus 0.95 µs (filter) of latency without changing the refresh rate.
- **Analog bandwidth.** At −3 dB, 200 kHz on the slow channels and 1 MHz on the fast ones — more decisive than the sampling rate when choosing a sampling strategy.

Every channel delivers the same reading in three formats, told apart by a suffix in the port name: `AD` is the raw 12-bit value, `S` the value already scaled to physical units and `SF` that same value filtered. Fast channels carry the letter `Q` before the channel number and slow ones the letter `S`. Figure 83 shows the interface of one channel and where the channels sit on the board.

Table 16 summarises the driver.

5.6.2 How to use

The physical ports of this driver are the `PORT_QUICK_ADC_*` and `PORT_SLOW_ADC_*` signals; leave them as the template delivers them. If the block is placed in a new project, also make sure that its address in the **Address Editor** is `0x8000_0000`.

How the driver is used depends on whether the SmartRCP software interface is part of the design or not. The interface is indispensable for the advanced operations: floating-point conversion, gain and offset correction, low-pass filtering and alarms.



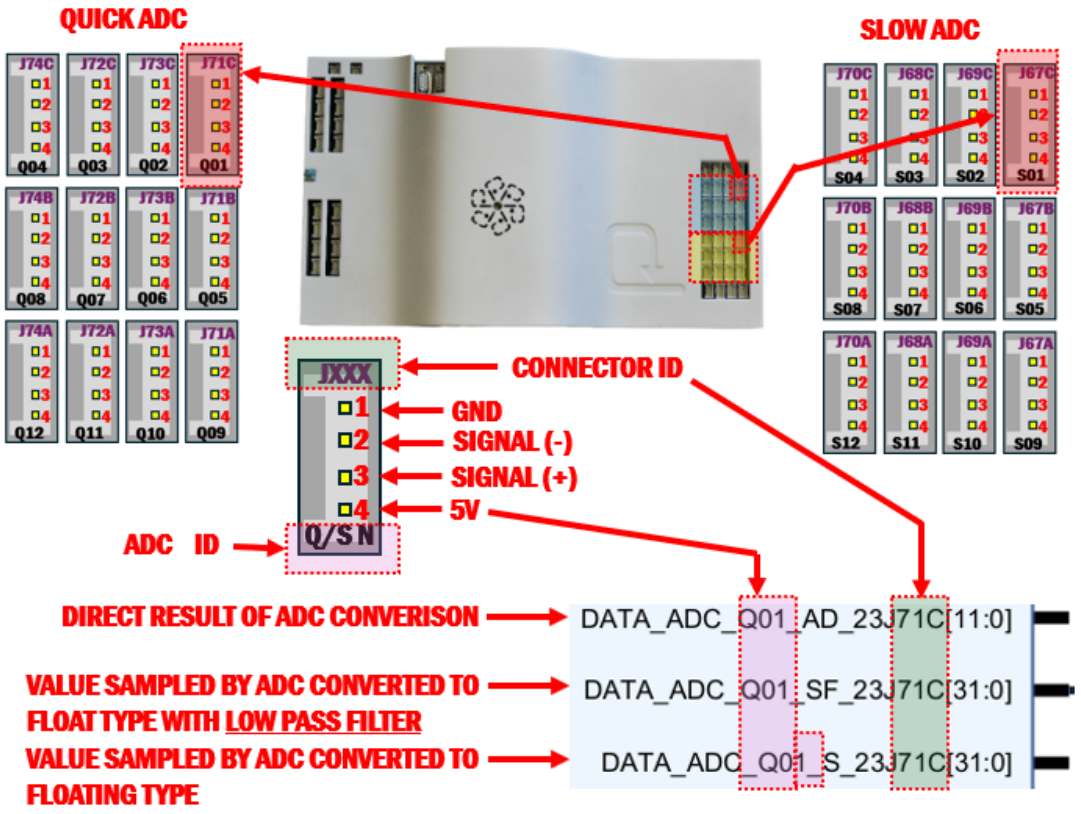


Figure 83: ADC channels on the board (fast on the left, slow on the right), the pin-out of each input header — ground, signal −, signal +, 5 V — and the three outputs every channel offers for the same reading: raw, filtered and scaled.

Driver	SmartRCP_ADc_Driv
Controls	24 analog inputs (12 fast, 12 slow), 0–5 V, 12 bits, with three output formats per channel and alarms per group.
Update rate	Fast channels: 625 kSPS (1.6 µs). Slow channels: 121.21 kSPS (8.25 µs). Processing adds 0.8 µs (offset and gain) plus 0.95 µs (filter) = 1.75 µs of latency, without changing the refresh rate.
Limits	1.2207 mV per LSB. Real analog bandwidth at −3 dB: 200 kHz on the slow channels, 1 MHz on the fast ones. Analog front-end gain: 1 V/V on the slow channels, 0.66 V/V on the fast ones.

Table 16: Analog-to-digital converter driver.

Without the software interface. Only the direct outputs of the driver are usable: the `AD` ports, which deliver valid 12-bit values. Connect them to the user's logic and, where needed, convert them to a voltage with Equation 3, where N_{read} is the 12-bit unsigned value read from the port and V_{read} the voltage present at the ADC input.

$$V_{read} = V_{ref} \times \frac{N_{read}}{2^{12} - 1}, \qquad V_{ref} = 5\,\mathrm{V} \tag{3}$$

The user must also compensate the offset and the gain added by the sensing chain in front of the input, so that the real value of the measured quantity is recovered.

With the software interface. All the advanced functions are available: the reading as a 32-bit floating-point number, and filters applied to it according to the needs of the application, all adjusted from the interface. The user's IP can be connected to the scaled outputs (`S`) or to the filtered ones (`SF`), as the design requires. The scaled value comes from the raw one by applying first the offset and then the gain, both set for that channel from the interface (Equation 4).

$$\mathtt{S} = (\mathtt{AD} + \mathrm{offset}) \times \mathrm{gain} \tag{4}$$

The gain configured from the interface only compensates the external sensing chain: the gain of the ADC conversion itself is already corrected internally, so the user does not have to account for it. Figure 84 shows the functional blocks the driver is made of and which of their parameters the interface controls.

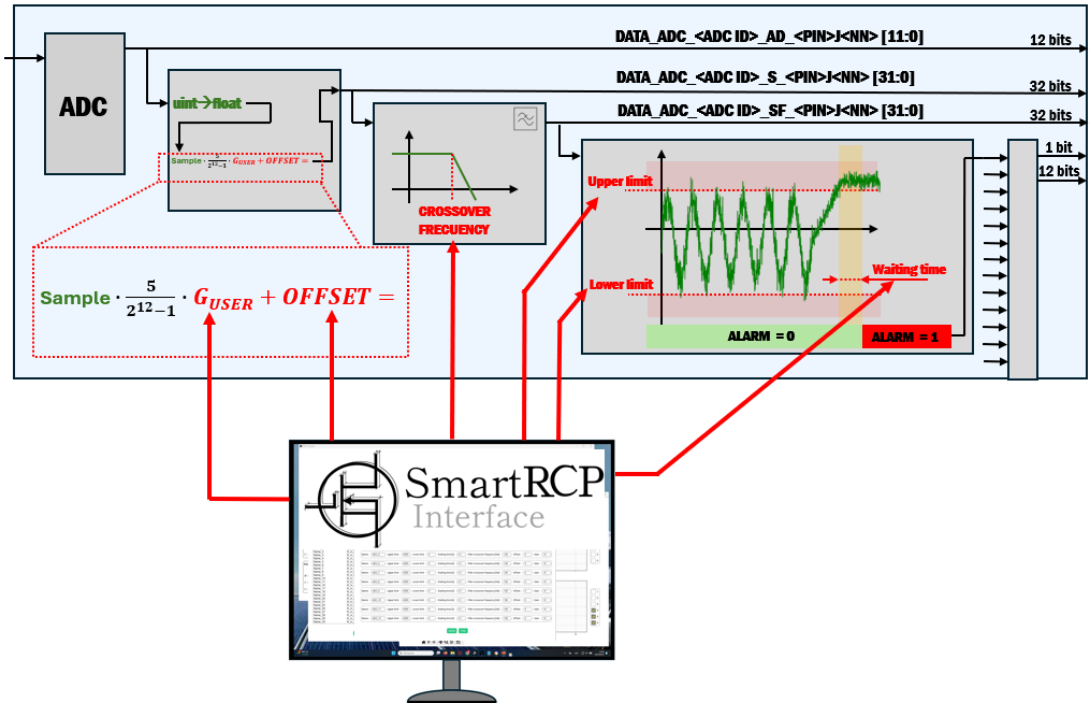


Figure 84: Functional blocks of the ADC driver for one channel — conversion, scaling, low-pass filter and alarm — and the parameters of each one that are set from the SmartRCP Interface.

Alarms are also set from the interface, with a maximum value, a minimum value and a minimum holding time, which allows accurate detection of abnormal conditions: if a

signal exceeds those limits for the defined time an alarm signal is raised, and it latches until it is rearmed from the interface (see the *SmartRCP Interface* manual). The alarm outputs are:

- `ALARM_GENERAL`: `'0'` if there is no alarm, `'1'` if any alarm has occurred.
- `DATA_ADC_Q_ALARM`: `'0'` if there is no alarm on the fast channels, `'1'` if there is.
- `DATA_ADC_S_ALARM`: the same for the slow channels.
- `DATA_ADC_Q_ALARM_LIST[11:0]`: one bit per fast channel, according to its number; a `'1'` marks an alarm on that channel.
- `DATA_ADC_S_ALARM_LIST[11:0]`: the same for the slow channels.

This lets the user identify the source of an alarm quickly and act on it.

5.6.3 Connections

Table 17 lists the port groups of the block and Table 18 the full name of the three data outputs of every channel.

Name	Size	Type	Description	Use
DATA_ADC_<ID>_AD_23J<xx>[11:0]	12 bit	Output	Raw 12-bit sample.	User
DATA_ADC_<ID>_S_23J<xx>[31:0]	32 bit	Output	Scaled value (offset, gain), float.	User
DATA_ADC_<ID>_SF_23J<xx>[31:0]	32 bit	Output	Scaled and filtered value, float.	User
ALARM_GENERAL, DATA_ADC_Q_ALARM, DATA_ADC_S_ALARM	1 bit	Output	Alarms: general, fast, slow.	User
DATA_ADC_Q_ALARM_LIST[11:0], DATA_ADC_S_ALARM_LIST[11:0]	12 bit	Output	Alarm status per channel, fast and slow.	User
PORT_QUICK_ADC_<ID>_23J<xx>, PORT_SLOW_ADC_1_4_RX, PORT_SLOW_ADC_5_8_RX, PORT_SLOW_ADC_9_12_RX	1 bit	Input	Connection to the physical pins of the SoC; 12 fast ports, same header as the channel, and 3 slow ports.	HW port
PORT_QUICK_ADC_<g>_CLK, PORT_QUICK_ADC_<g>_NCS, PORT_SLOW_ADC_<g>_CLK, PORT_SLOW_ADC_<g>_TX, PORT_SLOW_ADC_<g>_nCS	1 bit	Output	Connection to the physical pins of the SoC, for the groups <g> = 1_4, 5_8, 9_12.	HW port
S00_AXI	AXI	Slave	Register access, base address 0x8000_0000.	Template
clk, RST, ap_rst	1 bit	Input	200 MHz clock and resets, connected in the template.	Template

Table 17: Connections of the SmartRCP_ADc_Driv driver.

Ch.	Raw (AD)	Scaled (S)	Filtered (SF)
Q01	DATA_ADC_Q01_AD_23J71C[11:0]	DATA_ADC_Q01_S_23J71C[31:0]	DATA_ADC_Q01_SF_23J71C[31:0]
Q02	DATA_ADC_Q02_AD_23J73C[11:0]	DATA_ADC_Q02_S_23J73C[31:0]	DATA_ADC_Q02_SF_23J73C[31:0]
Q03	DATA_ADC_Q03_AD_23J72C[11:0]	DATA_ADC_Q03_S_23J72C[31:0]	DATA_ADC_Q03_SF_23J72C[31:0]
Q04	DATA_ADC_Q04_AD_23J74C[11:0]	DATA_ADC_Q04_S_23J74C[31:0]	DATA_ADC_Q04_SF_23J74C[31:0]
Q05	DATA_ADC_Q05_AD_23J71B[11:0]	DATA_ADC_Q05_S_23J71B[31:0]	DATA_ADC_Q05_SF_23J71B[31:0]
Q06	DATA_ADC_Q06_AD_23J73B[11:0]	DATA_ADC_Q06_S_23J73B[31:0]	DATA_ADC_Q06_SF_23J73B[31:0]
Q07	DATA_ADC_Q07_AD_23J72B[11:0]	DATA_ADC_Q07_S_23J72B[31:0]	DATA_ADC_Q07_SF_23J72B[31:0]
Q08	DATA_ADC_Q08_AD_23J74B[11:0]	DATA_ADC_Q08_S_23J74B[31:0]	DATA_ADC_Q08_SF_23J74B[31:0]
Q09	DATA_ADC_Q09_AD_23J71A[11:0]	DATA_ADC_Q09_S_23J71A[31:0]	DATA_ADC_Q09_SF_23J71A[31:0]
Q10	DATA_ADC_Q10_AD_23J73A[11:0]	DATA_ADC_Q10_S_23J73A[31:0]	DATA_ADC_Q10_SF_23J73A[31:0]
Q11	DATA_ADC_Q11_AD_23J72A[11:0]	DATA_ADC_Q11_S_23J72A[31:0]	DATA_ADC_Q11_SF_23J72A[31:0]
Q12	DATA_ADC_Q12_AD_23J74A[11:0]	DATA_ADC_Q12_S_23J74A[31:0]	DATA_ADC_Q12_SF_23J74A[31:0]
S01	DATA_ADC_S01_AD_23J67C[11:0]	DATA_ADC_S01_S_23J67C[31:0]	DATA_ADC_S01_SF_23J67C[31:0]
S02	DATA_ADC_S02_AD_23J69C[11:0]	DATA_ADC_S02_S_23J69C[31:0]	DATA_ADC_S02_SF_23J69C[31:0]
S03	DATA_ADC_S03_AD_23J68C[11:0]	DATA_ADC_S03_S_23J68C[31:0]	DATA_ADC_S03_SF_23J68C[31:0]
S04	DATA_ADC_S04_AD_23J70C[11:0]	DATA_ADC_S04_S_23J70C[31:0]	DATA_ADC_S04_SF_23J70C[31:0]
S05	DATA_ADC_S05_AD_23J67B[11:0]	DATA_ADC_S05_S_23J67B[31:0]	DATA_ADC_S05_SF_23J67B[31:0]
S06	DATA_ADC_S06_AD_23J69B[11:0]	DATA_ADC_S06_S_23J69B[31:0]	DATA_ADC_S06_SF_23J69B[31:0]
S07	DATA_ADC_S07_AD_23J68B[11:0]	DATA_ADC_S07_S_23J68B[31:0]	DATA_ADC_S07_SF_23J68B[31:0]
S08	DATA_ADC_S08_AD_23J70B[11:0]	DATA_ADC_S08_S_23J70B[31:0]	DATA_ADC_S08_SF_23J70B[31:0]
S09	DATA_ADC_S09_AD_23J67A[11:0]	DATA_ADC_S09_S_23J67A[31:0]	DATA_ADC_S09_SF_23J67A[31:0]
S10	DATA_ADC_S10_AD_23J69A[11:0]	DATA_ADC_S10_S_23J69A[31:0]	DATA_ADC_S10_SF_23J69A[31:0]
S11	DATA_ADC_S11_AD_23J68A[11:0]	DATA_ADC_S11_S_23J68A[31:0]	DATA_ADC_S11_SF_23J68A[31:0]
S12	DATA_ADC_S12_AD_23J70A[11:0]	DATA_ADC_S12_S_23J70A[31:0]	DATA_ADC_S12_SF_23J70A[31:0]

Table 18: Data outputs of the 24 ADC channels, as named on the block; the header of each channel is the last field of the name.

5.7 Write registers: OUTPUT_USER_REGISTERS

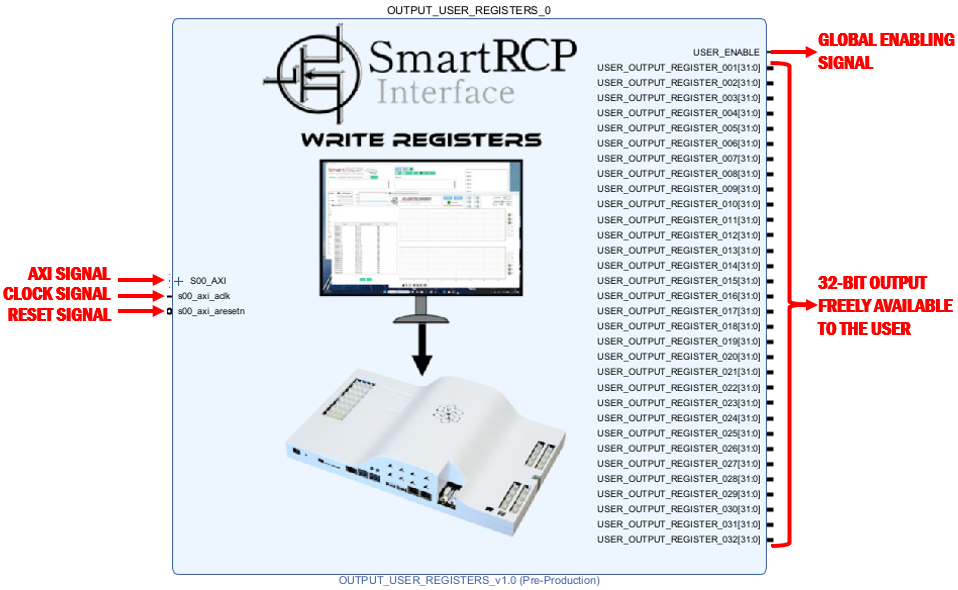


Figure 85: The `OUTPUT_USER_REGISTERS` write register block. The arrow gives the direction of travel, from the computer into the SmartRCP: this is the block the remote application presents as *Input-Register*.

5.7.1 Description

The `OUTPUT_USER_REGISTERS` IP belongs to the group of IPs specific to the SmartRCP software interface. It lets the user write values from the console, in floating-point or integer format, and makes them available inside the FPGA for the user's own designs: 32 outputs of 32 bits. It also has one 1-bit output, `USER_ENABLE`, reserved for an enable signal controlled from the interface: `'1'` while enabled, `'0'` otherwise. At power-up and after a reset the signal starts disabled (`'0'`).

The **Address Editor** must assign it the master base address `0x8002_0000`. This is already set in the template; if the IP is added to a new project, the setting must be checked. Table 19 summarises the block.

IP block	OUTPUT_USER_REGISTERS
Purpose	Carries up to 32 values of 32 bits from the remote application into your design, integer or floating point, plus one enable signal.
Update rate	On demand from the remote application.
Limits	The application clamps values to $\pm 49{,}998.0$ and silently truncates anything below 1×10^{-12} in absolute value. An unconfigured register reads 0; if you configure one and then delete it, the last value set keeps being sent.

Table 19: Write register block (from the computer to the programmable logic).

5.7.2 How to use

If the IP is added to a project other than the template, first confirm that its address in the **Address Editor** is the one given above; otherwise the design is not compatible with `APP_RCP_USER_INTERFACE.elf` and the software interface cannot be used. The check is only needed when a project is created from scratch.

The function of the block is entirely up to the user: its outputs are wired to whichever block of the design needs them, to carry control variables, status, operating modes, an enable signal and so on. The type of data sent from the interface has to match the type that block expects.

If a 32-bit output is connected to a narrower input, only the least significant bits are used. To drive a 5-bit unsigned input from the interface, for example, the connection is:



```
IP_USER_EXAMPLE[0] <= USER_OUTPUT_REGISTER_XXX[0]
IP_USER_EXAMPLE[1] <= USER_OUTPUT_REGISTER_XXX[1]
IP_USER_EXAMPLE[2] <= USER_OUTPUT_REGISTER_XXX[2]
IP_USER_EXAMPLE[3] <= USER_OUTPUT_REGISTER_XXX[3]
IP_USER_EXAMPLE[4] <= USER_OUTPUT_REGISTER_XXX[4]
NOT CONNECTED      <= USER_OUTPUT_REGISTER_XXX[5]
...
NOT CONNECTED      <= USER_OUTPUT_REGISTER_XXX[31]
```

5.7.3 Connections

Table 20 lists the ports of the block.

Name	Size	Type	Description	Use
USER_OUTPUT_REGISTER_001[31:0] … USER_OUTPUT_REGISTER_032[31:0]	32 bit	Output	Value written from the interface, integer or floating point; 32 ports.	User
USER_ENABLE	1 bit	Output	Global enable signal controlled from the interface.	User
S00_AXI	AXI	Slave	Register access from the processor, base address 0x8002_0000.	Template
s00_axi_aclk, s00_axi_aresetn	1 bit	Input	200 MHz clock and active-low reset, already connected in the template.	Template

Table 20: Connections of the OUTPUT_USER_REGISTERS block.

5.8 Read registers: INPUT_USER_REGISTERS

5.8.1 Description

The INPUT_USER_REGISTERS IP also belongs to the group of IPs specific to the SmartRCP software interface. It lets the user read, from the interface, internal values of the FPGA, so that their current state and their evolution over time can be followed. The values must always be 32-bit floating-point numbers. The block has 32 inputs of 32 bits, which the user connects to the custom blocks of the design.

The **Address Editor** must assign it the master base address 0x8001_0000. This is already set in the template; if the IP is added to a new project, the setting must be verified. Table 21 summarises the block.



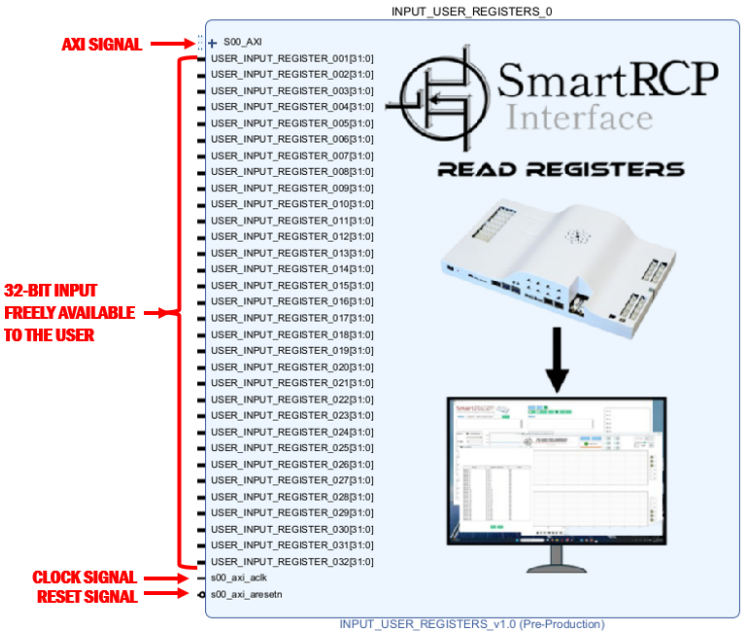


Figure 86: The `INPUT_USER_REGISTERS` read register block, with the arrow pointing the opposite way to the previous one: this is the block the remote application presents as *RCP Output-Registers*.

IP block	INPUT_USER_REGISTERS
Purpose	Carries up to 32 floating-point values from your design out to the remote application.
Update rate	Continuous, tied to how fast the remote application reads.
Limits	32-bit floating point only. Tie unused inputs to a constant.

Table 21: Read register block (from the programmable logic to the computer).

5.8.2 How to use

If the IP is added to a project other than the template, first confirm that its address in the **Address Editor** is the one given above; otherwise the design is not compatible with `APP_RCP_USER_INTERFACE.elf` and the software interface cannot be used. The check is only needed when a project is created from scratch.

Connect the inputs of the block to the data outputs of the IP blocks that are to be observed from the interface, remembering that the interface only supports 32-bit floating-point data. To observe values of another numeric type — the state of a state machine, an integer — one of two approaches is recommended:

1. **State machines and other small sets of values.** Write a block in VHDL or Verilog that assigns a predefined floating-point number to each state. The binary pattern of any floating-point number can be obtained with a tool such as `https://gregstoll.com/~gregstoll/floattohex/`.

2. **Integers with no fixed range.** For counters, large quantities or many values, use a High-Level Synthesis (HLS) block that casts the integer to a 32-bit floating-point number, the format the interface supports. HLS is written in C or C++, where the conversion is one line: `float output_float = (float) input_int;`

5.8.3 Connections

Table 22 lists the ports of the block.

Name	Size	Type	Description	Use
`USER_INPUT_REGISTER_001[31:0]` … `USER_INPUT_REGISTER_032[31:0]`	32 bit	Input	Floating-point value shown by the interface; 32 ports. Tie unused ones to a constant.	User
`S00_AXI`	AXI	Slave	Register access from the processor, base address `0x8001_0000`.	Template
`s00_axi_aclk`, `s00_axi_aresetn`	1 bit	Input	200 MHz clock and active-low reset, already connected in the template.	Template

Table 22: Connections of the `INPUT_USER_REGISTERS` block.

5.9 PL-DataLogger: RCP_DataLogger

5.9.1 Description

The `RCP_DataLogger` IP is a component developed by Power Smart Control for the SmartRCP, designed to work together with the SmartRCP software interface. The pair behaves much like a benchtop oscilloscope for the waveforms of signals inside the FPGA, whatever their origin — custom IP blocks or the analog-to-digital converters. The DataLogger cannot be used without the software interface. Its main features are:

- **Internal signal capture.** Up to 32 internal signals of the FPGA can be connected as probes and visualised.
- **Trigger.** A trigger level synchronises the capture. Only channel 1 can be the trigger source.
- **Sampling.** The minimum capture step is 40 ns; any longer step can be set.
- **Channels.** Six channels, the maximum number of signals that can be captured and displayed at the same time. Any of the 32 probes can be selected for any channel.



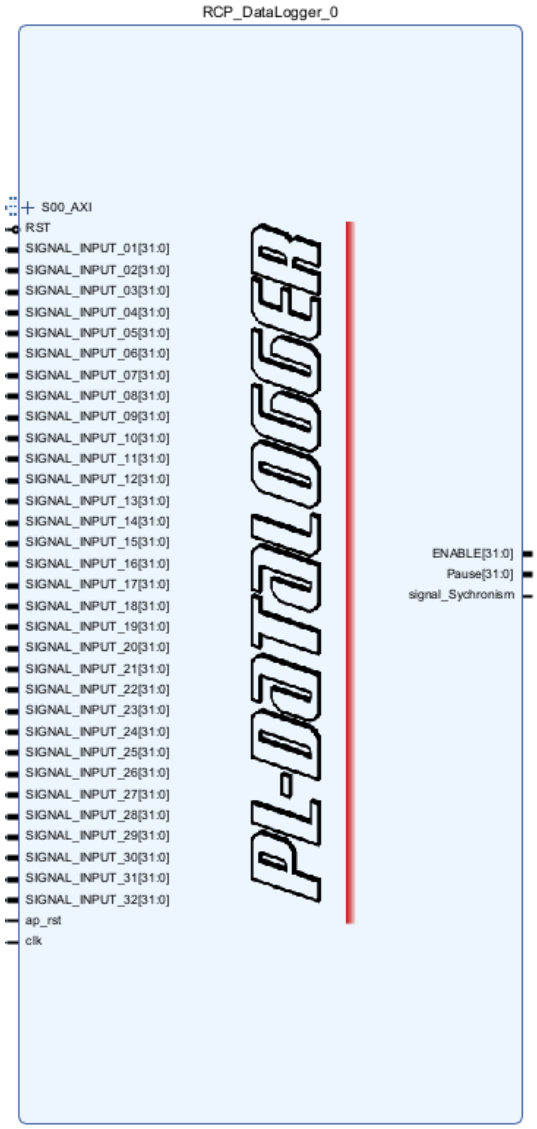


Figure 87: The `RCP_DataLogger` block. On the left, from top to bottom: the `S00_AXI` interface, the `RST` reset, the 32 floating-point probes `SIGNAL_INPUT_01` to `SIGNAL_INPUT_32`, and `ap_rst` and `clk`; on the right, the debugging outputs `ENABLE`, `Pause` and `signal_Sychronism`, whose name is reproduced exactly as it appears in the template.

- **Floating-point data.** The signals connected to the 32 probes must be 32-bit floating-point values.

The DataLogger has two operating modes:

- **Single capture.** One capture is performed when requested from the software interface. After the request, the DataLogger waits until the signal on channel 1 crosses the trigger level or the maximum waiting time of 25 ms elapses; then 15,000 samples are captured per channel, spaced by the time step configured in the interface, the data is sent to the interface and the DataLogger stops.



- **Continuous capture.** The DataLogger waits for the trigger condition on channel 1, or for the 25 ms time-out, captures 2,500 samples per channel spaced by the configured time step, sends them to the interface and repeats, updating the waveforms in real time until a stop command is sent from the interface.

The capture window is the time step times the number of samples: at the 40 ns minimum step, continuous mode captures 0.1 ms of data. Every setting — time step, signal selection per channel, trigger — is made from the software interface; the *SmartRCP Interface* manual explains how.

The IP must sit at address `0x8000_1000` in the **Address Editor** (Figure 88). This is preconfigured in the template; if the IP is added to a new project the address must be verified, or the IP will not work and no signal will reach the interface. It must also be clocked at 200 MHz: with any other clock the sampling step and the capture times no longer meet their specification, which directly affects the accuracy of the captures and their integration with the interface.

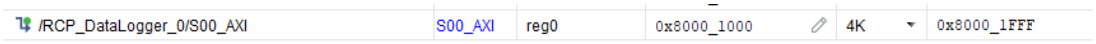


Figure 88: The `RCP_DataLogger` entry in Vivado's **Address Editor**, at `0x8000_1000` with a 4 K range.

The interface of the block (Figure 87) has three functional parts:

- **Reset, `ap_rst`, clock and AXI signals.** Standard signals of any IP block with an AXI interface. In a new project they are connected with **Run Connection Automation**; the only additional considerations are the address mapping described above and, if several clocks are available, that the clock is the 200 MHz one. `ap_rst` must be connected to the reset block, to the output of the same polarity (active high, the one used by Vitis HLS blocks; see section 5.1).
- **32 floating-point probes.** The 32-bit inputs `SIGNAL_INPUT_01` to `SIGNAL_INPUT_32` that feed the six channels. The user connects them as the design requires; only floating-point signals are supported.
- **Internal debugging signals.** The outputs `ENABLE`, `Pause` and `signal_Sychronism`, used to debug the operation of the DataLogger itself; they are left unconnected.

Table 23 summarises the block.

5.9.2 How to use

The DataLogger is designed to be driven from the software interface, so the user only has to:

IP block	RCP_DataLogger
Purpose	Captures waveforms inside the FPGA on 6 of its 32 inputs, chosen from the remote application.
Update rate	Minimum capture step 40 ns, configurable to any longer value.
Limits	Window: 2,500 samples per channel in continuous mode, 15,000 in single-shot mode. Forced trigger at 25 ms if the condition is not met. The trigger channel is always channel 1. Tie unused inputs to a constant.

Table 23: Built-in DataLogger block.

1. **Select and connect the signals.** Connect the signals to capture to the inputs `SIGNAL_INPUT_<NN>[31:0]`, where `<NN>` is the probe number, from 01 to 32. Only 32-bit floating-point signals are supported; a signal of another type — an integer, a state — must be converted first with an intermediate block, exactly as described for the read registers in section 5.8: a VHDL or Verilog block that assigns a floating-point value to each state, or an HLS block that casts the integer (`float output_float = (float) input_int;`).
2. **Tie the unused inputs.** Connect every probe that is not used to a 32-bit constant, so that synthesis warnings do not mask other messages.
3. **Run the software part.** The remote application `APP_RCP_USER_INTERFACE.elf` (delivered as `APP_RCP_USER_INTERFACE_A53_0.elf`) must run on the reserved core, `psu_cortexa53_0`, as described in the *Getting started* guide. It is what enables the software interface and, through it, the DataLogger: signal selection, time step and trigger are all set there.

With these steps done and the remote application running on `psu_cortexa53_0`, the DataLogger is operational. Its detailed configuration is described in the *SmartRCP Interface* manual.

5.9.3 Connections

Table 24 lists the ports of the block.

Name	Size	Type	Description	Use
`SIGNAL_INPUT_01[31:0]` ... `SIGNAL_INPUT_32[31:0]`	32 bit	Input	Floating-point probes; 32 ports. Tie unused ones to a constant.	User
`ENABLE[31:0]`, `Pause[31:0]`	32 bit	Output	Internal debugging signals; left unconnected.	—
`signal_Sychronism`	1 bit	Output	Internal debugging signal; left unconnected.	—
`S00_AXI`	AXI	Slave	Register access from the processor, base address `0x8000_1000`.	Template
`clk`, `RST`, `ap_rst`	1 bit	Input	200 MHz clock and the two resets, already connected in the template.	Template

Table 24: Connections of the `RCP_DataLogger` block.

5.10 RMS and average calculation: IP_BLOCK_RMS_AVG_v1

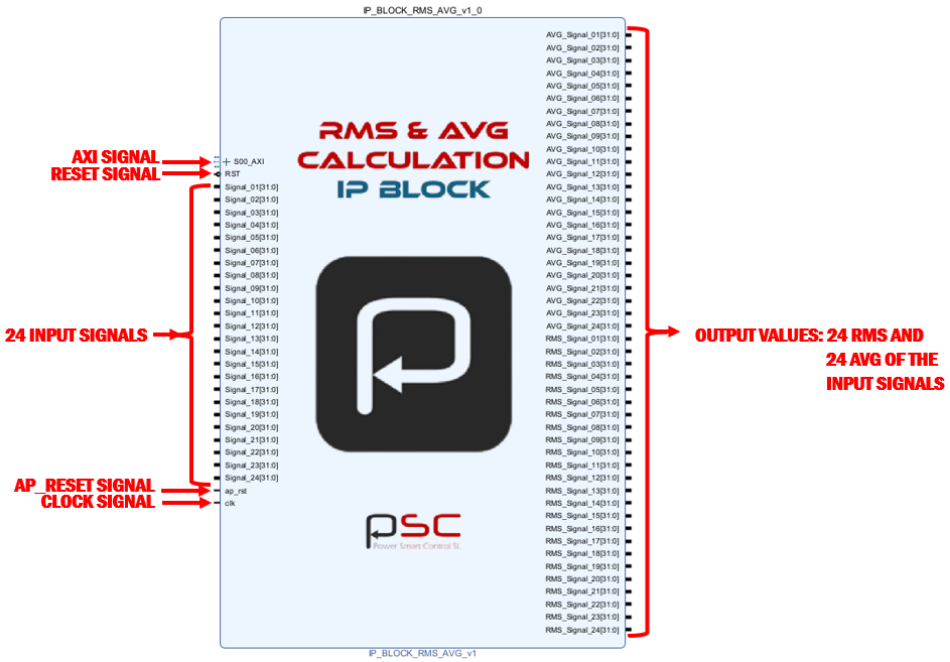


Figure 89: The `IP_BLOCK_RMS_AVG_v1` RMS and average block: 24 floating-point inputs on the left, 24 average and 24 RMS outputs on the right.

5.10.1 Description

The `IP_BLOCK_RMS_AVG_v1` block computes the RMS value and the mean value of the signals captured by the ADCs, and its AXI interface lets the SmartRCP software interface read those values directly. Its key features are:

- **Inputs.** 24 inputs, `Signal_01` to `Signal_24`, which in the template come wired to the filtered outputs of the ADC driver (section 5.6).
- **Outputs.** For each input the block produces two outputs carrying the same number, `RMS_Signal_` and `AVG_Signal_`: 48 outputs in all, 32-bit floating point.
- **AXI interface.** Lets the software interface read the RMS and average values.

In the template all the connections are already made and the address of the block is preset to `0x8000_2000` in the **Address Editor**; if the block is added to a new project, the address must be verified. The block must be connected to the 200 MHz clock. Table 25 summarises it.

IP block	IP_BLOCK_RMS_AVG_v1
Purpose	Computes the RMS value and the mean value of 24 signals, and delivers each result separately.
Update rate	Follows whatever input signal is connected.
Limits	If you change an input, the remote interface keeps showing the name of the original ADC channel even though the value now belongs to the new signal.

Table 25: RMS and average calculation block.

5.10.2 How to use

Although the connections of the block are preconfigured to the filtered outputs of the ADCs, the user is free to modify or duplicate them.

Example 1: reusing an input. To know the RMS and mean value of a signal generated inside the design, take one of the inputs wired to an ADC channel that is not going to be used: disconnect it from the ADC and connect the user's signal instead. The RMS and average values then appear in the software interface in the place the disconnected ADC channel occupied, still under that channel's name.

Example 2: duplicating the block. Alternatively, instantiate a second `IP_BLOCK_RMS_AVG_v1` for the new signals. If a user signal is connected to `Signal_01`, its results are available at `AVG_Signal_01` and `RMS_Signal_01`. These 32-bit floating-point outputs can be connected directly to the inputs of another block that records them, such as the read registers of section 5.8. In this case the memory address of the new block is irrelevant, but it must still be connected to the 200 MHz clock.



5.10.3 Connections

Table 26 lists the ports of the block.

Name	Size	Type	Description	Use
`Signal_01[31:0]` ... `Signal_24[31:0]`	32 bit	Input	Floating-point signals to process; 24 ports, wired to the filtered ADC outputs in the template.	User
`AVG_Signal_01[31:0]` ... `AVG_Signal_24[31:0]`	32 bit	Output	Mean value of each input.	User
`RMS_Signal_01[31:0]` ... `RMS_Signal_24[31:0]`	32 bit	Output	RMS value of each input.	User
`S00_AXI`	AXI	Slave	Register access from the processor, base address `0x8000_2000`.	Template
`clk`, `RST`, `ap_rst`	1 bit	Input	200 MHz clock and the two resets, already connected in the template.	Template

Table 26: Connections of the `IP_BLOCK_RMS_AVG_v1` block.

5.11 General-purpose interrupt timers: RCP_8_iNTERRUT_TIMERS

The names `RCP_8_iNTERRUT_TIMERS` and `AXI_Interrupt_TIMPER_1` to `AXI_Interrupt_TIMPER_8` are reproduced exactly as the IP declares them, unusual spelling included: they are what you will see in Vivado and what the software library refers to.

5.11.1 Description

The `RCP_8_iNTERRUT_TIMERS` block gives the SmartRCP a set of hardware timers that generate periodic or single-shot interrupts towards the processing system. It integrates eight fully independent timers, each with its own AXI interface and its own interrupt line, so that up to eight different functions can run at precise time intervals, asynchronously and with priority over the code being executed at that moment. Its key features are:

- **Timers.** Eight independent 32-bit timers. Each has its own period, its own operating mode and its own interrupt line, so the behaviour of one never interferes with the others.



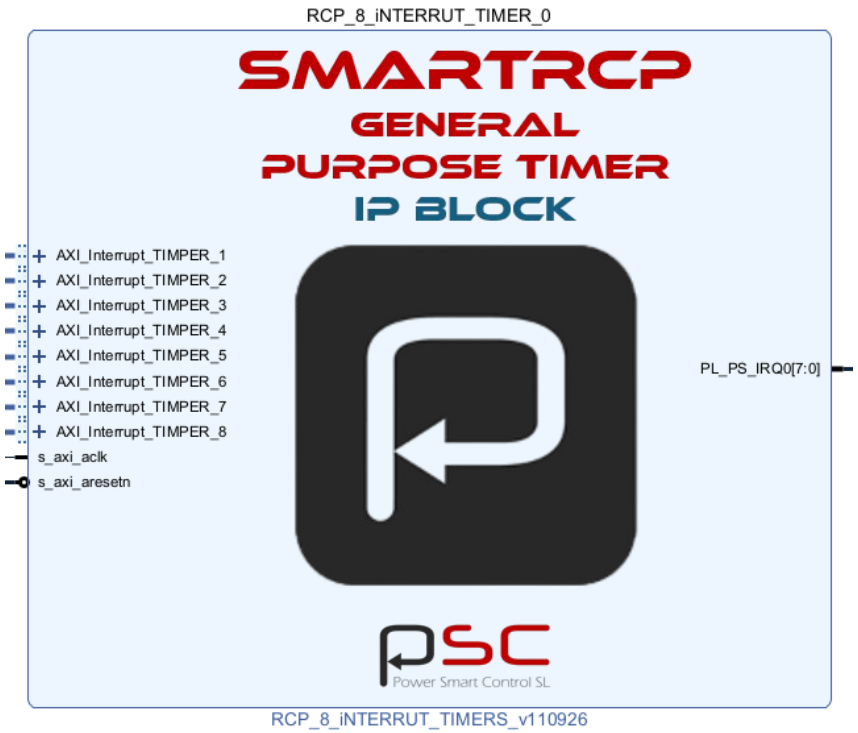


Figure 90: The `RCP_8_iNTERRUT_TIMERS` block: eight AXI interfaces, one per timer, clock and reset on the left, and the 8-bit interrupt bus `PL_PS_IRQ0[7:0]` on the right.

- **AXI interfaces.** Eight independent AXI interfaces, `AXI_Interrupt_TIMPER_1` to `AXI_Interrupt_TIMPER_8`, one per timer. Through them the software sets the period, starts and stops each timer and acknowledges its interrupts.
- **Interrupt output.** All the interrupt lines are grouped in one 8-bit bus, `PL_PS_IRQ0[7:0]`, connected directly to the `IRQ0` input of the processing system. Each bit is an independent interrupt that the processor identifies without any software polling.
- **Resolution and range.** At 200 MHz each timer has a resolution of 5 ns and a maximum period of approximately 21.47 s.
- **Automatic reload.** In periodic mode the counter is reloaded by hardware in the same clock cycle in which it expires, so the period never drifts, however long the processor takes to service the interrupt.

In the template all the connections are already made: clock, reset, the eight AXI interfaces and the `PL_PS_IRQ0[7:0]` bus towards the processing system. The addresses of the eight timers are preset in the **Address Editor**, with a 64 K range each, as Table 27 shows. The interrupt ID is the value by which the processor identifies each timer, and the one the software library uses to link a timer to a user function. If the block is added to a new project, the addresses must be verified and the `PL_PS_IRQ0[7:0]` bus connected to the `IRQ0` input of the processing system, previously enabled in its PS-PL configuration.

AXI interface	Memory address	Interrupt bit	Interrupt ID
AXI_Interrupt_TIMPER_1	0x8008_0000	PL_PS_IRQ0[0]	121
AXI_Interrupt_TIMPER_2	0x8009_0000	PL_PS_IRQ0[1]	122
AXI_Interrupt_TIMPER_3	0x800A_0000	PL_PS_IRQ0[2]	123
AXI_Interrupt_TIMPER_4	0x800B_0000	PL_PS_IRQ0[3]	124
AXI_Interrupt_TIMPER_5	0x800C_0000	PL_PS_IRQ0[4]	125
AXI_Interrupt_TIMPER_6	0x800D_0000	PL_PS_IRQ0[5]	126
AXI_Interrupt_TIMPER_7	0x800E_0000	PL_PS_IRQ0[6]	127
AXI_Interrupt_TIMPER_8	0x800F_0000	PL_PS_IRQ0[7]	128

Table 27: Address, interrupt bit and interrupt ID of each of the eight timers, as preset in the template.

The block needs no configuration at hardware level: the number of timers is fixed, and the period and operating mode of each one are set entirely by software at run time. It must be connected to the 200 MHz clock: the periods computed by the software library are referred to that frequency, so any deviation translates directly into a proportional error in the resulting periods. Table 28 summarises the block.

IP block	RCP_8_iNTERRUT_TIMERS
Purpose	Eight independent hardware timers that interrupt the processing system, each linked by software to a user function.
Update rate	5 ns resolution, maximum period about 21.47 s per timer, hardware reload in the cycle the counter expires.
Limits	Priority sets the order of attention when several timers expire at once, but a function already in execution is never interrupted by another timer: keep the linked functions short. Periods assume the 200 MHz clock.

Table 28: General-purpose interrupt timers block.

5.11.2 How to use

The block is handled from the software through the library supplied with the SmartRCP, which presents each timer as a channel to which the user links a function of their own. Once the channel is started, the function runs automatically every time the timer expires, interrupting the code in progress and returning to it afterwards.



Each channel admits two operating modes. In **periodic** mode the function runs indefinitely at the configured interval. In **single-shot** mode the function runs once and the channel stops, ready to be armed again whenever the user decides.

Each channel also admits its own priority level. When two or more timers expire simultaneously, the processor attends first the one with the highest priority. Priority sets the order of attention only: a function already in execution is never interrupted by another timer, which is why the linked functions should be as short as possible.

Example 1: periodic sampling. To read a value at a fixed interval, link the reading function to a channel in periodic mode with the desired period. From then on the function runs at exactly that interval, without the main program measuring time.

Example 2: delayed action. To run an action once, after a given time, link the function to a channel in single-shot mode. The function runs when the time elapses and the channel stops by itself; the main program can arm it again at any moment, which makes this mode useful for time-outs and delayed responses.

Example 3: combined use. The eight channels can be used simultaneously with different periods and priorities: the fastest channels with the highest priority for critical tasks, the slowest ones for supervision or logging. Channels that are not used remain stopped and consume no processing time.

5.11.3 Connections

Table 29 lists the ports of the block.

Name	Size	Type	Description	Use
`AXI_Interrupt_TIMPER_1` … `AXI_Interrupt_TIMPER_8`	AXI	Slave	Register access to each timer, base addresses of Table 27.	Template
`PL_PS_IRQ0[7:0]`	8 bit	Output	One interrupt line per timer, to the `IRQ0` input of the processing system.	Template
`s_axi_aclk`, `s_axi_aresetn`	1 bit	Input	200 MHz clock and active-low reset, already connected in the template.	Template

Table 29: Connections of the `RCP_8_iNTERRUT_TIMERS` block.



5.12 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 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 chapter 9.

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.

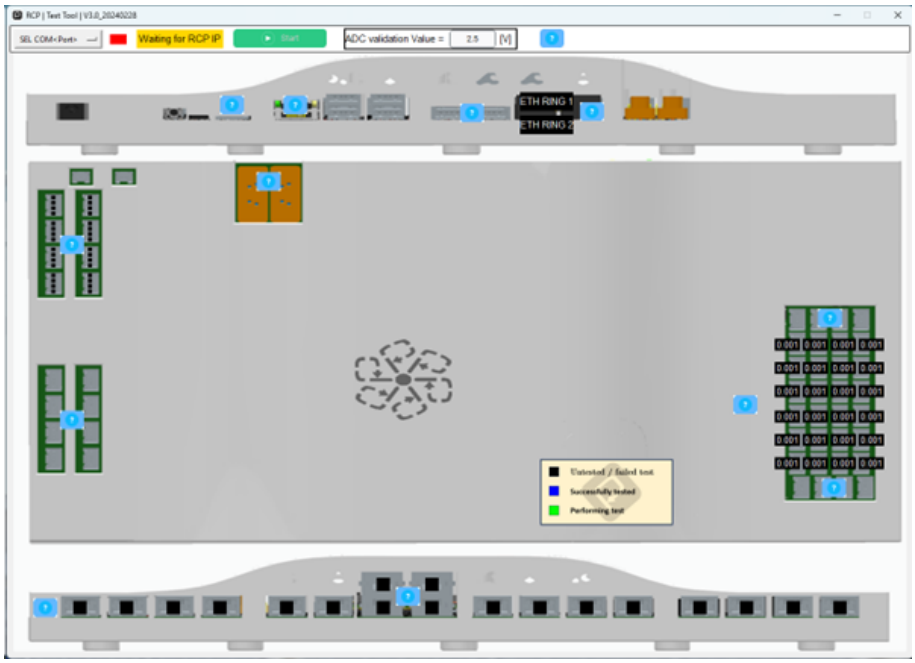


Figure 91: `RCP_Test_Tool` Windows application main window.

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

7 Synchronous Buck Converter demonstrator

This demonstrator is a practical guide for the SmartRCP user and a solid foundation for the development of control and processing projects in power systems. The following sections describe the hardware, a low-power synchronous step-down converter board that runs from a 5 V supply and can be powered by an ordinary 2 A (or higher) mobile-phone charger, and then the recommended workflow to adapt the RCP_TEMPLATE to the particular needs of each project.

The guide covers the whole process: the development of the user IPs in Vitis HLS, the customization of the Vivado project, the software configuration in Vitis IDE, the programming of the device and, finally, the configuration and control of the converter from the SmartRCP software interface. Although it is illustrated with a step-down converter, the same workflow applies to any other power-control project the user may wish to implement. The chapter closes with the operational outcome of the project, a guided functional test and the analysis of its results.

The demonstrator assumes that the user makes full use of the SmartRCP functionalities, including the desktop software interface application, which is used to control the converter.

> **Software versions used in this example**
>
> The screenshots and register names of this demonstrator were taken with the SmartRCP Interface version **2024.2**. The chapter will be updated to the current Interface in a future revision of this manual. The procedure is unchanged: entering the same register values in the current console produces the behaviour described here.

7.1 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:

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

- Two power MOSFETs with diode and snubber
- Output LC filter



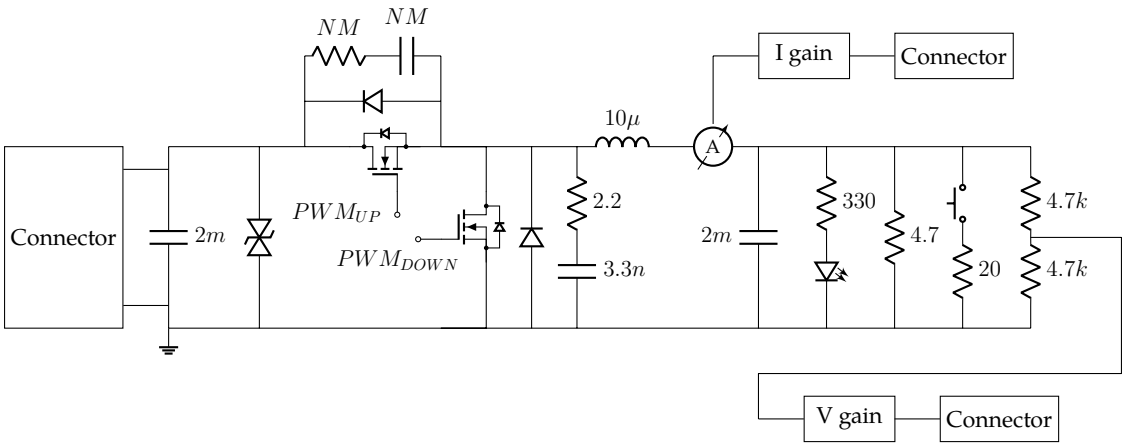


Figure 92: Buck Converter topology included in RCPBuckBoard

- 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 4.7Ohm.

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: 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. 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.

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.

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

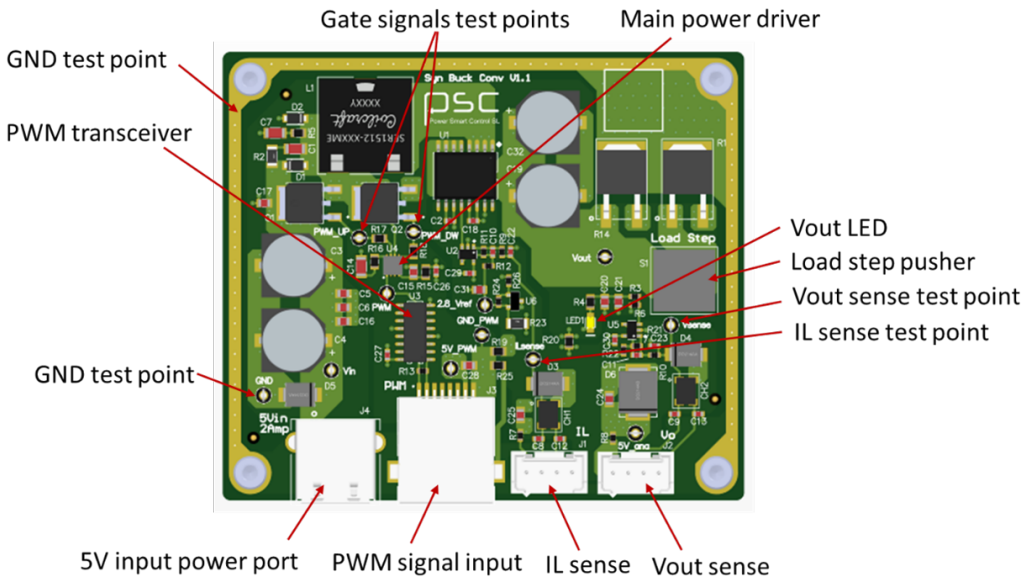


Figure 93: RCPBuckBoard.

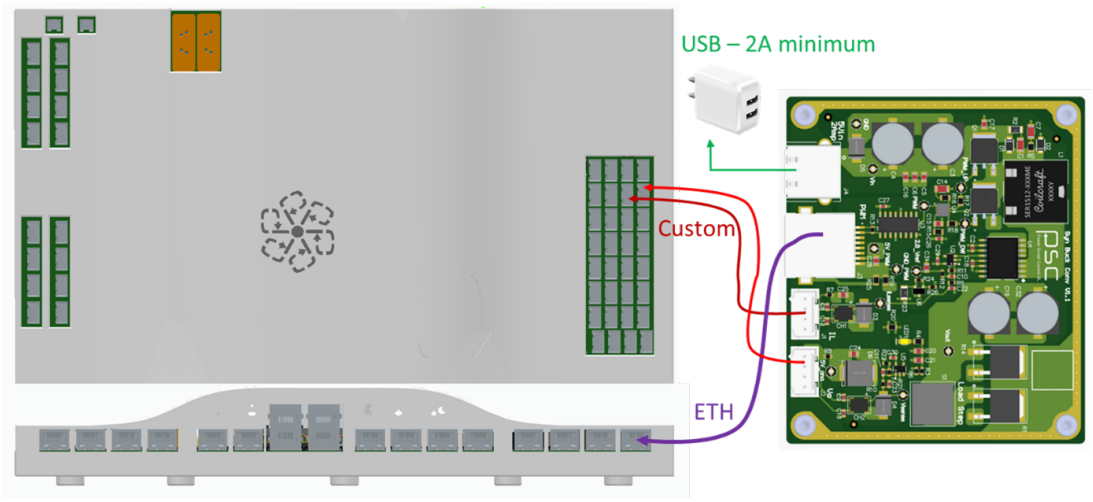


Figure 94: Connection among RCPBuckBoard and SmartRCP.

measured. It shall be noted that both signals are not referred to the same point so two isolated voltage proves are required or a short-circuit will be performed among grounds. 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.

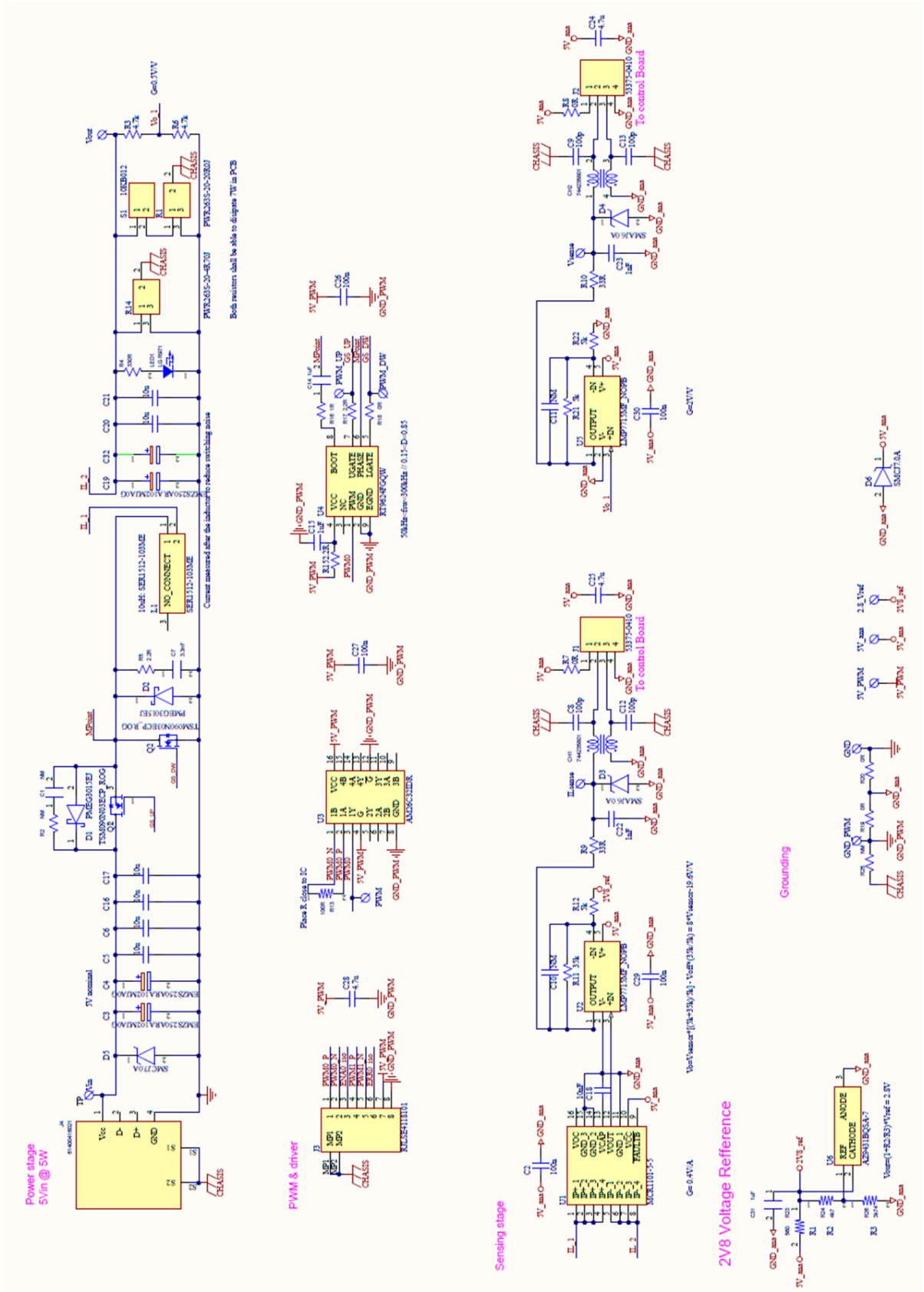


Figure 95: RCPBuckBoard schematic.

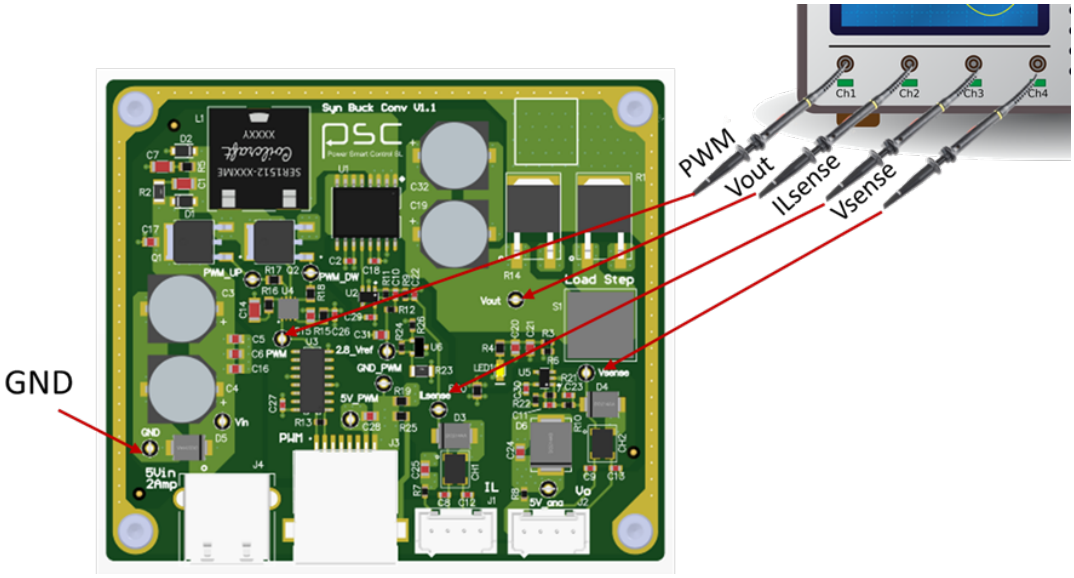


Figure 96: RCPBuckBoard connection to a desktop oscilloscope: PWM, output voltage and sensing voltage.

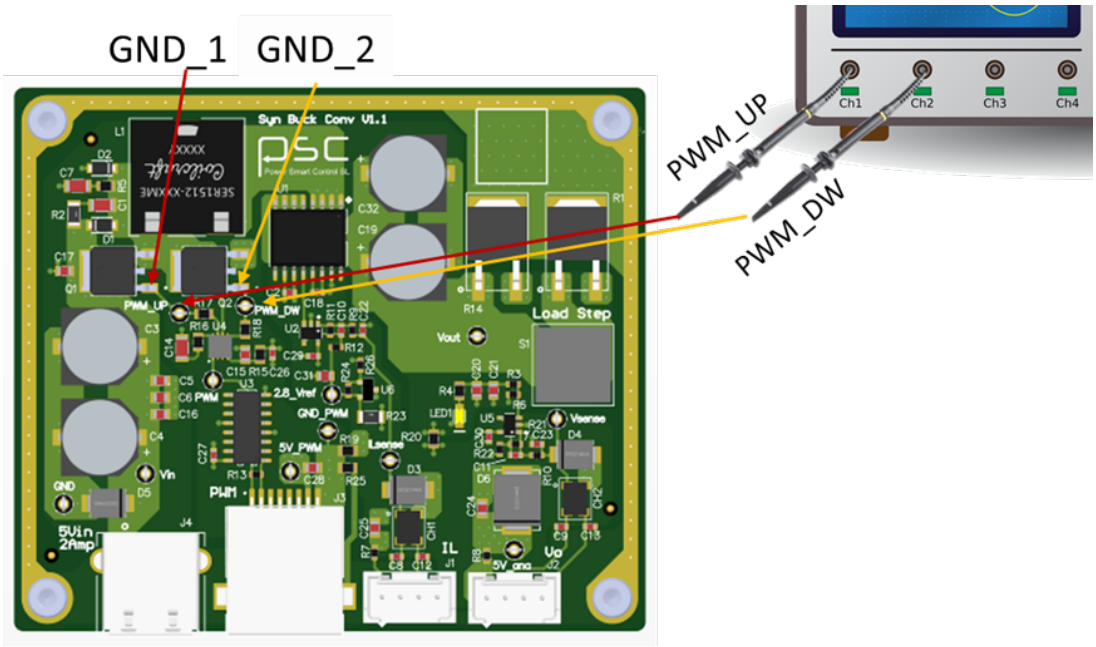


Figure 97: RCPBuckBoard connection to a desktop oscilloscope: PWM pulses of both transistors.

7.2 Workflow for customizing the RCP_TEMPLATE

This section sets out the recommended workflow for SmartRCP users to adapt the RCP_TEMPLATE to the specific needs of their projects. Throughout the steps described, the design of an average output-voltage control for a buck converter is used as the example. Figure 98 summarises the workflow; the following sections explain each stage in detail.

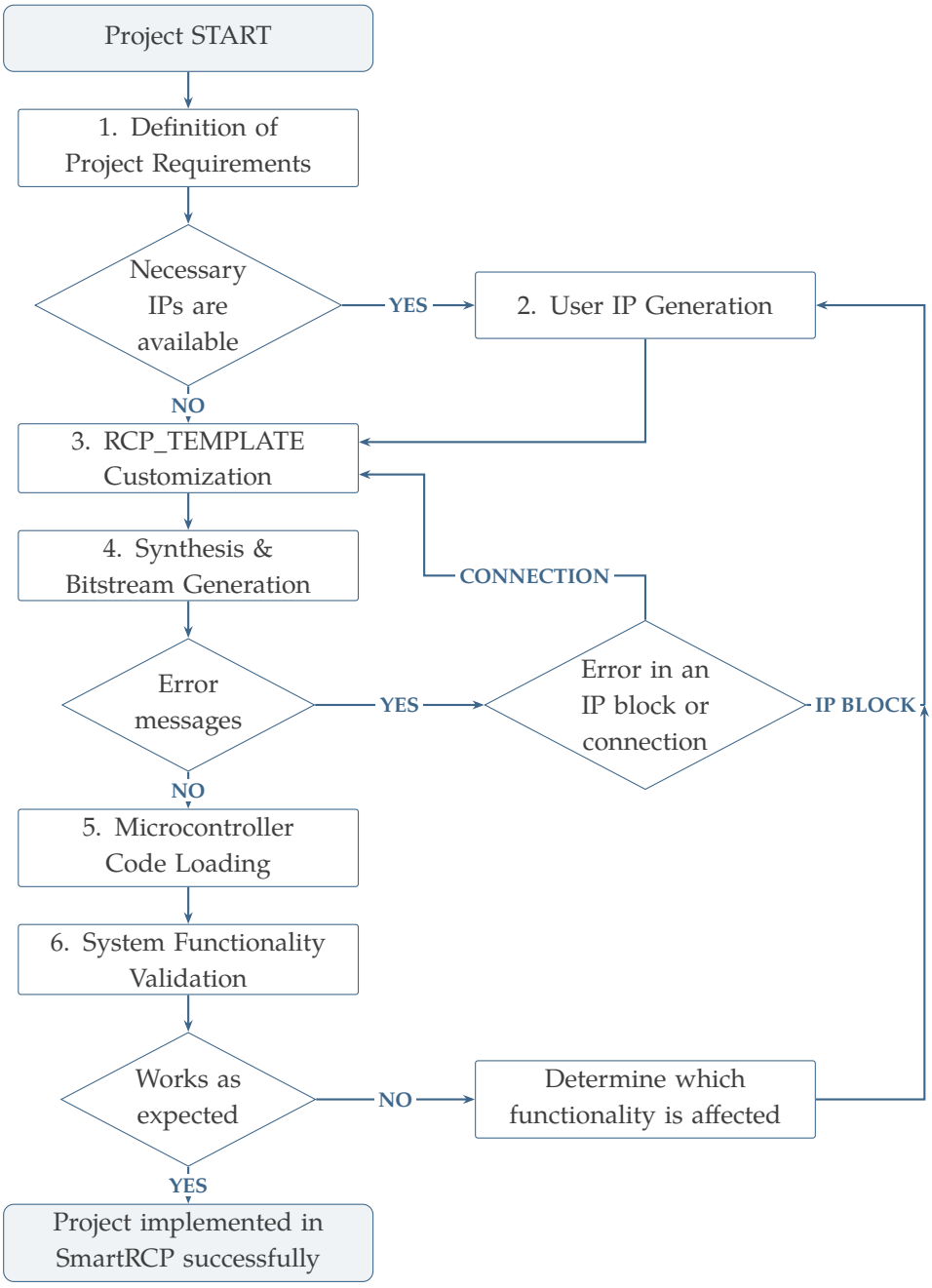


Figure 98: Recommended workflow for customizing the RCP_TEMPLATE to a new project. The loops on the right return to stage 3 when a connection is wrong and to stage 2 when an IP block has to be corrected.

1. **Definition of the SmartRCP functional requirements and resources.** The first stage is to identify the functional requirements of the project and the SmartRCP system resources needed to meet them. This includes specifying the control parameters, the interfaces, the required IPs and the hardware resources that will be used to implement the design.
2. **Generation of custom user IPs.** In this phase the specific IPs required by the project are developed. The stage is detailed in section 7.3, where the generation

of an IP from C++ code is presented; one of the IPs needed to control the buck converter is used to illustrate the step.

3. **Customization of the RCP_TEMPLATE.** This stage focuses on adding the user IPs to the design and connecting them. Section 7.4 explains how to include the required IPs and how to make the appropriate connections in the Vivado block editor.
4. **Validation, synthesis and implementation of the project in Vivado.** Here the design is synthesized and the bitstream is generated: the file that holds the binary configuration of the FPGA, used in the implementation stage of the project. Section 7.5 details the steps to generate and export the bitstream that will then be used to configure the SmartRCP hardware.
5. **Code generation and loading for the microprocessors.** Section 7.6 explains how to generate or load the code executed by the system microprocessors. In this case the preconfigured application that communicates with the SmartRCP software interface is used.
6. **Final design validation and generation of the boot files.** The final stage consists of validating the complete design and generating the boot files required to initialise the device. Section 7.7 describes how to load the code through the JTAG port, the usual method during development, and how to boot from a `BOOT.bin` file stored on the microSD card. Note that a design loaded over JTAG is lost when the device is powered off or rebooted, whereas the microSD card lets the system run the program automatically at every boot. Projects that use the software interface, such as this one, need an additional customization step, described in section 7.8: the configuration of the ADC gains and offsets, the alarm parameters and the other control settings required by the operational requirements of the system.

7.3 Developing IPs for Vivado: C code and Vitis HLS

This section describes the procedure to generate an IP in Vitis HLS according to the functional requirements demanded by the project.

7.3.1 Project introduction and functional blocks

The objective is to implement in the SmartRCP a buck converter control system whose main control parameter is the average voltage at the output of the converter. The software interface allows the user to modify the control parameters, to manage the operating mode (open or closed loop), to adjust the switching frequency and so on. In addition, the user can monitor the waveforms sampled by the ADCs and those gener-



ated internally, through the DataLogger function of the SmartRCP (an oscilloscope-like functionality).

The system offers open-loop control, where the user directly sets the duty cycle and the switching frequency, and closed-loop control, where the user sets the PID controller parameters (proportional, integral, derivative) and the voltage reference. Configurable limits are also defined for the maximum and minimum switching frequency and for the maximum and minimum admissible voltage of the system, all of them manageable from the software interface.

Functional blocks and required IPs

Figure 99 shows the functional blocks that constitute the IPs required to implement the buck converter control system.

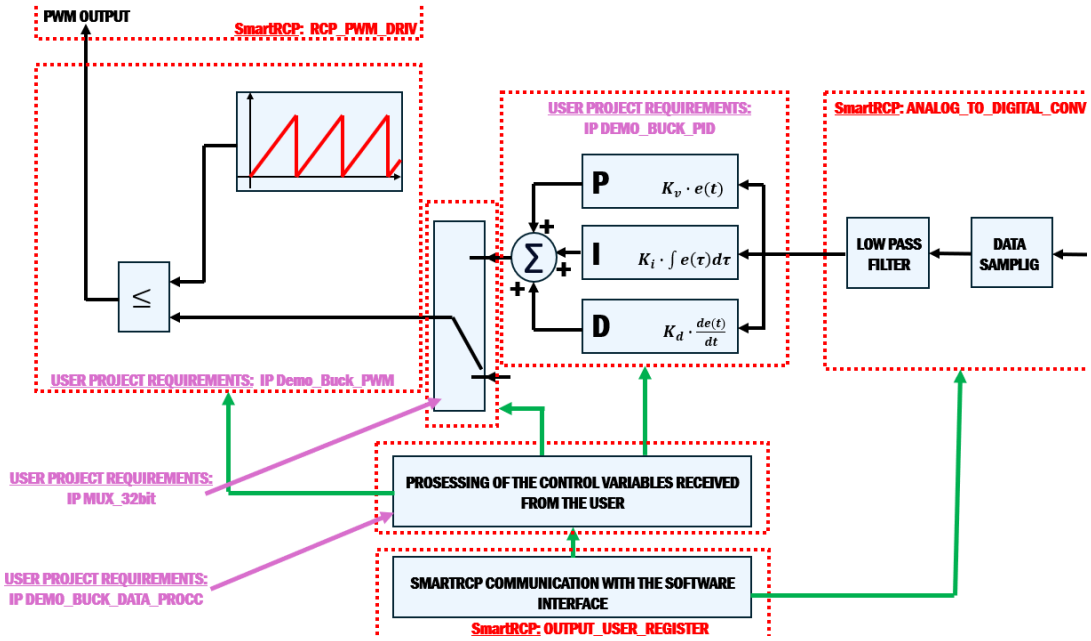


Figure 99: Functional blocks of the buck converter control: user IPs (PWM, PID, data processing, multiplexer) and the SmartRCP drivers they connect to.

All the IPs developed in this project, together with their source code, the Vitis HLS configuration files and the C++ code, are available in the `IP_HLS_SRC` folder of the `SmartRCP_Demo_Buck` project archive. This archive contains every file generated as a result of this tutorial.

7.3.2 IP generation with Vitis HLS: the DEMO_BUCK_PWM block

In this step Vitis HLS is used to generate the IPs that implement the functionalities required by the project. The main focus is on the creation of the PWM generation IP,

detailing the specific procedure in Vitis HLS. For the rest of the IPs used in the project a general description of their operation, interface and role is given, omitting the generation process, which is the same as for the PWM IP.

The process to create an IP in Vitis HLS is described with the `DEMO_BUCK_PWM` block as the example, so that the user understands the complete procedure to transform a control algorithm into Vitis HLS compatible C++ code. Once generated, the IP block can be embedded in the RCP_TEMPLATE and adapted to the functions required by the design.

IP block function specification

- **Carrier signal generation (sawtooth).** The IP block generates a sawtooth carrier whose frequency can be adjusted by the user. The carrier uses the floating-point data type.
- **Signal comparison.** The carrier is compared with a modulating signal to define the output level. When the carrier is lower than the modulating signal the output is set to logic '1' (the MOSFET of the buck converter is turned on); otherwise it is set to logic '0' (the MOSFET is turned off). The modulating signal is an input parameter of the block, also of floating-point type. Figure 100 shows the three signals.

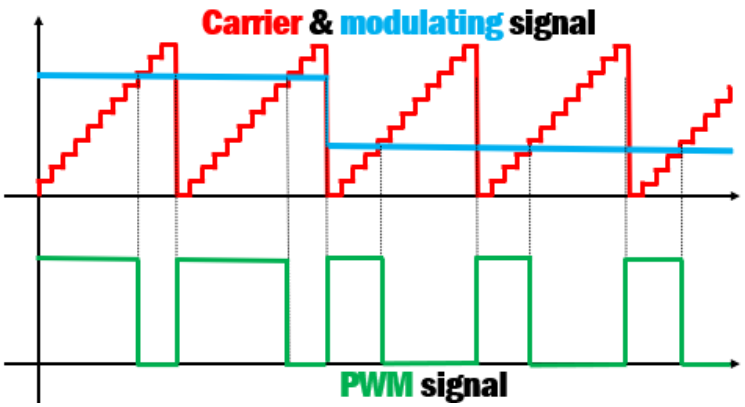


Figure 100: Sawtooth carrier, modulating signal and the resulting PWM signal.

C++ implementation for IP generation

Using C++, code is developed that meets the requirements specified in the previous phase. The code includes the specific configurations needed for the IP block to be generated correctly in Vitis HLS. Figure 101 shows the annotated source of the `DEMO_BUCK_PWM` block.

When developing the code in Vitis HLS, some key technical aspects must be considered:

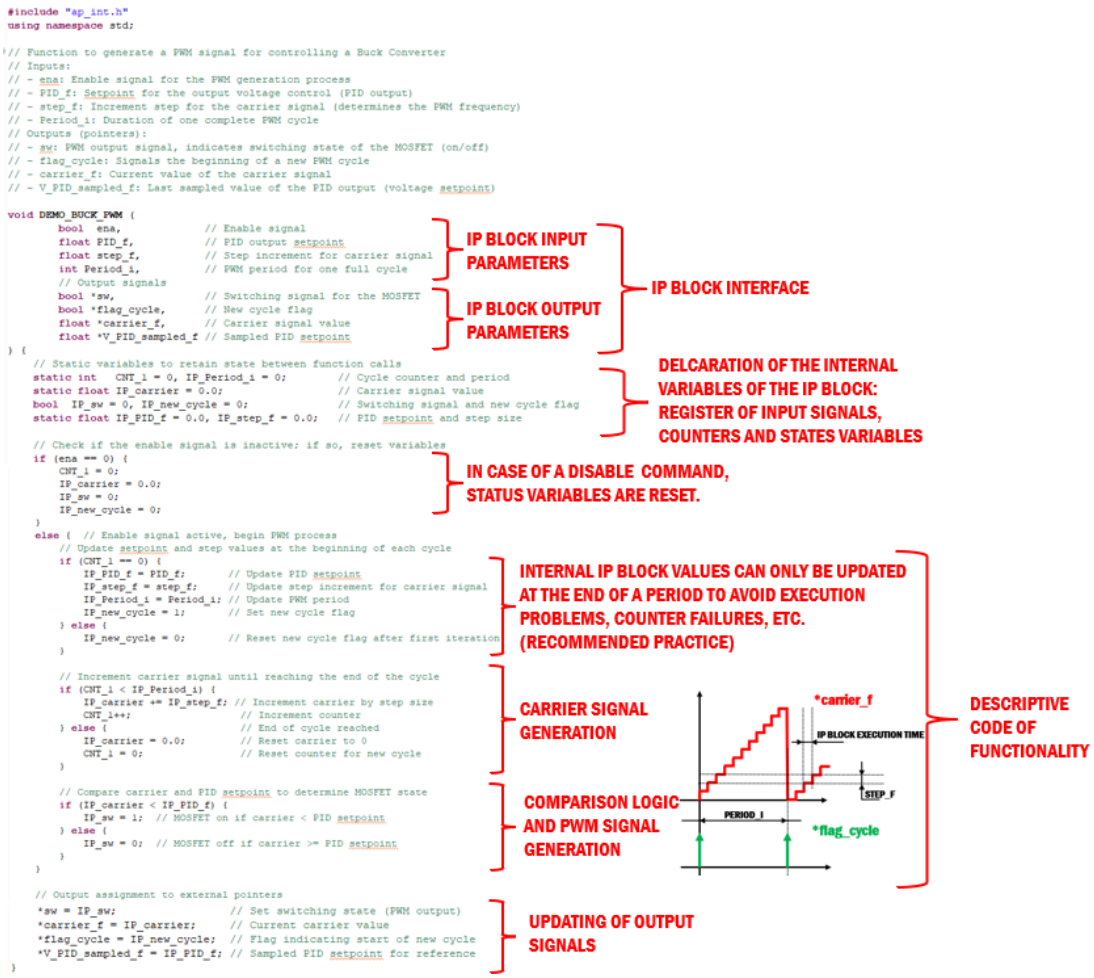


Figure 101: C++ source of the DEMO_BUCK_PWM block: interface, internal state variables, carrier generation, comparison and output assignment.

- **Declaration of input and output variables.** In the main function that defines the IP, the output variables are distinguished by an asterisk next to the variable name (they are pointers), while the input variables are declared in the conventional way.
- **Behaviour of the variables between executions.** By default the code of the IP block executes sequentially, and every variable not declared as persistent is reset to 0 at each execution cycle. For a variable to retain its value between execution cycles it must be declared `static` before its data type.
- **Control through directives.** Vitis HLS allows the use of directives that configure advanced aspects of the IP, such as the execution mode (sequential, concurrent or mixed), the resource allocation in the synthesis and the maximum number of clock cycles per execution. Consult the Xilinx documentation and resources to optimise these settings and to maximise the performance of the IP blocks developed for your projects.



Project creation and configuration in Vitis HLS

In this step the project for the `DEMO_BUCK_PWM` IP is created in Vitis HLS, configuring the synthesis options, the resource allocation and the clock constraints. This ensures that the IP meets the operational requirements and is ready to be integrated into the RCP_TEMPLATE. Follow the steps of Figures 102 and 103.

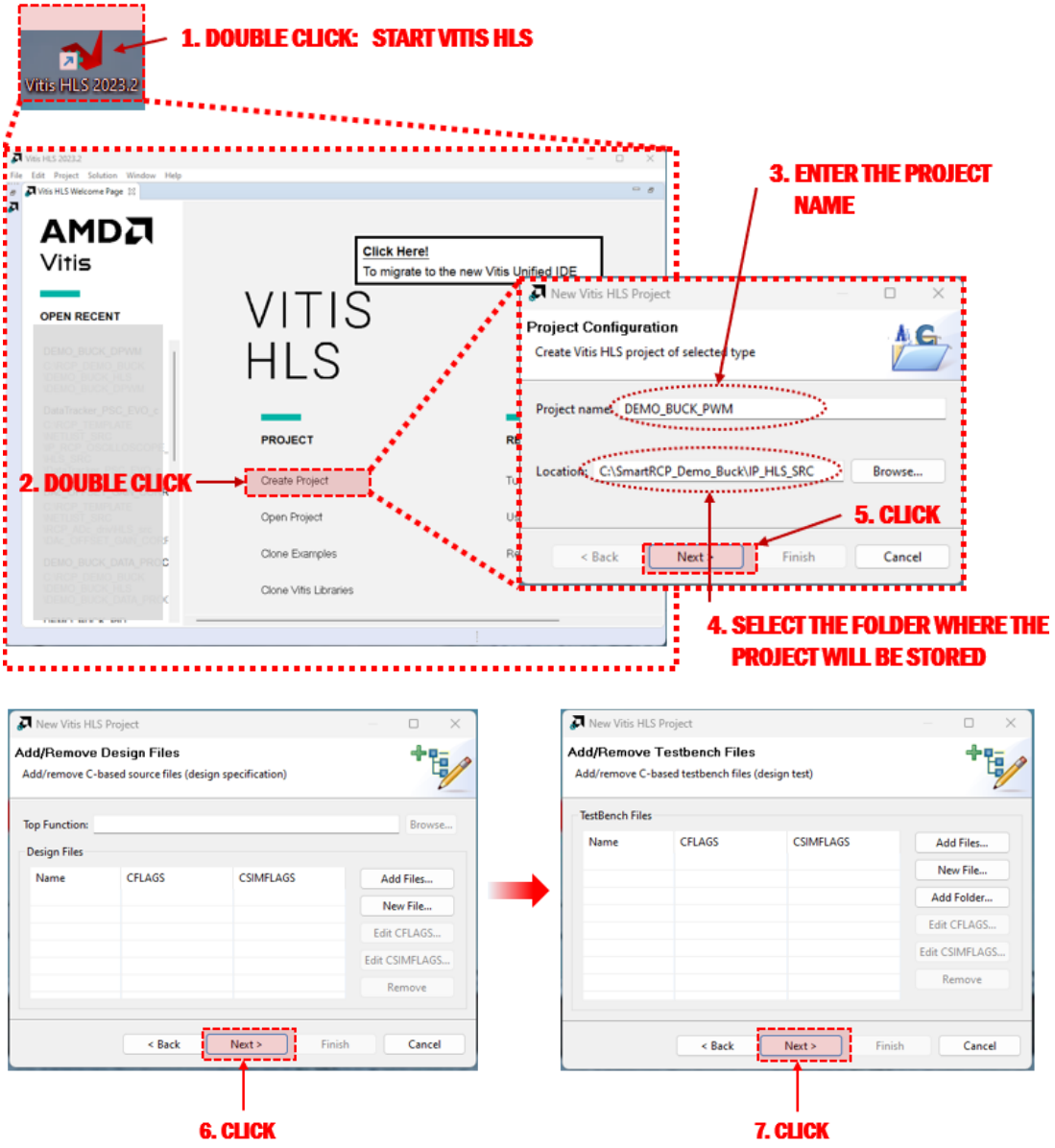


Figure 102: Vitis HLS project creation, steps 1 to 7: launching Vitis HLS, naming the project and its location, and skipping the design and testbench file pages.

.cpp file creation and project setup in Vitis HLS

In this step the C++ (`.cpp`) file that implements the functionality of the IP block is created, and the Vitis HLS workspace is configured. The file holds the code of the IP block

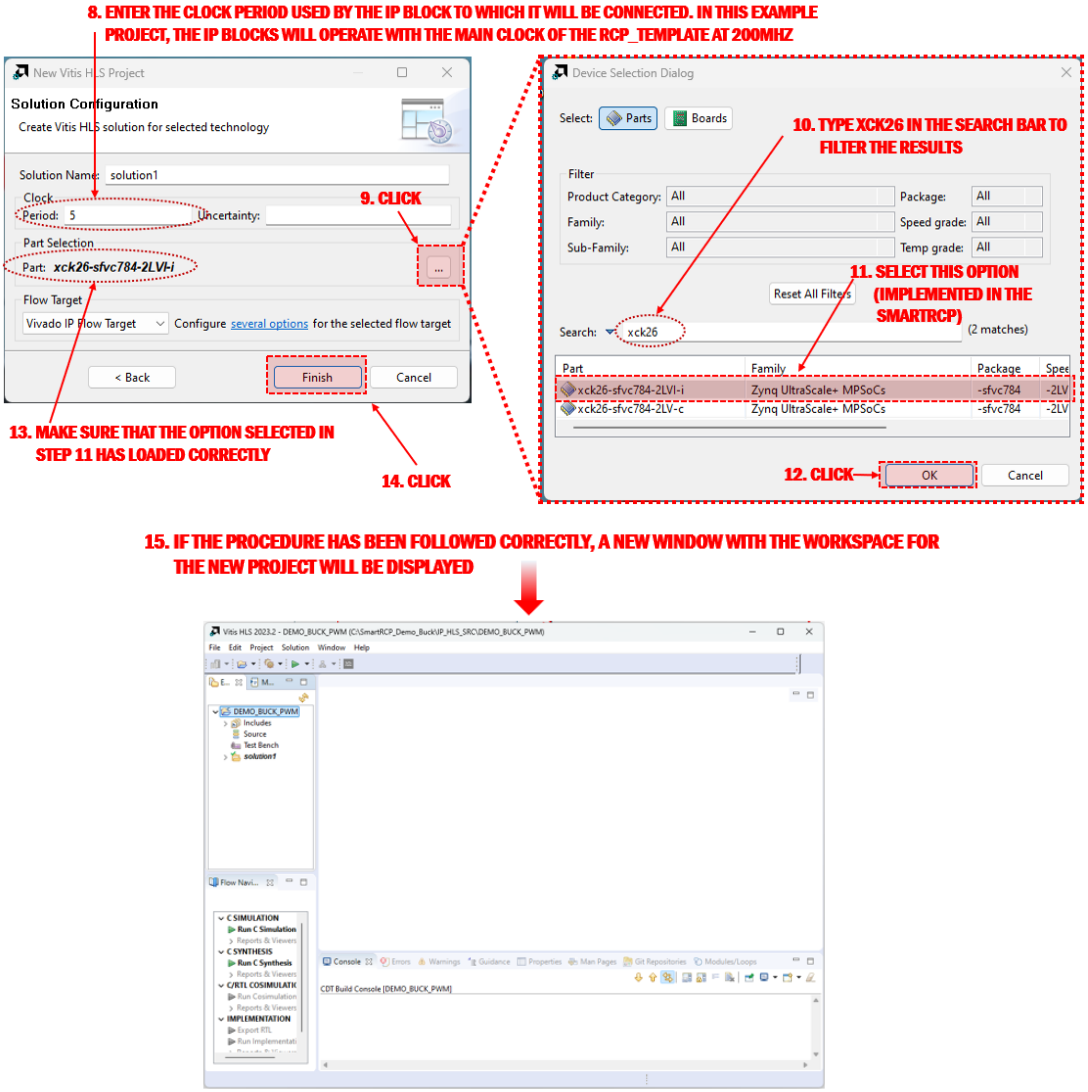


Figure 103: Vitis HLS project creation, steps 8 to 15: clock period of the RCP_TEMPLATE main clock, selection of the xck26 part and the new project workspace.

developed in the previous steps and should be named after the project (`<project>.cpp`). Although it is not mandatory, it is recommended that the name of the main function in the code matches the project name, because it will be the identifier of the IP when it is exported to or imported into Vivado.

Next, the Vitis HLS workspace is configured to designate this function as the top function of the project, so that the tool can generate the IP block correctly. Follow the steps of Figures 104 and 105.

Configuration of directives in Vitis HLS

In Vitis HLS, directives are specific configurations that let the user control different aspects of the synthesis of the IP block, optimising resources and adapting the design to

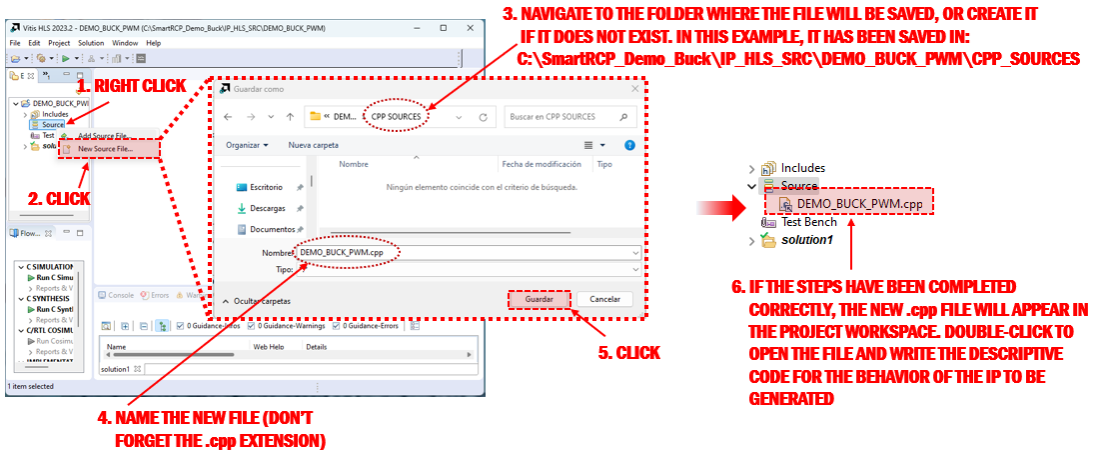


Figure 104: Source file creation, steps 1 to 6: adding a new source file to the project and naming it with the .cpp extension.

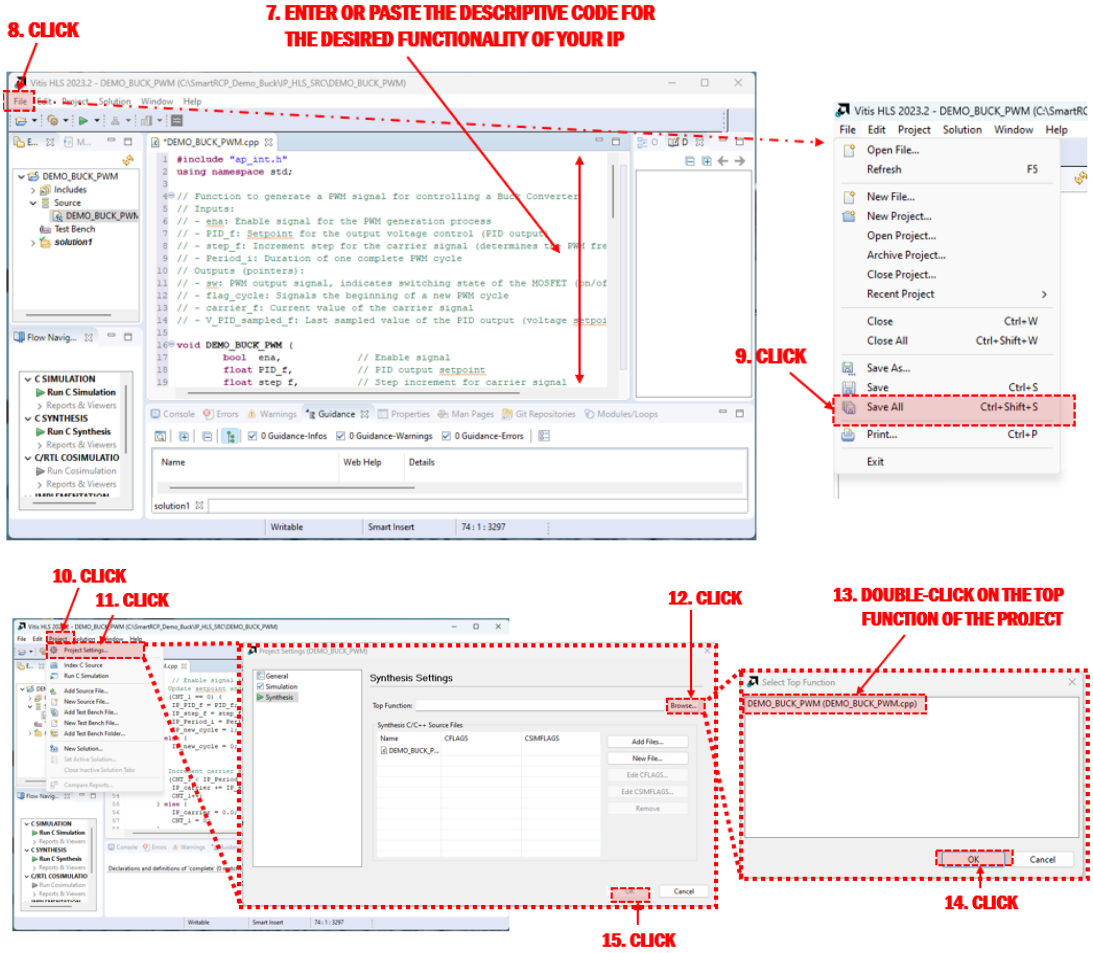


Figure 105: Source file creation, steps 7 to 15: pasting the descriptive code, saving, and selecting the top function in the project settings.

specific requirements. Directives do not alter the functional code; they adjust parameters and behaviours during the synthesis. They cover preferences and requirements

ranging from latency control to the management of the data interfaces, which increases the versatility of the generated IP blocks. Some common directives are:

- **Pipeline.** Sets up a pipeline to optimise latency and throughput by reducing the execution time of loops. Useful when a continuous flow of data is required.
- **Unroll.** Expands loops so that several iterations execute in parallel, optimising the use of resources when the design requires high processing speed.
- **Interface.** Controls how data enters and leaves the IP block, allowing the configuration of the communication interface, such as AXI4-Stream or AXI4-Lite.
- **Resource.** Defines the type of resource used for a specific operation (for example, LUT or DSP multipliers), optimising the hardware usage according to the design objectives.
- **Latency.** Specifies the maximum latency allowed for an operation, useful to adjust the response of the IP to the system requirements.

To go deeper into the use of directives and to get the most out of Vitis HLS, Xilinx offers detailed guidance and tutorials in its Vitis HLS documentation and in the Xilinx Learning Center, with resources on the design flow and on advanced optimisation strategies. In this project a directive that simplifies the IP interface is used, as shown in Figures 106 and 107.

IP synthesis, implementation and export in Vitis HLS

In this step the C++ code is synthesized, which gives a first analysis of its validity and an estimate of the FPGA resources the IP will need. The process verifies the adequacy of the code, points out possible optimisations and evaluates the fulfilment of the design requirements. Once the synthesis and its validation are complete, the IP is implemented and exported for integration into the user's Vivado project. Start the synthesis as shown in Figure 108.

The synthesis can take a variable amount of time, depending on the specifications of the computer. At the end, the tool generates a results report in the centre of the work area (Figure 109). The report gives the resources used and the latency of the IP block, crucial parameters to ensure the proper synchronization of the design. Bear in mind that the IP block with the highest latency conditions the synchronization of the other IP blocks, which must be taken into account during the final integration and, if necessary, in the design and redesign of the operating logic of the project.

Finally, the synthesized design is implemented, which makes it importable into Vivado. The duration of this process also depends on the computer. Follow the steps of Figure 110.



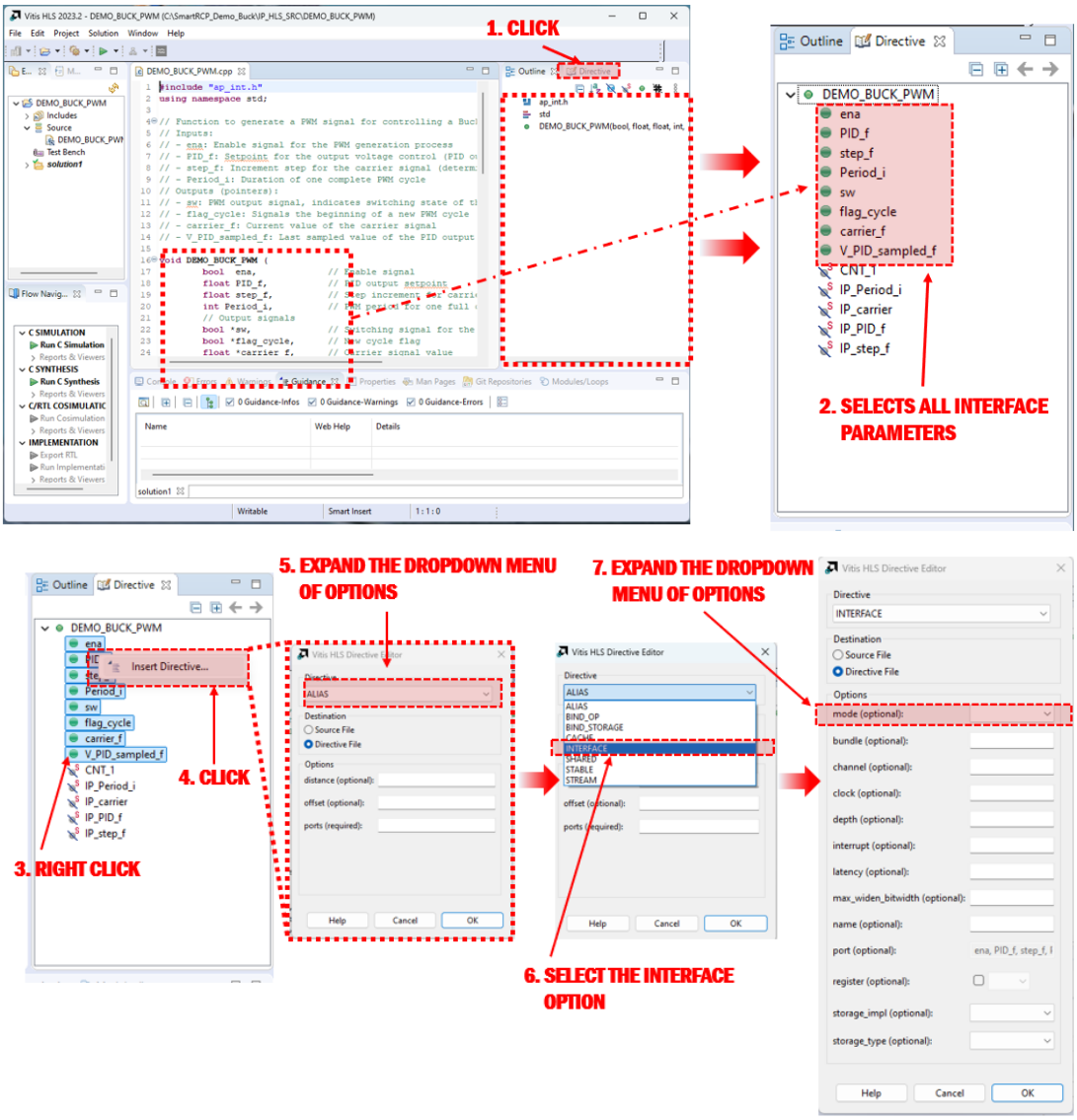


Figure 106: Interface directive, steps 1 to 7: selecting all the interface parameters in the Directive tab and choosing the INTERFACE directive.

At the end of the implementation a new report appears in the work area (Figure 111), showing the timing compliance and the resources consumed by the IP block. These are the final values of the generated IP.

Final considerations on IP generation in Vitis HLS

When the IP is imported into Vivado and added to the block diagram it appears as in Figure 112.

In addition to the input and output signals specified in the header of the descriptive function, the IP block includes a set of signals generated automatically by Vitis HLS. These signals, grouped in the `AP_CTRL_HS` interface, manage the control and synchro-

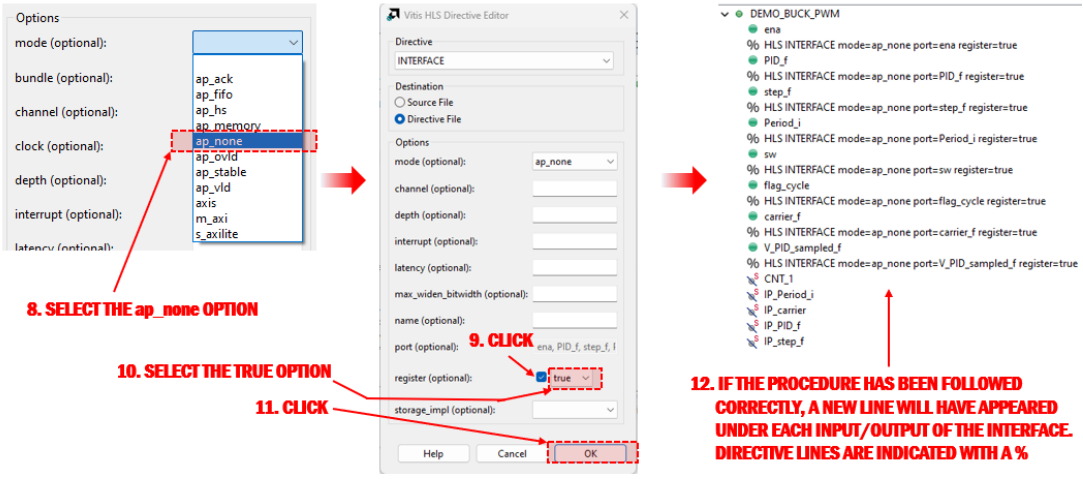


Figure 107: Interface directive, steps 8 to 12: mode `ap_none`, register option set to `true`, and the resulting directive line under each port.

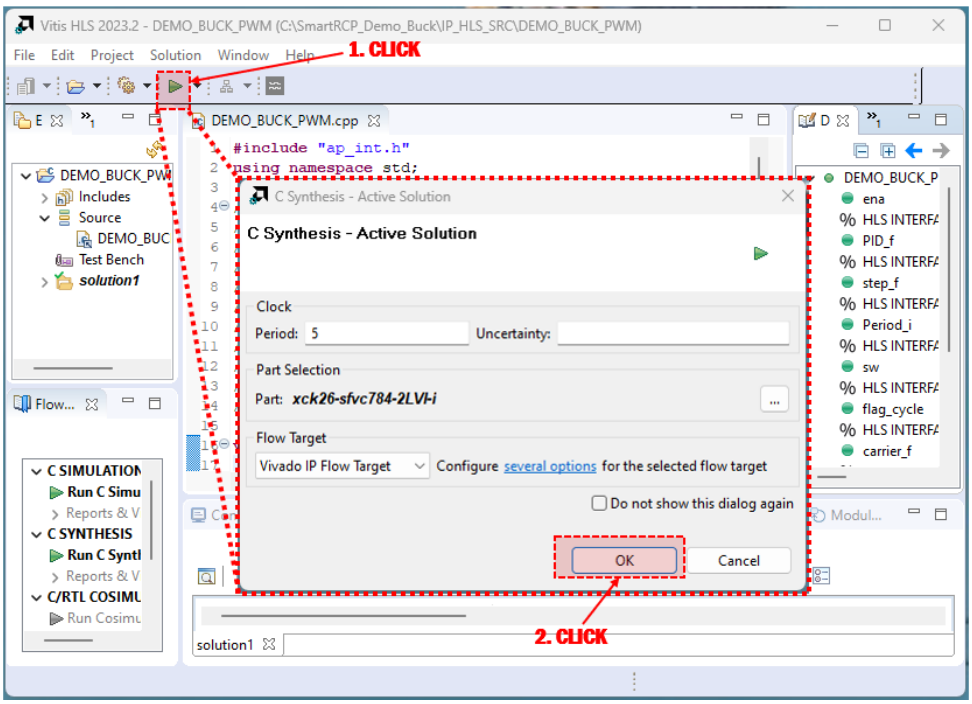


Figure 108: C synthesis, steps 1 and 2: the Run C Synthesis button and the solution dialog, with the clock period and the selected part.

nization of the IP block: the clock, the reset and other control signals. Figure 113 summarises their functions.

This guide has covered the generation of an IP block compatible with the RCP_TEMPLATE, starting from specific functional requirements. For the other IPs of the project a description of their operation follows; all the code and the corresponding Vitis HLS projects are located in the `IP_HLS_SRC` folder. The content of this tutorial gives SmartRCP users the basis to generate IP blocks from descriptive C++ code, but Vitis

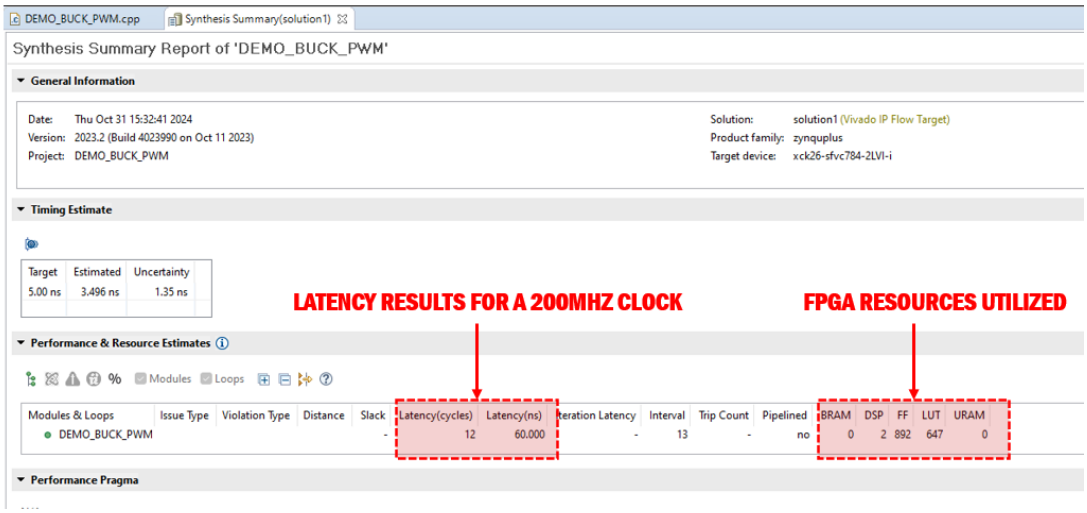


Figure 109: Synthesis summary of DEMO_BUCK_PWM: latency for a 200 MHz clock and FPGA resources used.

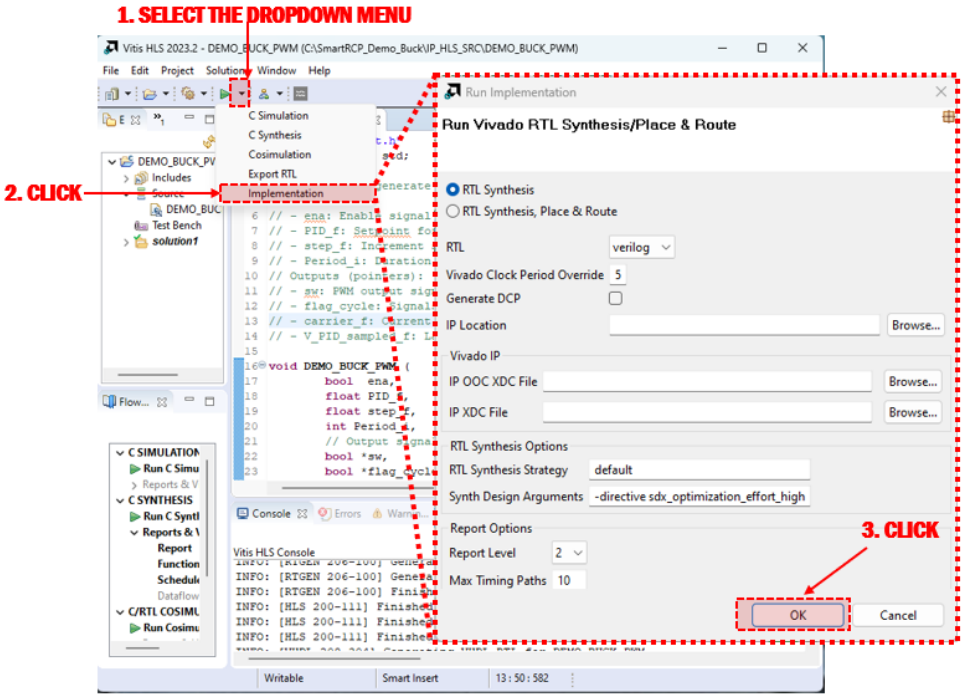


Figure 110: Implementation, steps 1 to 3: the Implementation entry of the run menu and the Run Implementation dialog.

HLS offers a much wider range of functionality: consult the technical documentation and the additional resources available online to make the most of the tool.

7.3.3 The DEMO_BUCK_PID block

This IP block implements a PID compensator that regulates the output voltage of the buck converter. The conversion of the descriptive code into an IP block follows the same



DEMO_BUCK_PWM.cpp | Implementation(RTL Synthesis)(solution1)(DEMO_BUCK_PWM_export.rpt)

Project: DEMO_BUCK_PWM
Solution: solution1
Device target: xck26-sfvc784-2LVI-i
Implementation tool: Xilinx Vivado v.2023.2

Run Constraints & Options

Name	Value
Design Constraints & Options	
RTL Synthesis Options	
Reporting Options	

Resource Usage

	Verilog
SLICE	0
LUT	462
FF	942
DSP	2
BRAM	0
URAM	0
LATCH	0
SRL	3
CLB	0

Final Timing

	Verilog
CP required	5.000
CP achieved post-synthesis	2.625

Timing met

Figure 111: Implementation report of DEMO_BUCK_PWM: resource usage and final timing.

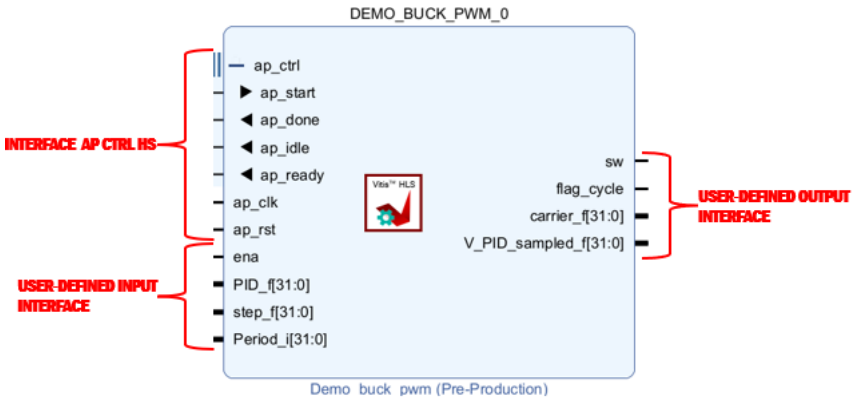


Figure 112: The DEMO_BUCK_PWM block in Vivado: AP_CTRL_HS interface, user-defined inputs and user-defined outputs.

steps detailed for the PWM block. Both the source code and the Vitis HLS project of the PID block are located in `SmartRCP_Demo_Buck\IP_HLS_SRC`, under the name `DEMO_BUCK_PID`. All the code is commented in detail to facilitate its analysis. Figure 114 shows the block diagram of the C++ code developed for this project.

In the diagram, the signals marked with dashed arrows are the inputs of the IP block, all of them in floating-point format. The only output of the block, `u`, also floating point, is the input of the PWM block, where it acts as the modulating signal of the PWM pulses: `u` is compared with the sawtooth generated internally in the PWM block to regulate the duty cycle of the MOSFET. Figure 115 shows the block as Vivado draws it and Figure 116 the most relevant results of its synthesis and implementation.

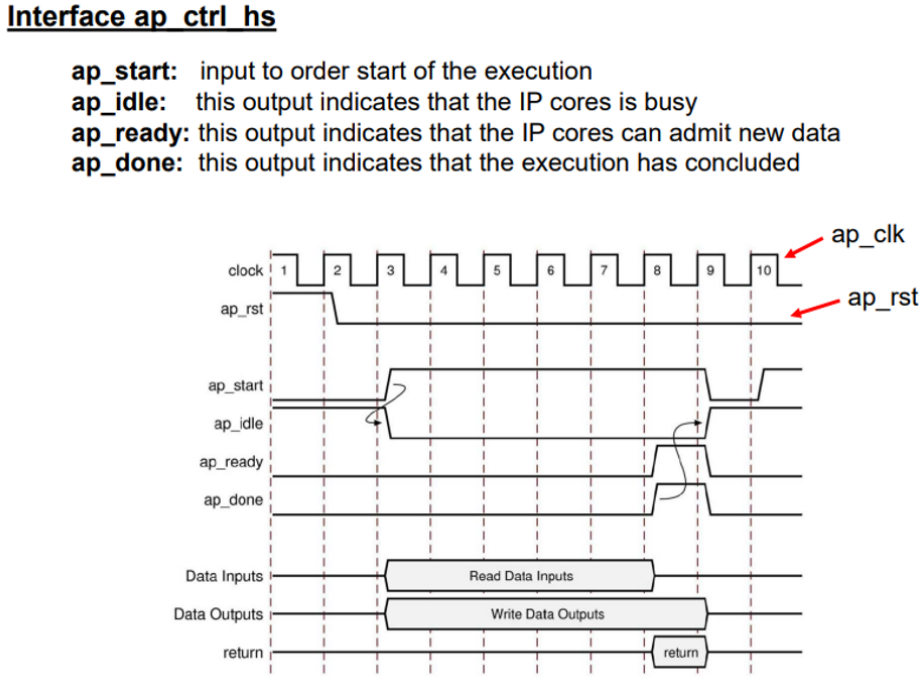


Figure 113: The `ap_ctrl_hs` interface: meaning of `ap_start`, `ap_idle`, `ap_ready` and `ap_done`, and their timing over one execution.

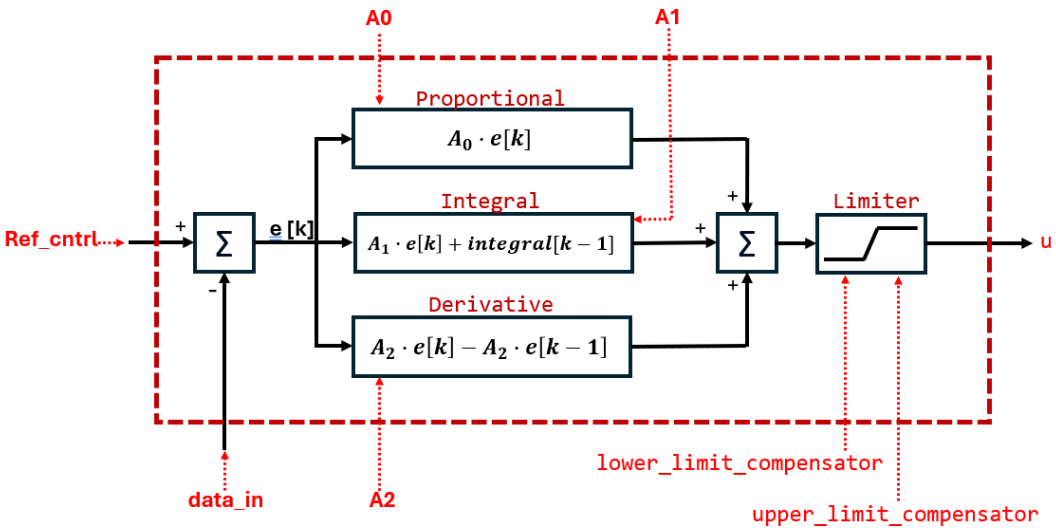


Figure 114: Block diagram of the DEMO_BUCK_PID code: proportional, integral and derivative terms with coefficients A0, A1 and A2, and the output limiter.

7.3.4 The DEMO_BUCK_DATA_PROCCESSING block

This IP block is located in `SmartRCP_Demo_Buck\IP_HLS_SRC` under the name `DEMO_BUCK_DATA_PROCCESSING`; both the source code and the Vitis HLS project are available for analysis and modification. The block was implemented following the procedure described for the PWM IP block.

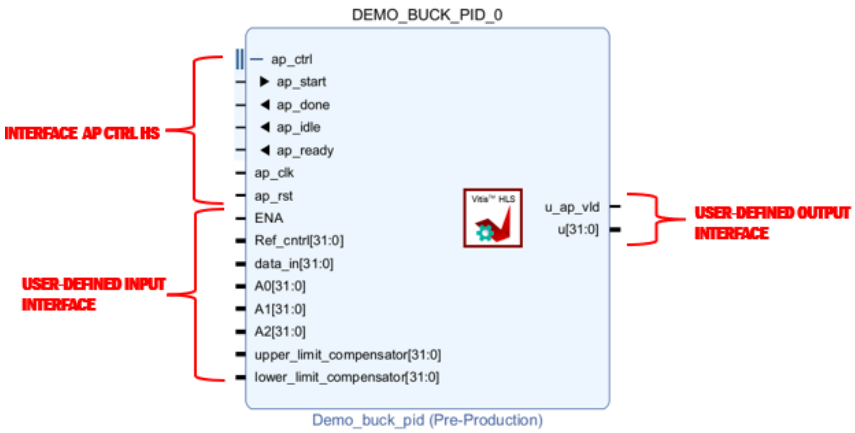


Figure 115: The DEMO_BUCK_PID block in Vivado.

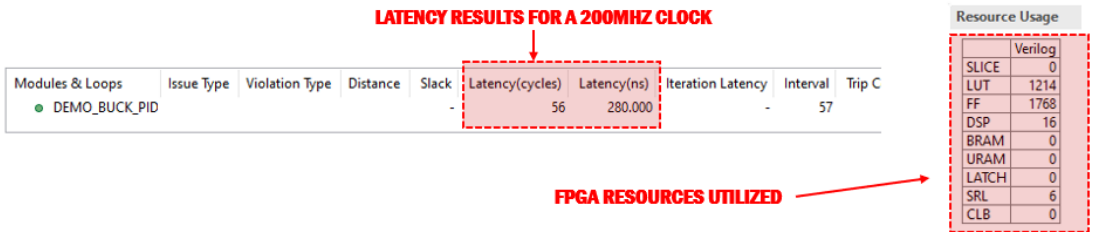


Figure 116: DEMO_BUCK_PID: latency for a 200 MHz clock and FPGA resources used.

The block processes the high-level data entered by the user in the SmartRCP software application and converts it into the parameters required by the project IP blocks. Its functionality includes:

- **PID coefficient conversion.** Transforms the high-level PID variables entered in the software interface, K_p, T_d and T_i, into the coefficients A0, A1 and A2 used by the PID equation implemented in the IP block.
- **Switching frequency calculation.** Converts the switching frequency selected by the user into its equivalent in clock cycles, adjusted to the parameters required by the FPGA.
- **Duty cycle to reference voltage conversion.** In open-loop mode, adapts the duty cycle reference, in %, to a voltage signal for the PWM, which acts as a direct reference (modulating signal).

In short, the block converts the user input data into a valid, optimised format for interpretation and processing in the FPGA. Figure 117 shows the block in Vivado and Figure 118 the most relevant results of its synthesis and implementation.



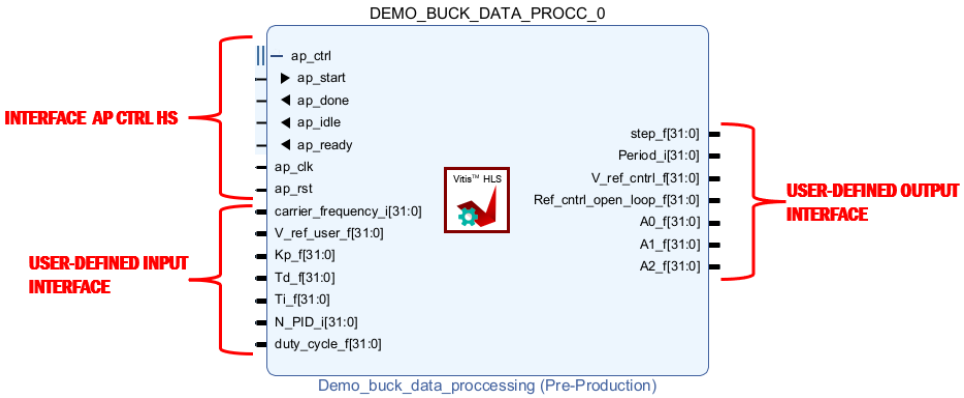


Figure 117: The DEMO_BUCK_DATA_PROCC block in Vivado.

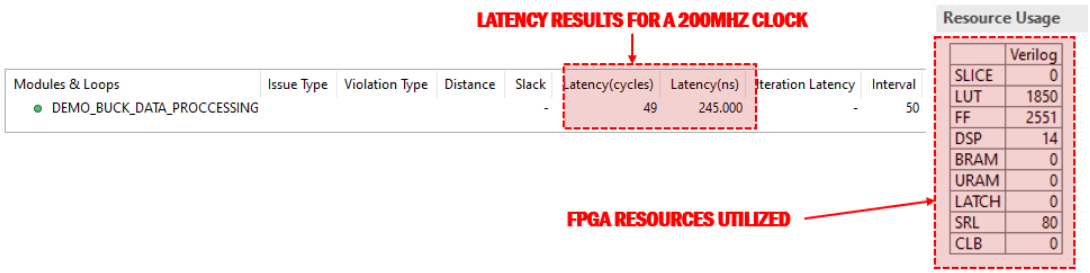


Figure 118: DEMO_BUCK_DATA_PROCCESSING: latency for a 200 MHz clock and FPGA resources used.

7.4 RCP_TEMPLATE customization: adding and interconnecting user IPs

This section details the process to adapt the Vivado project to the specific needs of the user, with the buck converter project as the example. It is organised as follows:

1. **Project setup and renaming of the Vivado template.** The Vivado project is unzipped and renamed. The tutorial starts from the default RCP_TEMPLATE archive, which is adapted to the project requirements. The final files of the example are delivered in the `SmartRCP_Demo_Buck` archive for reference.
2. **Introduction to the Vivado workspace environment.** The project is opened and the working environment of Vivado is introduced: the basic concepts and the essential elements of the workspace a user needs to modify the project. This guide includes the minimum information required; users are encouraged to consult the documentation, tutorials and guides that Xilinx provides to take full advantage of the tool.
3. **Integrating the user-developed IPs from Vitis HLS.** The steps to include the IPs generated in Vitis HLS in the Vivado environment, and how to make the connections between those IPs and the RCP_TEMPLATE drivers.

4. **Synchronization of the IPs with a custom VHDL block.** The IPs developed in HLS need to be synchronized. A VHDL block dedicated to this synchronization is designed within the interconnection step, so that all the IPs operate coherently and with the proper timing.
5. **Adding the ILA IP block for design debugging.** The Integrated Logic Analyzer (ILA) block is added to the project. It is useful to debug the design, with a functionality similar to an oscilloscope, but it has limitations in the interpretation of floating-point data, the number of samples, the refresh rate and the communication, which is only through JTAG.
6. **Summary and final user considerations.** A brief summary and recommendations to optimise the use of Vivado in this type of project, together with additional considerations for future modifications and improvements.

The `SmartRCP_Demo_Buck` archive (compressed file) contains the final project, already customized, implemented and fully functional, reflecting all the steps and configurations detailed in this tutorial.

7.4.1 Project setup and renaming of the Vivado template

To prepare the working environment and customize it for the new project, the steps are detailed from scratch, so that this section can serve as a generic guide for any future development in Vivado. The process includes decompressing, renaming and adjusting the RCP_TEMPLATE archive in Vivado.

First, unzip the RCP_TEMPLATE file and, when doing so, name it `SmartRCP_Demo_Buck` (Figure 119). It is recommended to decompress it directly on the main disk (for example in `C:\`) or in a folder with few hierarchy levels: an excessively long path may cause errors in Vivado during synthesis and implementation.

There are two ways to launch Vivado and load the `RCP_TEMPLATE.xpr` project file (Figure 120):

- **Direct file opening.** The easiest way is to double-click the `RCP_TEMPLATE.xpr` file in the operating system. Vivado opens and loads the project automatically.
- **Access through the Vivado interface.** Alternatively, open Vivado and select **Open Project**, then navigate to the location of `RCP_TEMPLATE.xpr` and select it.

7.4.2 Introduction to the Vivado workspace environment

This section presents the minimum fundamental concepts a user needs to develop a project in Vivado: the essential elements of the interface and the basic knowledge re-



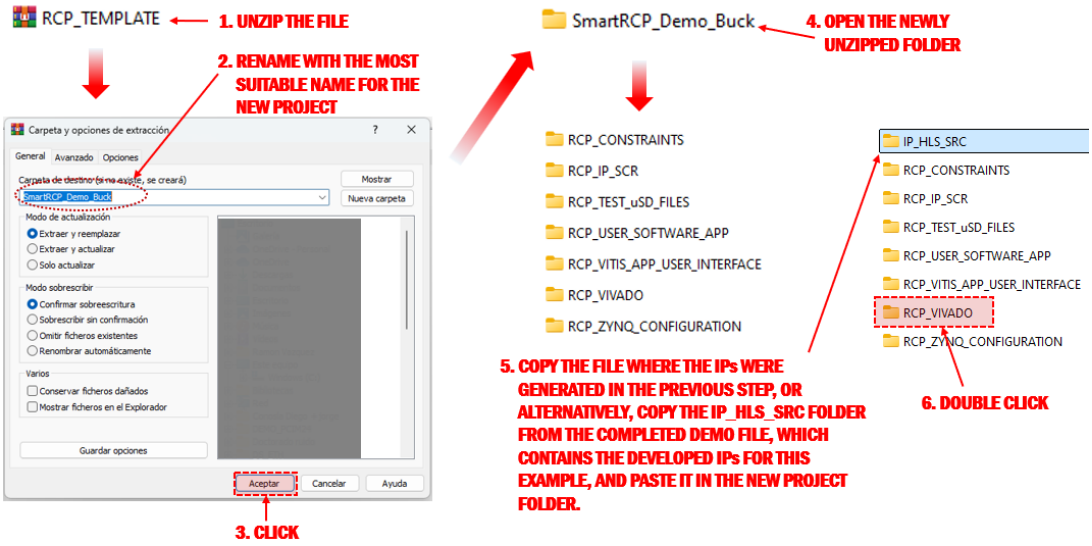


Figure 119: Project setup, steps 1 to 6: unzipping the RCP_TEMPLATE under the new name and copying the IP_HLS_SRC folder into it.

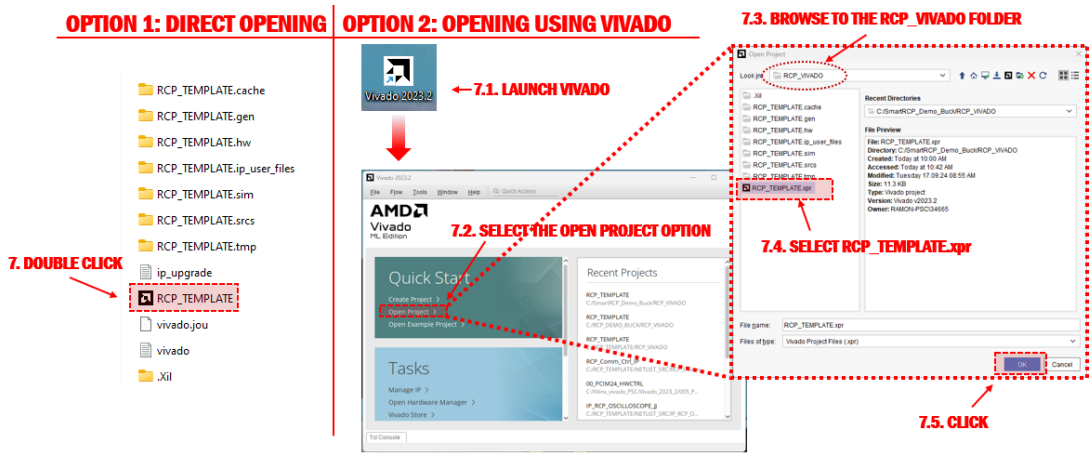


Figure 120: Project setup, step 7: the two ways of opening the project, from the file explorer or from the Vivado Quick Start page.

quired to navigate and use the tool efficiently. Figure 121 marks the five areas described below.

Flow Navigator

The Flow Navigator holds the tools and options essential for the project development. It follows the logical order of the workflow, which makes it easier to navigate and manage the project.

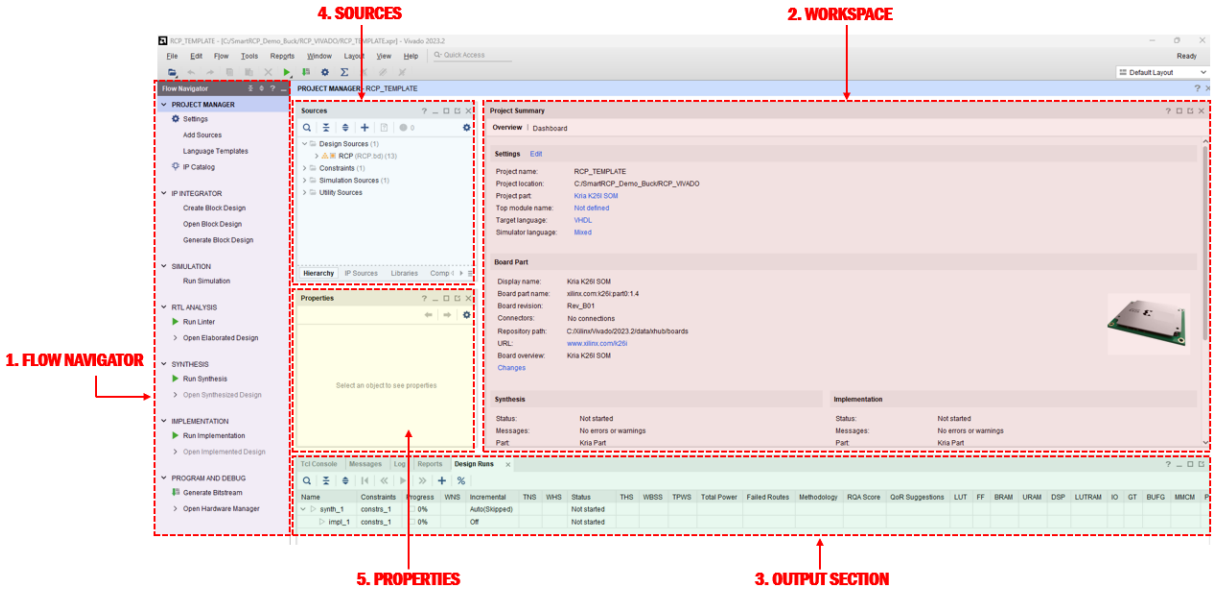


Figure 121: The Vivado window with the RCP_TEMPLATE open: Flow Navigator, Workspace, Output section, Sources and Properties.

Project Manager. In this area the user selects the essential configuration parameters of the project and adds or creates digital designs. The available options include:

- **Settings**: adjusts the synthesis and implementation strategies, prioritising aspects such as signal propagation times or the minimisation of resource usage.
- **Add Sources**: creates or adds new sources to the project. In the example project it is used to generate the VHDL blocks needed for the synchronization.
- **IP Catalog**: opens a new window in the Workspace where the IPs generated in Vitis HLS can be added.

IP Integrator. In this area the block diagrams that establish the connections between IPs are created or edited. The relevant option for RCP_TEMPLATE users is **Open Block Design**: it opens the block design that contains all the basic configurations and the IPs of the SmartRCP drivers, already connected. In this space the user adds their own IPs and makes the connections the project needs.

Synthesis. Once the blocks are connected in the diagram, the synthesis is started from the Synthesis section with **Run Synthesis**. At the end of the process the results are available in **Open Synthesized Design**.

Implementation. With the synthesis complete, the implementation is started with **Run Implementation**. When it finishes, the results of this stage can be reviewed in **Open Implemented Design**.

Program and Debug. This section holds the **Generate Bitstream** option, which creates the file used to program the board and needed in the microprocessor programming stage. If it is selected without having completed the synthesis and implementation, both processes run automatically and sequentially. The section also offers **Open Hardware Manager**, which switches the Workspace to a view focused on design debugging, especially useful when an ILA (Integrated Logic Analyzer) block has been added to the design: the signal viewer of the ILA is accessed from this workspace.

Workspace

The Workspace is the main development area of Vivado, where the windows and editors needed to manage the project appear. Through the workflow blocks, the Workspace gives direct access to the graphical editor in which IP blocks are created and connected visually. Likewise, when VHDL or Verilog code has to be modified, the Workspace shows a text editor for the source code. This area centralises the visualisation and manipulation of every design component: it is the space where the interaction of the user with the elements of the project takes place.

Output section

The Output section is where the results generated during the synthesis, implementation and simulation stages are presented: resource usage, timing and error or warning reports. It allows the user to review the project results, to identify problems and to decide on the optimisations needed before the final implementation.

Sources

The Sources section lists all the VHDL or Verilog blocks created for or added to the project. Blocks can be dragged from here directly into the graphical design. The section also presents a hierarchical view of the elements, showing the relationship between the different IP blocks and the wrapper.



Properties

In the Properties section the user modifies the properties of any element selected in the block diagram, such as the name of the block, which helps to keep the design organised.

7.4.3 Integrating user-developed IPs from Vitis HLS

This section describes the integration of the IPs developed in Vitis HLS and the steps needed to interconnect the IP blocks and the drivers.

Adding the Vitis HLS IP blocks to the project

To begin with, the IP blocks generated in Vitis HLS are added to the project as a repository, following the steps of Figure 122.

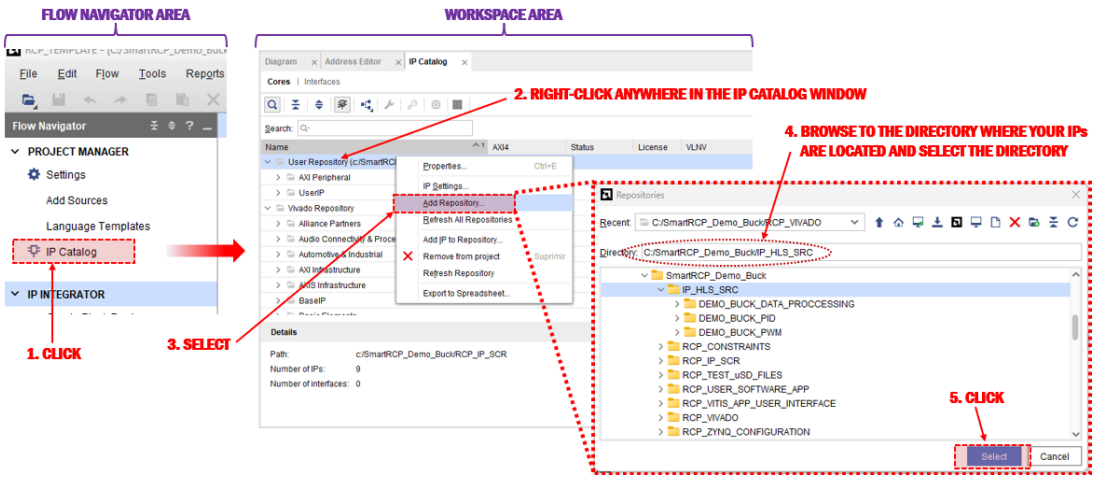


Figure 122: Adding the IP repository, steps 1 to 5: IP Catalog, Add Repository and selection of the IP_HLS_SRC directory.

If the steps were followed correctly, a window pops up indicating the number of IPs recognised as valid in the specified directory (Figure 123). This confirms that the IPs developed in Vitis HLS have been detected and are ready for integration.

Adding the IPs to the block design

For this step the RCP_TEMPLATE block design is opened. It already contains the basic structure needed to operate the SmartRCP: the IP drivers and the Zynq UltraScale+ MPSoC configuration. Add the IPs as shown in Figure 124. If the procedure was successful, three new IP blocks appear in the workspace (Figure 125).

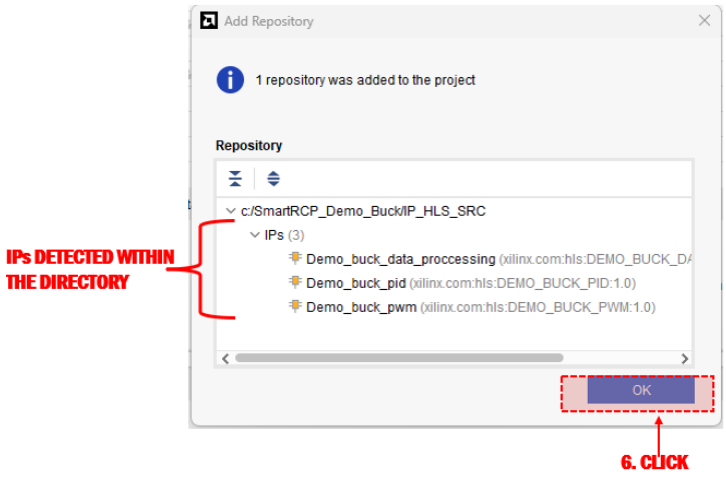


Figure 123: Adding the IP repository, step 6: the three IPs detected in the directory.

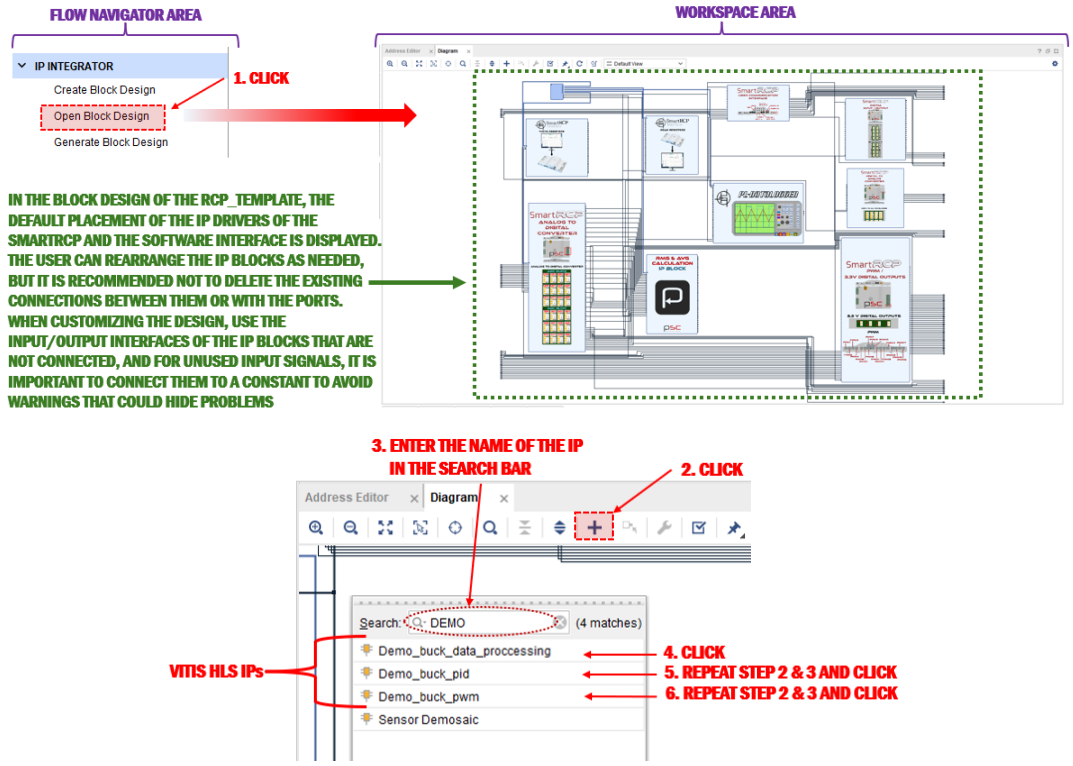


Figure 124: Adding the IPs to the block design, steps 1 to 6: Open Block Design, the add-IP button and the search for the three Vitis HLS IPs.

IP block interconnection

In this step the user makes the connections the project needs, either between the IP blocks designed by the user or with the IP blocks of the SmartRCP. Regardless of the design, some signals, such as the clock and the reset, can be connected automatically. When IP blocks are added, a green message appears at the top of the workspace: check the **Run Connection Automation** options to connect the clock and reset signals of the

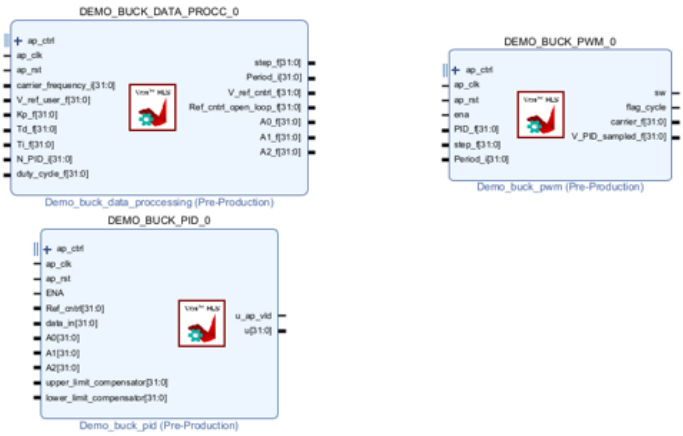


Figure 125: The three user IP blocks after being added to the block design.

new IP blocks automatically (Figure 126). Remember that, if there are several clock sources, you have to select which one each IP is connected to.

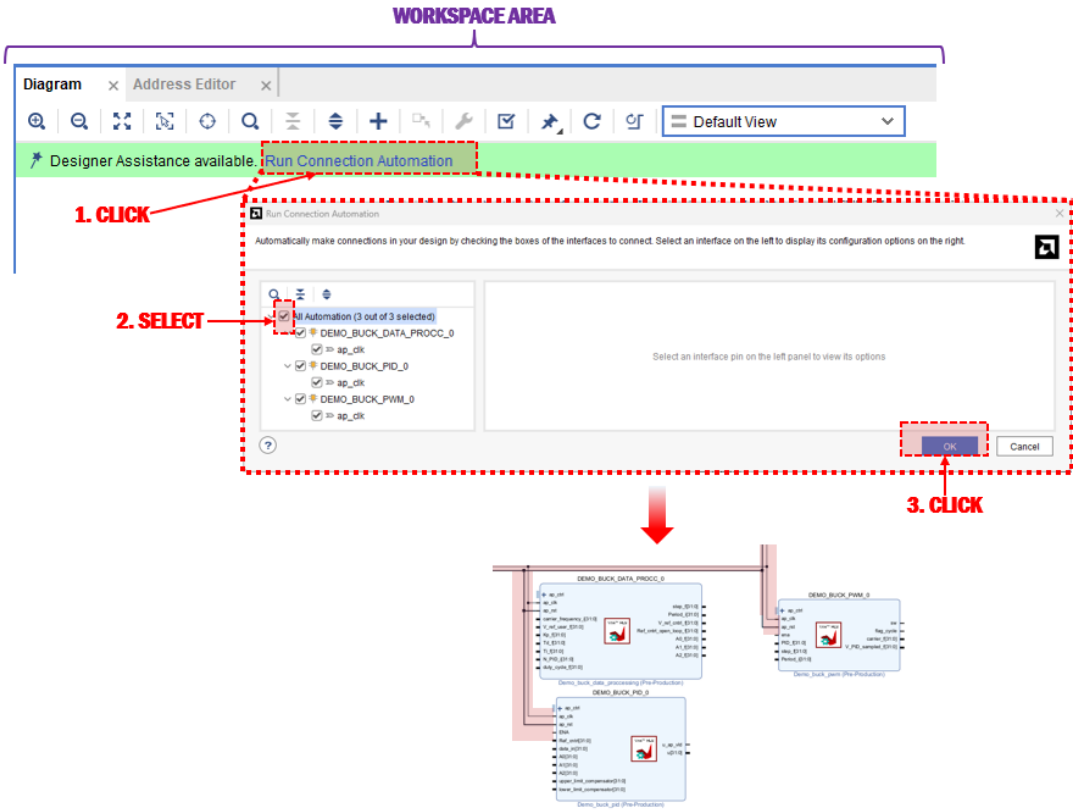


Figure 126: Connection automation, steps 1 to 3: the clock and reset of the three blocks connected automatically.

Data processing block. The first stage of the interconnection is to connect the `DEMO_BUCK_DATA_PROCC` block, which processes the high-level data configured from the SmartRCP desktop application and converts it into the parameters required by the system IPs. To connect two IPs, left-click the input or output signal and, keeping the button pressed, drag the signal to the input or output it has to be connected to. The values configured in the software interface are available in the write registers IP (`OUTPUT_USER_REGISTERS`); make the connections shown in Figure 127.

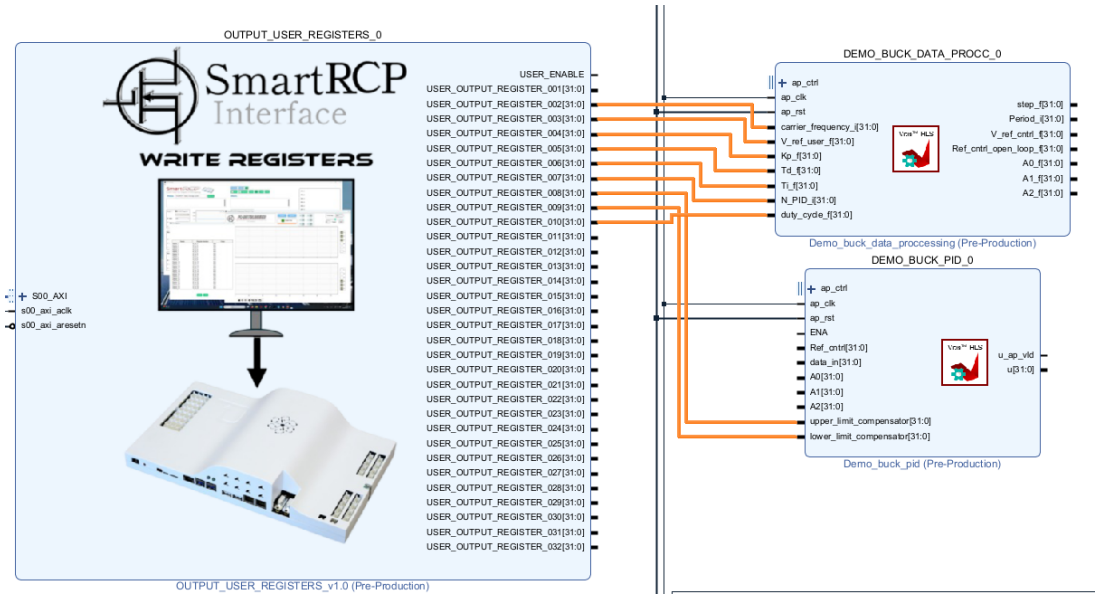


Figure 127: Connections between the write registers driver and the DEMO_BUCK_DATA_PROCC block.

The next step is to interconnect the data processing block with the other IP blocks developed in Vitis HLS, so that the data processed by this block is distributed to the other modules of the system. Make the connections of Figure 128.

Feedback from the ADC. Next, the input of the PID regulator that receives the current value of the average output voltage of the buck converter is connected to the output of the ADC used to sample and control the voltage of the converter. In this configuration the ADC QUICK 1 channel has been selected, using its filtered output, which delivers data in floating-point format (Figure 129).

Multiplexer block MUX_32bit. Since the converter must be able to work in open loop or in closed loop, the signal used as modulator of the PWM pulses (the signal compared with the sawtooth carrier) has to be selectable. To achieve this, a block that selects between two input signals according to the operating mode is designed in VHDL. The design also serves to illustrate the generation of VHDL code in Vivado (Figure 130).

At this point a VHDL template has been created in the file `MUX_32bit.vhd`, ready to hold the digital design of the multiplexer block. Next, the VHDL code that describes the

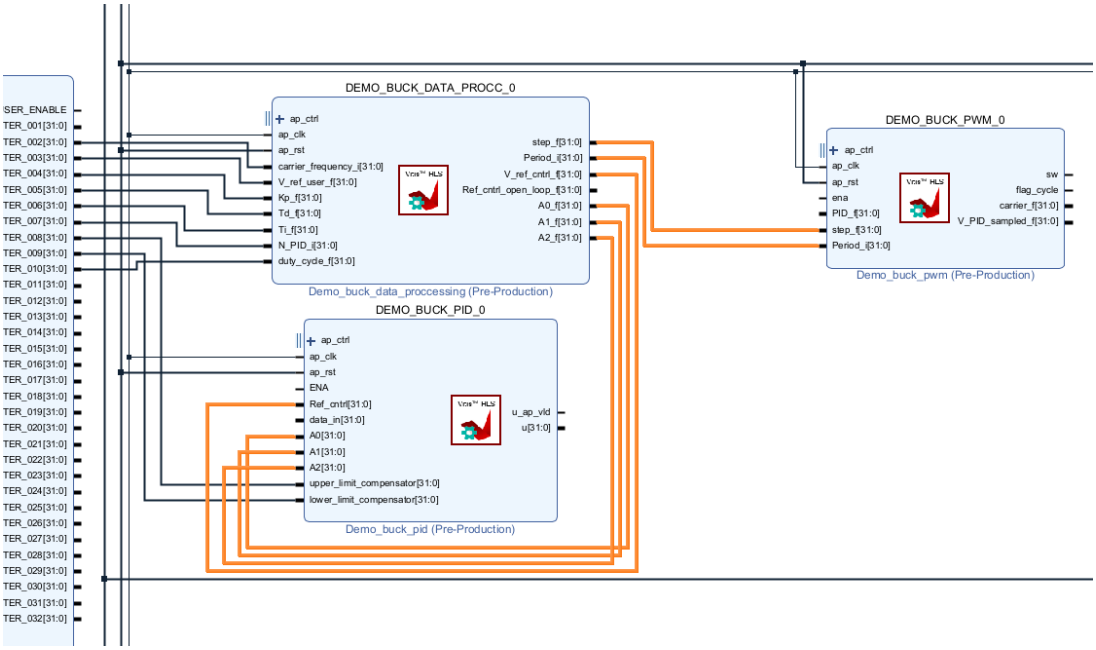


Figure 128: Connections from DEMO_BUCK_DATA_PROCC to the PID and PWM blocks.

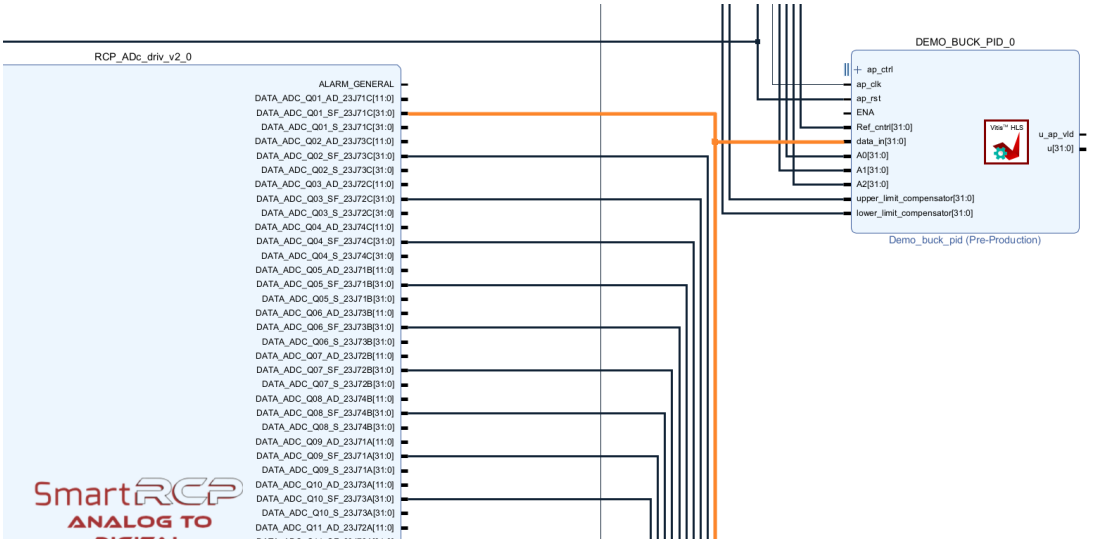


Figure 129: The filtered output of ADC QUICK 1 connected to the `data_in` input of the PID block.

operation of the multiplexer must be written or inserted (Figure 131). All the VHDL files used in this project are located in `SmartRCP_Demo_Buck\IP_VHDL_SRC`. This tutorial does not go into the details of VHDL or Verilog code generation, since that is not its purpose; the abundant documentation available online is recommended for a deeper understanding.

The `MUX_32bit` block has three 32-bit inputs and one 32-bit output, each with a specific function in the control of the system:

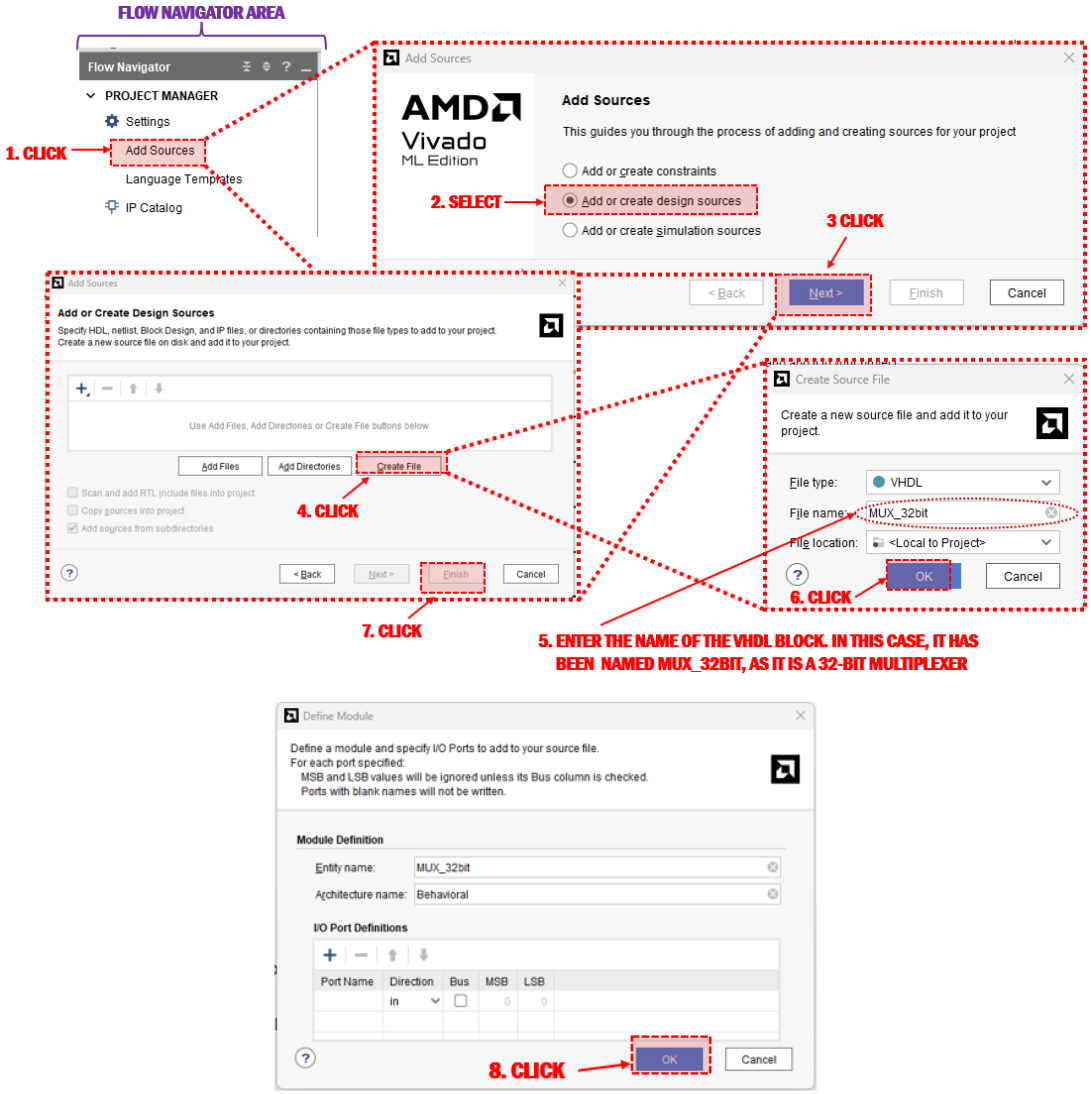


Figure 130: Creating a VHDL source, steps 1 to 8: Add Sources, Create File with the name MUX_32bit, and the Define Module dialog.

- `REF_OPEN_LOOP`: the reference value for the open-loop control mode, in which the user manually defines the duty cycle of the PWM signal.
- `REF_CLOSE_LOOP`: the output of the PID controller, which adjusts the output voltage according to the reference set by the user.
- `SELECTOR`: the control signal that defines which reference is passed to the output. If `SELECTOR` is 0, `REF_OUTPUT` takes the value of `REF_OPEN_LOOP`; for any other value it takes `REF_CLOSE_LOOP`.
- `REF_OUTPUT`: the output, connected as modulator to the PWM block and compared with the sawtooth to generate the PWM pulses.

To add the `MUX_32bit` block to the block design, right-click it in the Sources area and, holding the button, drag it to the work area. Then make the connections of Figure 132.



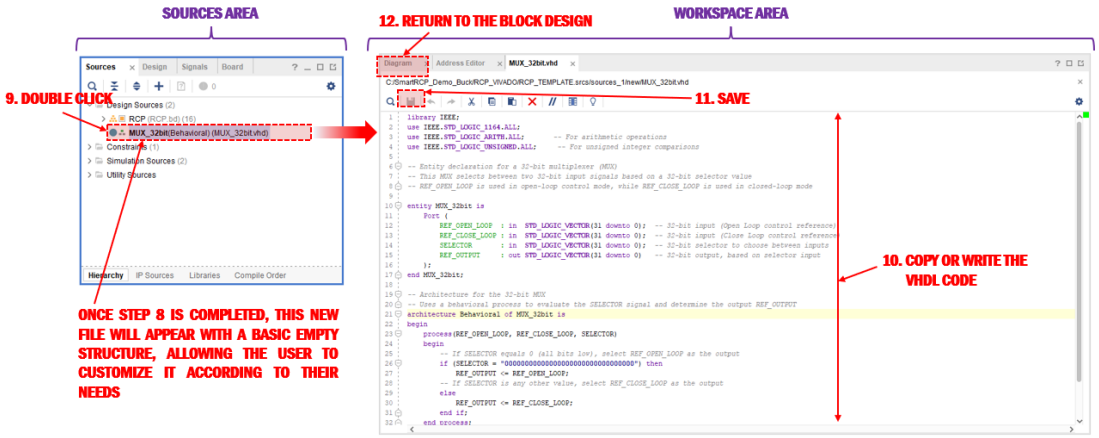


Figure 131: Creating a VHDL source, steps 9 to 12: the empty entity opened from the Sources area, the multiplexer code, save, and return to the block design.

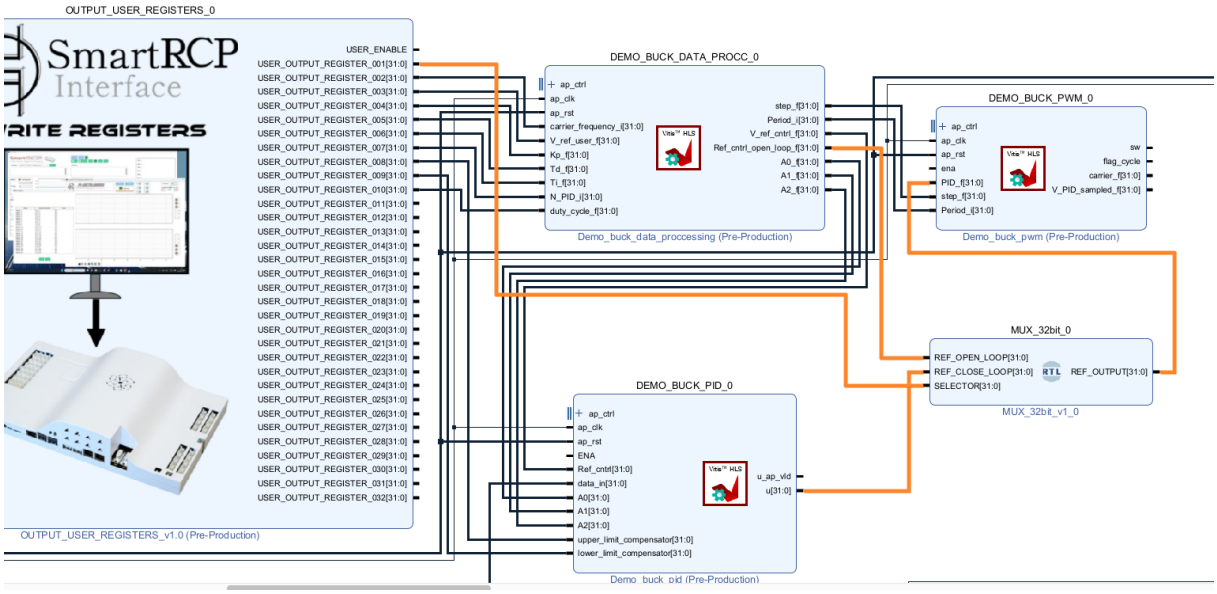


Figure 132: Connections of the MUX_32bit block: open-loop reference from the data processing block, closed-loop reference from the PID, selector from the write registers and output to the PWM block.

Enable logic AND_OP. The design implements an enable logic for the converter that depends on two conditions:

1. **User command from the console.** The user must enable the system from the SmartRCP software console. When the system is enabled, the `USER_ENABLE` signal of the write registers block takes the value 1; otherwise it is 0.
2. **ADC alarm status.** For demonstration purposes an alarm function is used as a protection (for example against overcurrent or overvoltage). The alarms trigger when the ADC values exceed a user-configured threshold for a specified time, both configurable from the software interface. When any alarm is active, the

`ALARM_GENERAL` signal of the ADC driver IP block is set to 1; under normal conditions it is 0.

The final enable signal of the converter requires both conditions: the user has activated the system and no alarm is active. This ensures that the converter is disabled automatically when the user shuts it down from the console or when the configured voltage or current limits are exceeded. To implement this logic a VHDL block performs a logical AND between `USER_ENABLE` and the negated `ALARM_GENERAL`, producing the final enable signal so that the safety conditions are respected at all times. The enable block, named `AND_OP`, was developed with the same steps used for `MUX_32bit`. Make the connections of Figure 133.

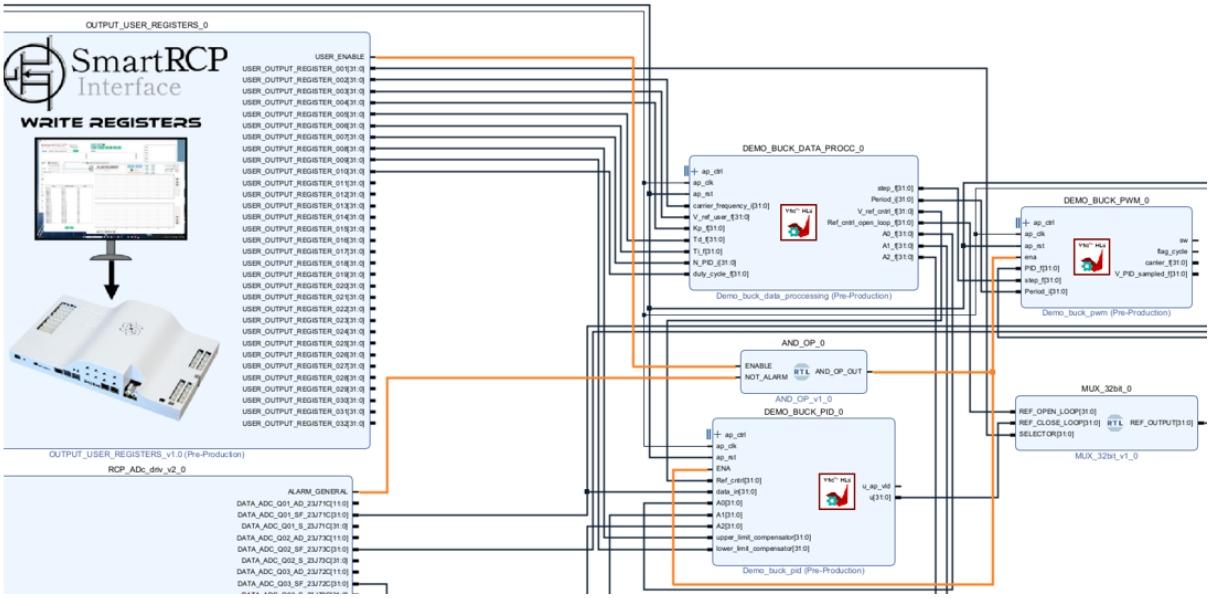


Figure 133: Connections of the AND_OP block: USER_ENABLE from the write registers, ALARM_GENERAL from the ADC driver, output to the enable inputs of the user blocks.

Synchronization of the HLS blocks. As explained in section 7.3, when IP blocks are created from descriptive code, Vitis HLS adds signals to the interface to manage and control the operation of the block; Figure 113 shows them. The key signal that must always be managed is `ap_start`, since it enables the execution of the block. Of the three IP blocks developed for this project, `DEMO_BUCK_DATA_PROCCESSING`, `DEMO_BUCK_PID` and `DEMO_BUCK_PWM`, only the latter two require precise synchronization.

The `DEMO_BUCK_DATA_PROCCESSING` block registers and keeps stable values at its outputs, so its values are updated simultaneously without additional synchronization and its `ap_start` can be connected to a constant 1. However, `DEMO_BUCK_PID` and `DEMO_BUCK_PWM` require coordination so that the control of the converter is accurate and stable.

To handle this synchronization, a dedicated VHDL block generates the `ap_start` signals that activate both blocks at the appropriate time. When designing this code it is essen-

tial to consider the specific latency of each block, obtained during its synthesis in Vitis HLS. The `sync_pulse_generator` block developed for this purpose generates synchronization pulses for the PWM and PID modules at configurable intervals: using internal counters, it triggers `sync_pwm` every given number of clock cycles (`pwm_cycle_count`) and `sync_pid` every given number of carrier periods (`pid_cycle_count`). This logic guarantees that the modules operate in a synchronized manner.

Remaining connections. To finalise the interconnection, the output signals of the PWM block are connected to the PWM output driver of the system. In this design the `PWM18` output has been chosen, so the signal is connected to the corresponding input of the driver and that driver output is enabled.

In addition, the SmartRCP LEDs are assigned to display the system status: LED1 is connected to the enable signal and LED2 to the alarm signal. The control of these LEDs is available in the communications driver `RCP_Comm_Ctrl_V3` (section 5.2).

All the outputs of the `DEMO_BUCK_DATA_PROCC` block are also connected to the inputs of the read registers driver, which allows the results of the calculations of the processing block to be displayed. Since this driver only accepts data in floating-point format, only the outputs whose name ends in `_f` are connected; to display integer values, insert an intermediate block that performs the integer-to-float conversion.

The signals the user may wish to monitor in real time, such as the current and the voltage (filtered and unfiltered), the PID output and the carrier and modulator signals of the PWM block, are connected to the inputs of the PL-DataLogger. This allows a detailed real-time analysis of the system variables and their visualisation in the SmartRCP software interface.

Unused inputs. To avoid warnings during the design flow, every SmartRCP driver and IP input not used in this project is connected to a constant of the same data type. This prevents unnecessary warnings from hiding real design problems. Figure 134 shows the procedure for the unused inputs of the DACs in this demonstrator. Repeat it for every SmartRCP IP input that is not used in the project.

All the important connections (PL-DataLogger, read and write registers) are detailed in the summary tables of section 7.4.5.

7.4.4 Adding the ILA IP block for design debugging

The ILA (Integrated Logic Analyzer) block is one of the most useful resources for debugging digital designs offered by Vivado. It fulfils a function similar to a waveform viewer, showing the time evolution of internal signals of the FPGA. Its functionality is



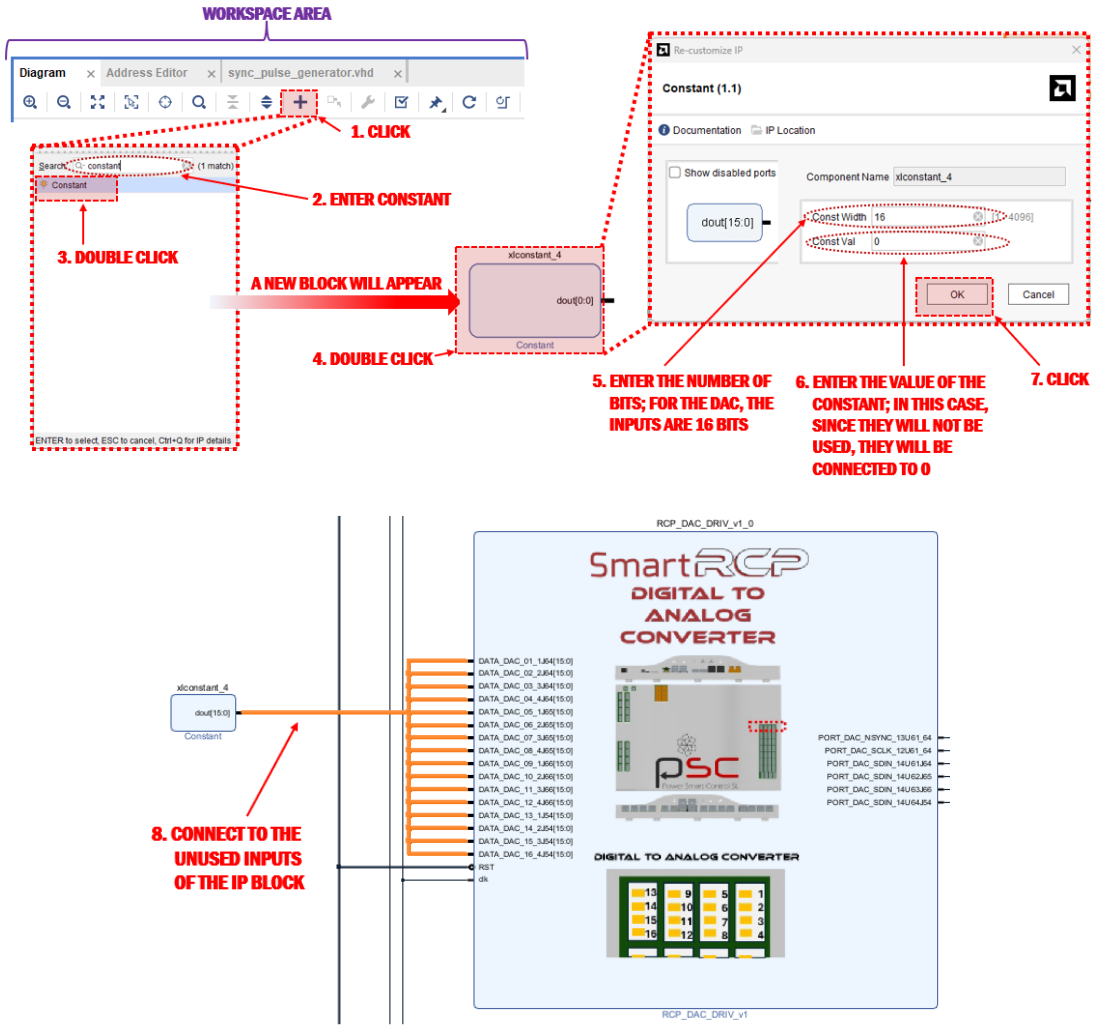


Figure 134: Connecting a constant to unused inputs, steps 1 to 8: adding a Constant block, setting its width (16 bits for the DAC) and value (0), and connecting it to the unused DAC inputs.

comparable to an oscilloscope: it offers a number of user-defined channels that can be connected to internal signals of the design.

The main utility of the ILA lies in verifying and debugging the operation of digital designs, which is crucial when checking the generation of signals, the synchronism between them, the activation of IP blocks and the consistency of the results. Its purpose, however, is debugging only: the ILA is not designed to act as a real oscilloscope for waveform display. It has certain limitations:

- It does not support the interpretation of floating-point numbers.
- Its communication is exclusively through JTAG, which limits the speed and forces the user to be physically close to the SmartRCP device.
- Proper configuration of the ILA requires advanced technical knowledge, as it includes several functionalities that must be adjusted for each project.

For users who need a true real-time oscilloscope, the PL-DataLogger block is recommended. This block, included in the SmartRCP template, allows signal monitoring over Ethernet with a configurable interface, similar to a traditional oscilloscope.

In this tutorial the ILA block is added to the design as an example, so that users understand the configuration process and can incorporate it into their projects if they wish. Users interested in the capabilities of this tool should consult the specific Xilinx documentation and tutorials available online. Follow the steps of Figure 135 to add the ILA block to the buck converter project.

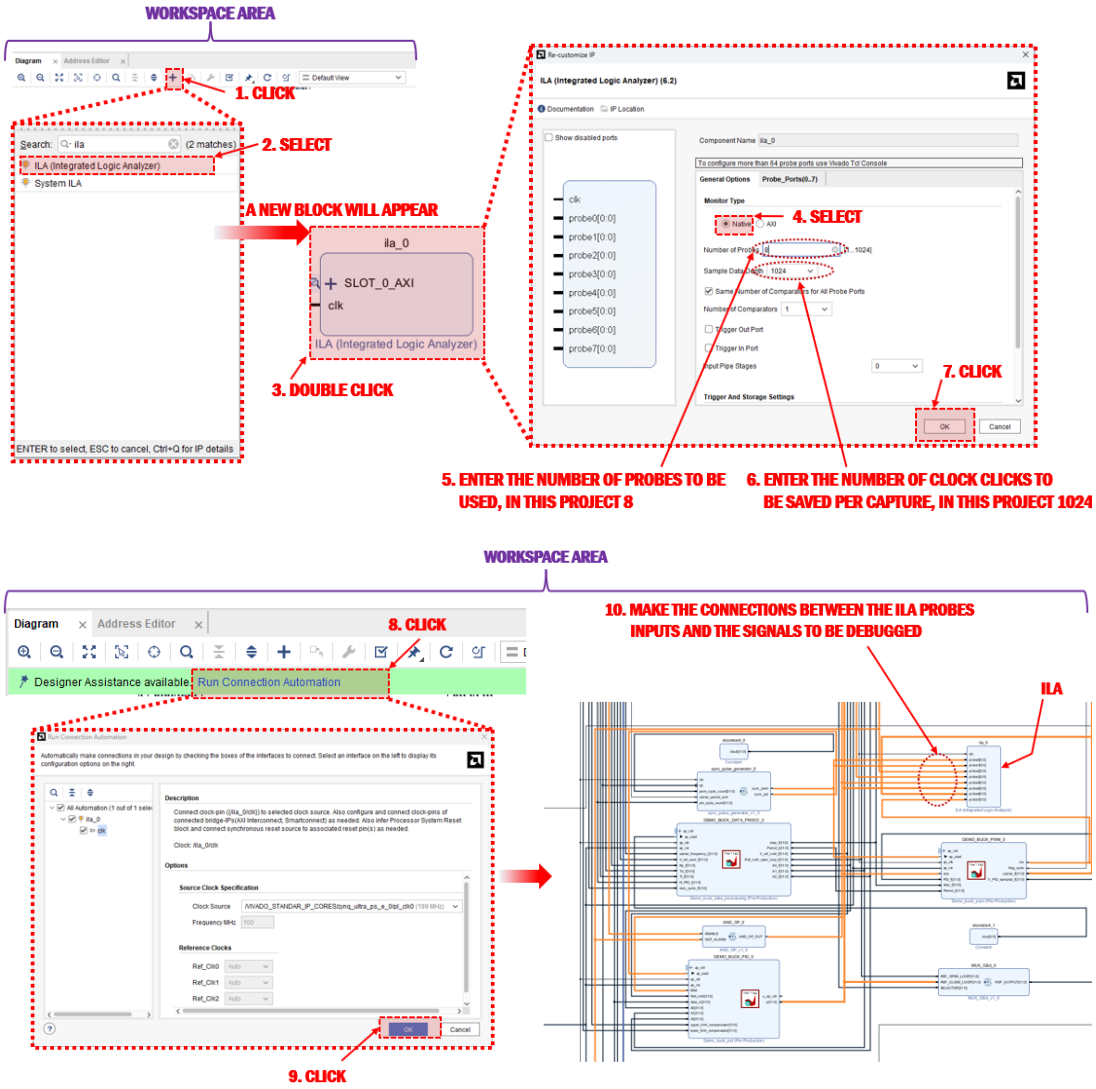


Figure 135: Adding the ILA, steps 1 to 10: search and configuration of the ILA (8 probes, 1024 samples), connection automation of its clock and connection of the probes to the signals to be debugged.

The consumption of memory blocks by the ILA is proportional to the number of declared probes, to the size of each one (the more bits in the signal connected to a probe, the more URAM blocks it consumes) and to the number of samples stored per capture. Adjust these

parameters carefully: an excessive use of resources can exceed the capacity of the K26 SoC and cause failures in the synthesis and implementation.

7.4.5 Summary and final user considerations

First, in order to organise the block diagram and keep the part of the design created by the user clearly identified within the RCP_TEMPLATE, the steps to generate a sub-design that contains the whole user design are shown. The same method allows additional sub-assemblies, and sub-assemblies within sub-assemblies, so that the user can arrange the screen in the most convenient way for the project. Follow the steps of Figure 136 to create a subset that contains all the blocks designed by the user.

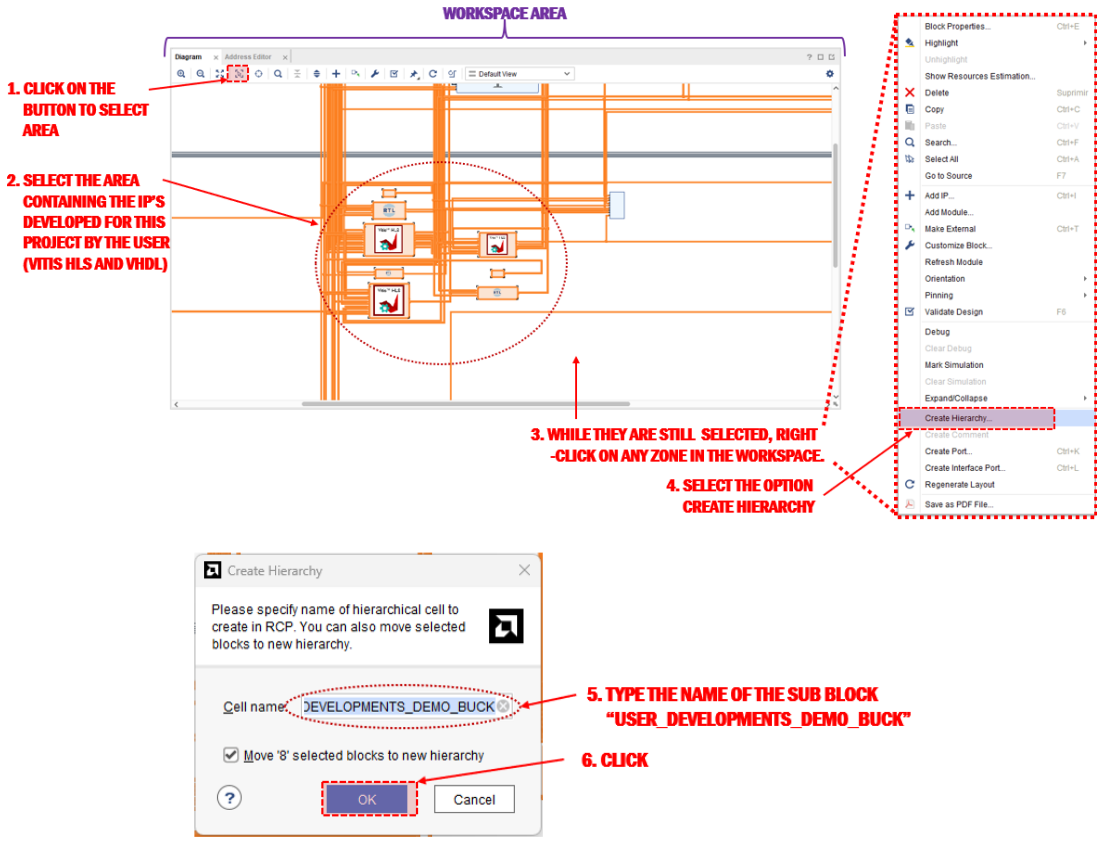


Figure 136: Creating a hierarchy, steps 1 to 6: area selection of the user blocks, Create Hierarchy in the context menu and the name of the new cell.

The final block diagram looks like Figure 137. To access the contents of the subset, the user can either expand it in the same workspace or open a new workspace with only the blocks of that subset (Figure 138).

Secondly, the main connections are summarised in Tables 30 to 33, highlighting the configurations relevant for the SmartRCP software interface: PL-DataLogger channels, read and write registers, and ILA probes. This information is essential to customize the software interface for the project.

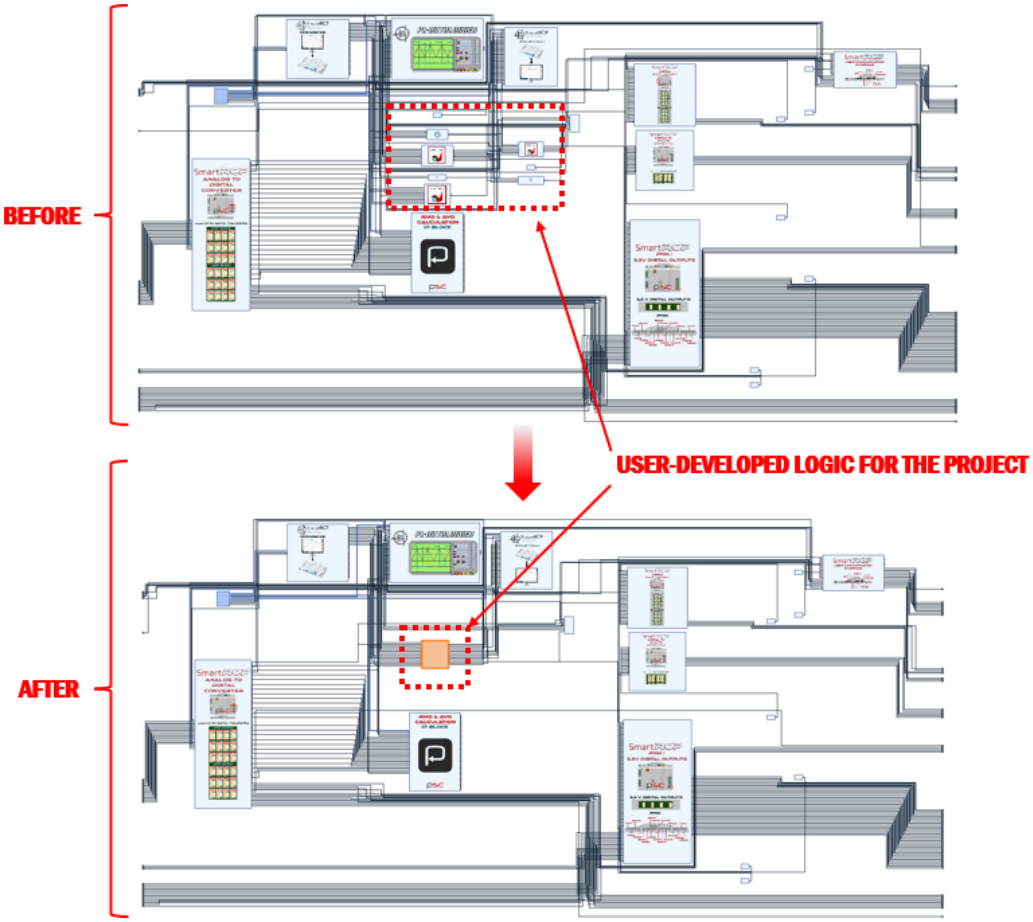


Figure 137: The block design before and after grouping the user logic into the USER_DEVELOPMENTS_DEMO_BUCK hierarchy.

Remember that the whole procedure, with additional steps and settings, is available in the `SmartRCP_Demo_Buck` archive, which holds the finished project. If the user prefers to perform each step from scratch, the finished project can be used as a reference alongside this tutorial. The tutorial is designed as a generic, sequential guide to the minimum steps needed to customize the RCP_TEMPLATE to the specific needs of any SmartRCP project using the RCP_TEMPLATE resources; its purpose is not to be an exhaustive guide to this particular project, but a structured basis for the development flow of future SmartRCP projects.

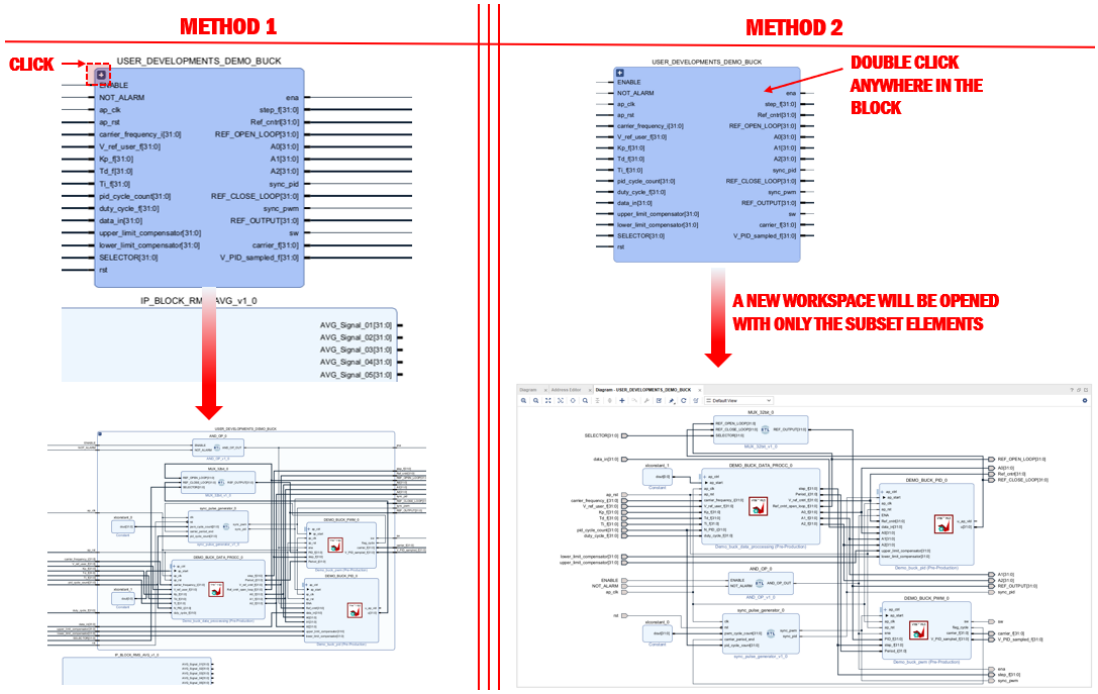


Figure 138: The two ways of viewing the contents of a hierarchy: expanded in place, or opened in a new workspace.

PL-DataLogger channel	Signal	Source IP block
IN_CH_01	Filtered buck converter output voltage	SmartRCP ADC driver, ADC QUICK 1
IN_CH_02	Unfiltered buck converter output voltage	SmartRCP ADC driver, ADC QUICK 1
IN_CH_03	Filtered current measurement of the buck converter coil	SmartRCP ADC driver, ADC QUICK 2
IN_CH_04	Unfiltered current measurement of the buck converter coil	SmartRCP ADC driver, ADC QUICK 2
IN_CH_05	PWM modulating signal	MUX_32bit
IN_CH_06	PWM carrier signal (sawtooth)	DEMO_BUCK_PWM
IN_CH_07	Modulating signal currently sampled by the PWM comparator	DEMO_BUCK_PWM
IN_CH_08	Control signal, PID output (closed-loop modulator)	DEMO_BUCK_PID
IN_CH_09	Control signal (open-loop modulator)	DEMO_BUCK_DATA_PROCC
IN_CH_10 to IN_CH_32	Manually controlled signal from the SmartRCP software interface application	Write registers, USER_OUTPUT_REGISTER_011

Table 30: Signals accessible from the PL-DataLogger.

Read registers input	Signal	Source IP block
User input register 001	Carrier signal voltage steps (sawtooth)	DEMO_BUCK_DATA_PROCC
User input register 002	Closed-loop voltage reference	DEMO_BUCK_DATA_PROCC
User input register 003	Open-loop modulating signal	DEMO_BUCK_DATA_PROCC
User input register 004	Result of the calculation of the PID coefficient A0	DEMO_BUCK_DATA_PROCC
User input register 005	Result of the calculation of the PID coefficient A1	DEMO_BUCK_DATA_PROCC
User input register 006	Result of the calculation of the PID coefficient A2	DEMO_BUCK_DATA_PROCC
User input registers 007 to 032	Manually controlled signal from the SmartRCP software interface application	Write registers, USER_OUTPUT_REGISTER_011

Table 31: Signals accessible from the read registers.

ILA probe	Signal	Source IP block
PROBE0	PWM ap_start pulses	sync_pulse_generator
PROBE1	PID ap_start pulses	sync_pulse_generator
PROBE2	General alarm signal	SmartRCP ADC driver
PROBE3	User enable signal	Write registers
PROBE4	Enable signal of the IP blocks developed for this project	AND_OP
PROBE5	Control signal, PID output (closed-loop modulator)	DEMO_BUCK_PID
PROBE6	PWM carrier signal (sawtooth)	DEMO_BUCK_PWM
PROBE7	PWM signal	DEMO_BUCK_PWM

Table 32: Signals accessible from the ILA.



Write registers output signal	**Configurable parameter**	**IP block it connects to**
User output register 001	Mode selector: 1 = closed loop, 0 = open loop	`MUX_32bit`
User output register 002	Switching frequency [kHz]	`DEMO_BUCK_DATA_PROCC`
User output register 003	Closed-loop reference voltage [V]	`DEMO_BUCK_DATA_PROCC`
User output register 004	PID K_p	`DEMO_BUCK_DATA_PROCC`
User output register 005	PID T_d	`DEMO_BUCK_DATA_PROCC`
User output register 006	PID T_i	`DEMO_BUCK_DATA_PROCC`
User output register 007	PID update, in carrier periods	`DEMO_BUCK_DATA_PROCC`
User output register 008	`UPPER_LIMIT_COMPENSATOR`	`DEMO_BUCK_PID`
User output register 009	`LOWER_LIMIT_COMPENSATOR`	`DEMO_BUCK_PID`
User output register 010	Duty cycle in open-loop mode [%]	`DEMO_BUCK_DATA_PROCC`
User output register 011	Auxiliary, to set values on unused probes	PL-DataLogger and read registers

Table 33: Parameters configurable from the SmartRCP software application through the write registers.



7.5 Generating the bitstream: synthesis and implementation

In this step the files that convert the block diagram into a bitstream are generated, so that the FPGA can be configured with the developed design. The process, which includes the synthesis and the implementation of the design in Vivado, transforms the block diagram into a binary file ready to be loaded into the hardware. The synthesis analyses the design, optimises resources and creates a low-level representation in logic gates ready to be implemented in the FPGA. The implementation then places and connects these elements in the FPGA architecture, respecting the timing, resource and topology constraints of the design. Start with the steps of Figure 139.

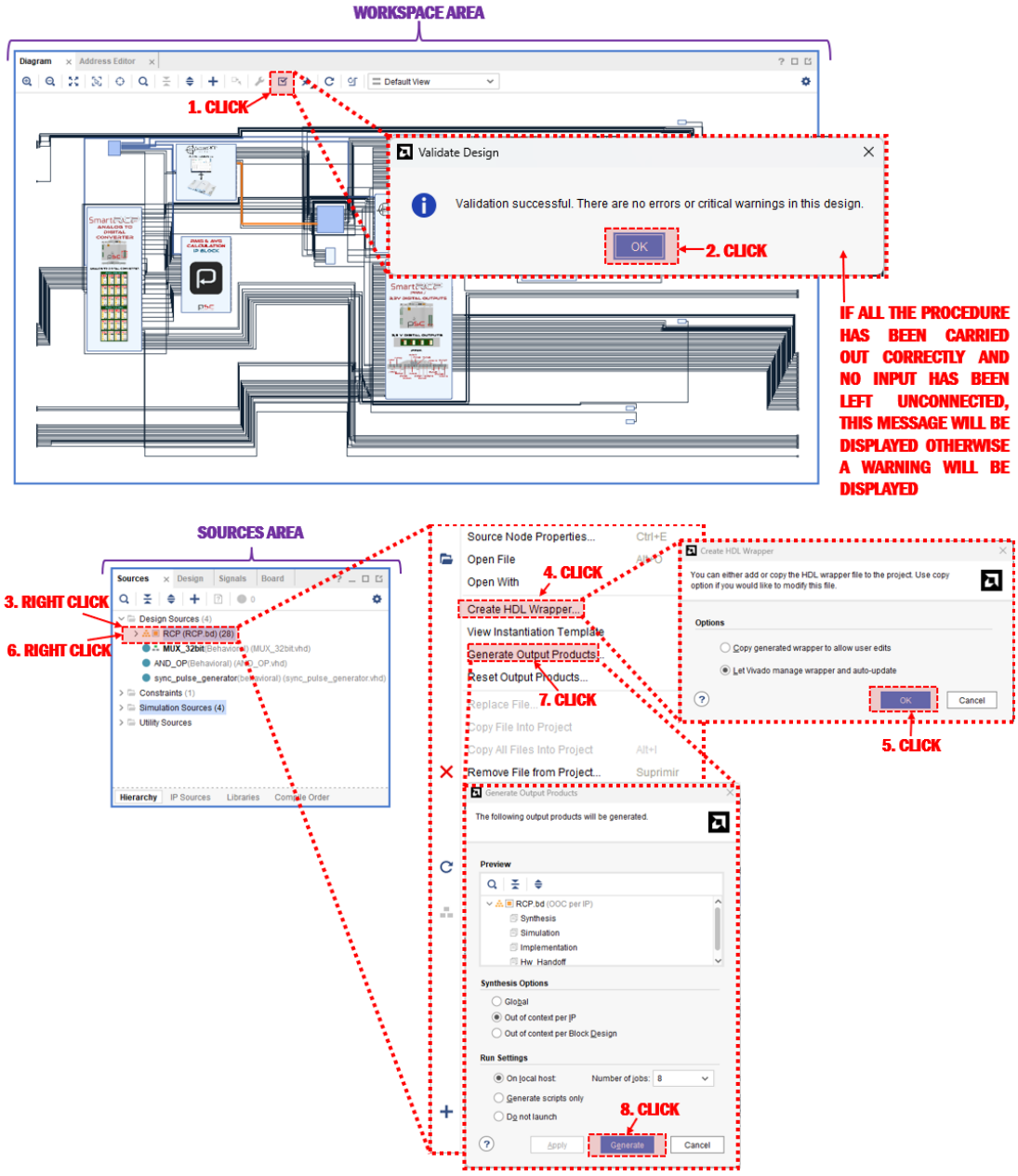


Figure 139: Bitstream generation, steps 1 to 8: Validate Design, Create HDL Wrapper and Generate Output Products from the Sources area.

The process started in step 8 may take several minutes, depending on the processing power of the computer (Figure 140).

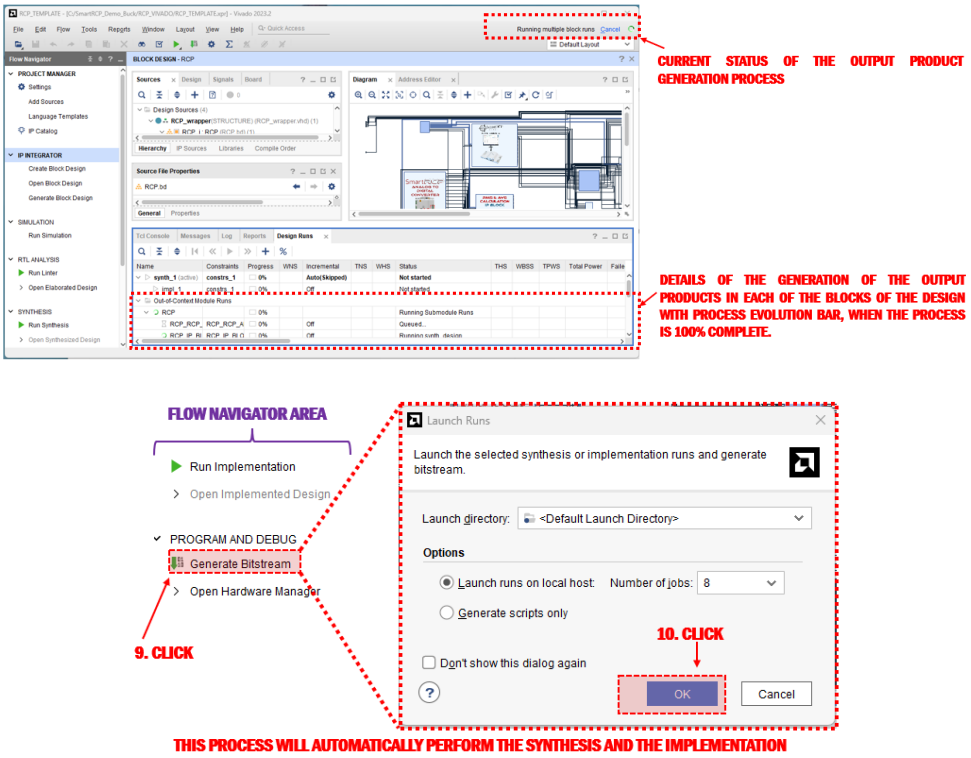


Figure 140: Bitstream generation, steps 9 and 10: progress of the output product generation, then Generate Bitstream and the Launch Runs dialog.

The process started in step 10 may also take several minutes (Figure 141). It produces the bitstream file, the final configuration to be loaded into the FPGA (Figure 142).

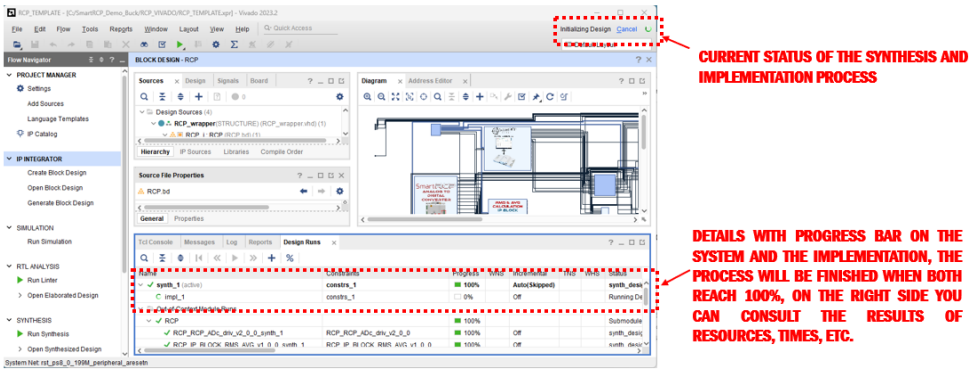


Figure 141: Bitstream generation: progress of the synthesis and implementation runs in the Design Runs tab.

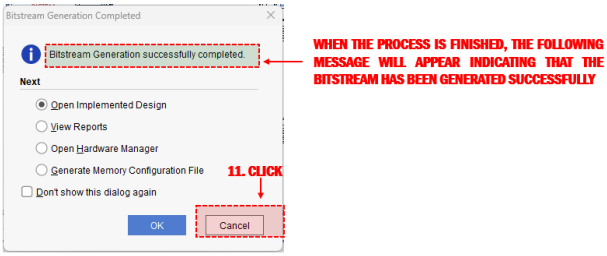


Figure 142: Bitstream generation, step 11: the message that confirms that the bitstream has been generated.

7.6 RCP_TEMPLATE customization: configurations in Vitis IDE

This section details the procedure to generate a project in Vitis IDE, needed to develop the software applications that run on the microprocessors of the K26 SoM. This project is essential to program the SmartRCP, both when generating the boot files for the microSD card and when loading the software directly on the board over JTAG. These configurations also allow the full integration between the software and the FPGA hardware. The steps are:

1. Export the files generated in the Vivado design flow of the previous section.
2. Create a project in Vitis IDE as the basis for the next steps.
3. Generate a generic "Hello World" application to show the minimum steps needed to create applications for one of the microprocessors available in the SmartRCP.
4. Review the considerations the user should take into account when developing applications that run in parallel with the software interface or on several cores.

7.6.1 Exporting files from Vivado to Vitis IDE

The first step is to export the hardware configuration file that results from the Vivado project. It is essential that the exported file includes the bitstream, since it holds the complete configuration to program the FPGA: make sure the corresponding option is selected in the export. In Vivado follow the steps of Figure 143.

The `BUCK_DEMO_HW.xsa` file generated with these steps is available in the `BUCK_DEMO_HW_VIVADO_EXPORT` folder of the archive that holds the final results of the buck converter demonstrator project.

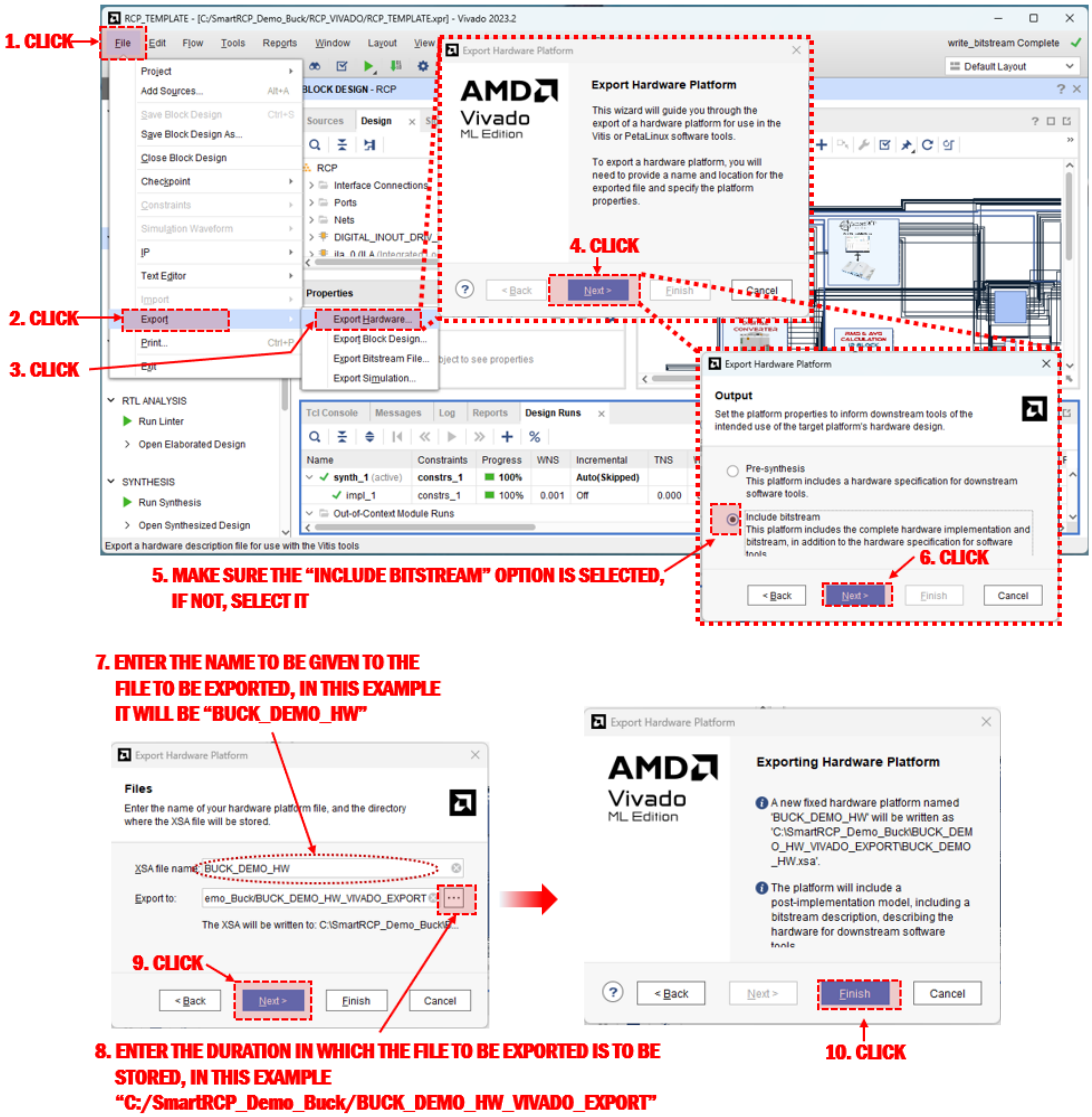


Figure 143: Export Hardware, steps 1 to 10: File, Export Hardware, Include bitstream, and the name and folder of the `.xsa` file.

7.6.2 Creating a new project in Vitis IDE

At this stage the hardware configuration file exported from Vivado is imported, and from it a project is created in Vitis IDE, customized for the SmartRCP. First, create a folder named `DEMO_BUCK_Vitis_IDE` where the Vitis IDE project will be saved, then follow the steps of Figures 144 to 146.

Step 11 assigns the application to core A53_0 because the Interface release of this demonstrator (2024.2) runs the interface application on R5_0. With a template that reserves A53_0 for the interface application (see the note in section 7.6.4), select A53_1 instead; nothing else in the procedure changes.

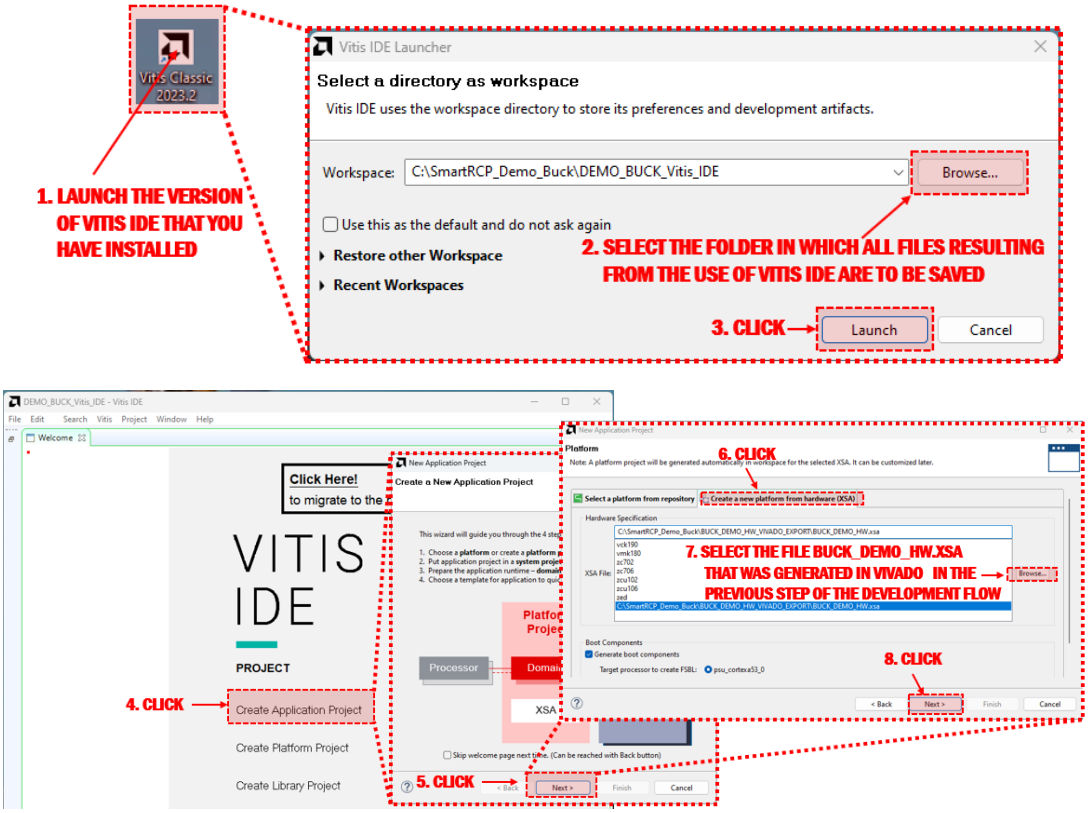


Figure 144: Vitis IDE project, steps 1 to 8: launching Vitis IDE with the new workspace, Create Application Project and the platform created from the exported `.xsa`.

7.6.3 Generating a basic "Hello World" application

This section gives the user the minimum knowledge to develop a bare-metal application that runs on one of the microprocessors of the SmartRCP, in this case the Cortex-A53_0 core. The application is based on the Vitis IDE "Hello World" example and requires basic knowledge of C programming.

In the previous step a Vitis project was created and associated with this first application, which runs on the A53_0 core. Starting from the "Hello World" example, some minimal modifications are made so that the user understands the file structure, identifies the source file of the code and learns the steps needed to adapt any example. The tutorial covers only the essentials; for a deeper understanding consult the Xilinx documentation available online. The example code has these characteristics:

- The SmartRCP periodically writes to the serial port the message `SmartRCP CORE A9_0 ACTIVE = cnt`, where `cnt` is a counter that increments from 1 to 100 and then restarts.
- The counter increments by 1 with each write to the serial port.
- The message is written every 2 seconds.

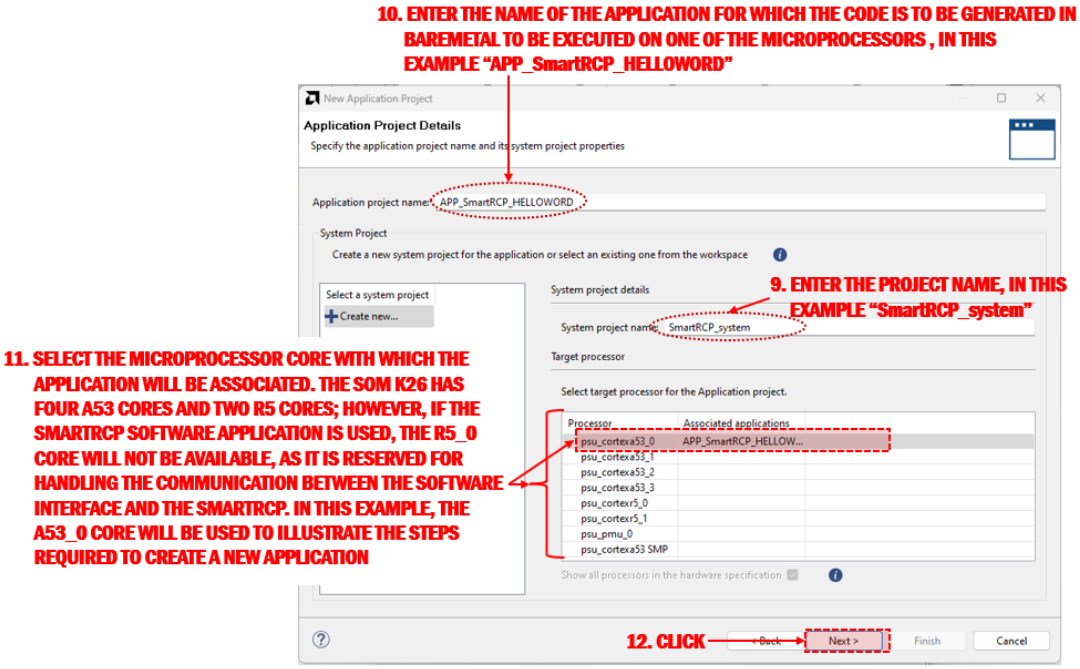


Figure 145: Vitis IDE project, steps 9 to 12: application and system project names and the selection of the target core.

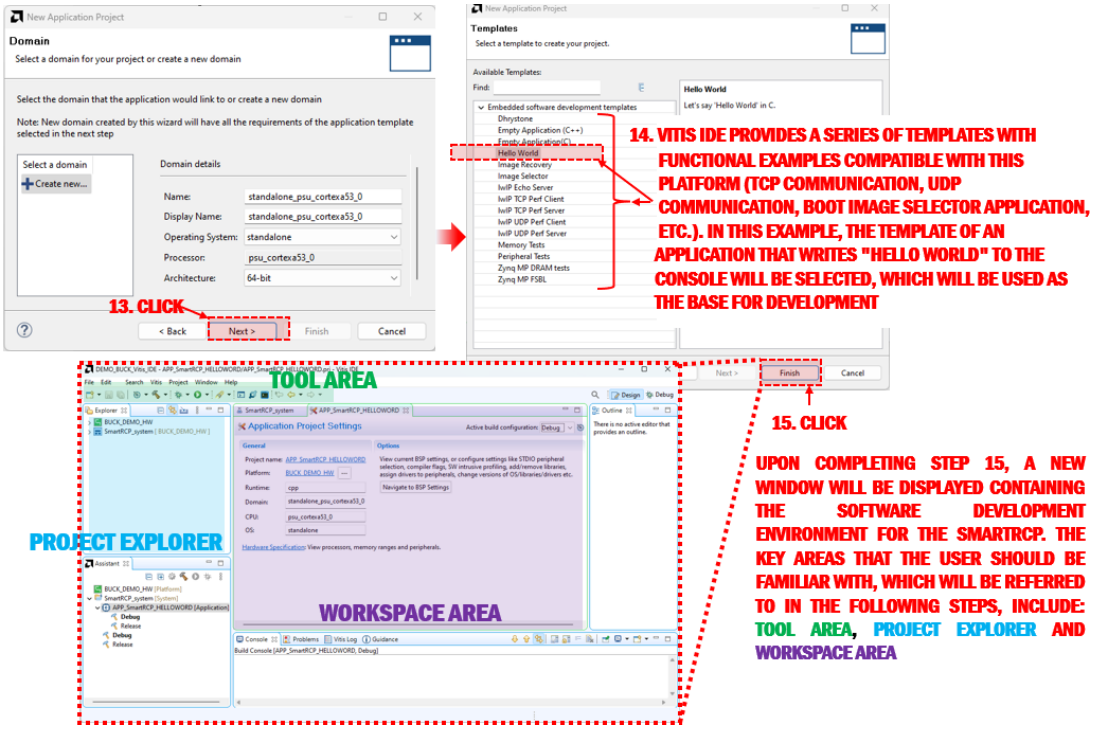


Figure 146: Vitis IDE project, steps 13 to 15: domain, the Hello World template and the resulting development environment.

Since the purpose of the tutorial is not to detail the code, it is not reproduced here; the complete code is available in the archive of the finished project provided to SmartRCP users. Follow the steps of Figures 147 and 148.

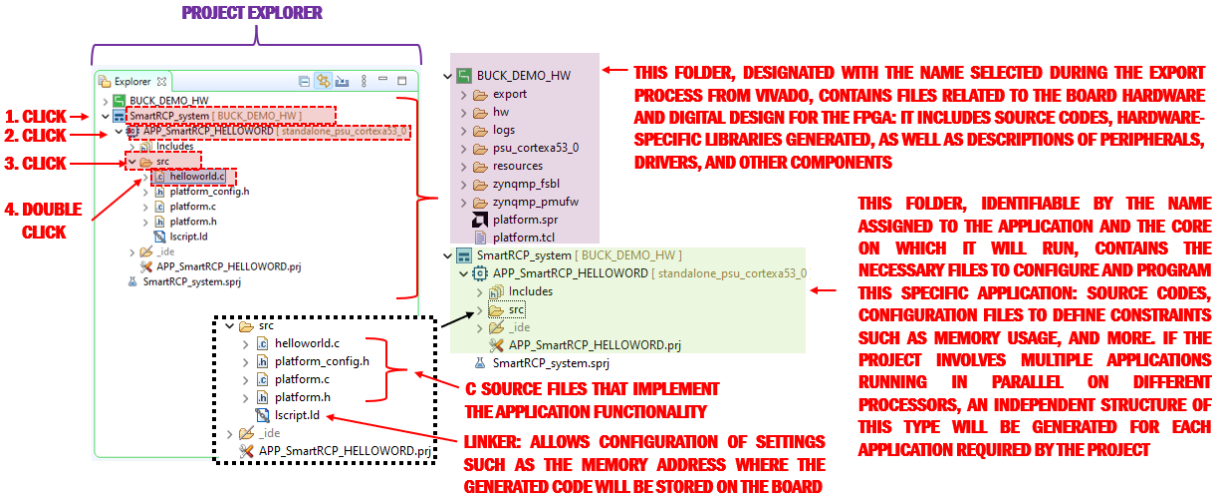


Figure 147: Hello World, steps 1 to 4: the platform and application folders in the Project Explorer, and the `src` folder with the C sources and the linker script.

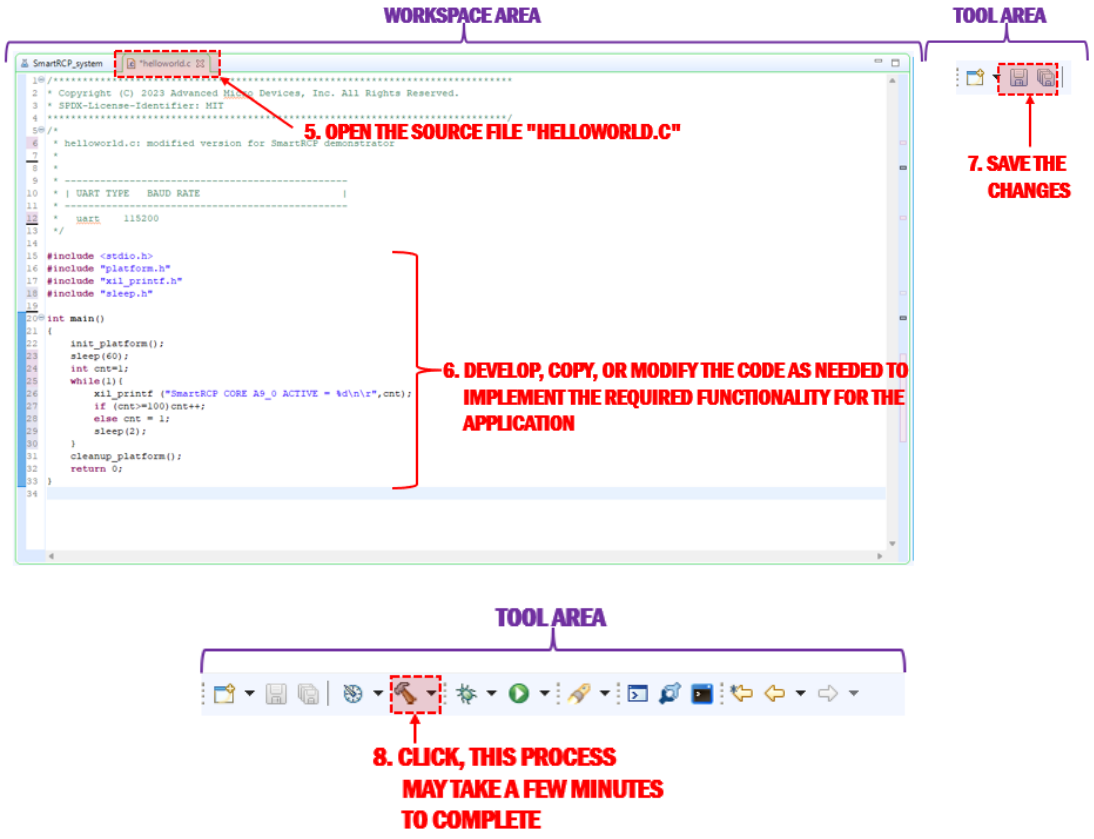


Figure 148: Hello World, steps 5 to 8: the modified `helloworld.c`, save, and the build button of the tool area.

7.6.4 Considerations for parallel applications on multiple cores

The SmartRCP incorporates a K26 SoC with two types of processing core: a quad-core Arm Cortex-A53 and a dual-core Arm Cortex-R5F. Each type is designed for specific

applications, and its recommended use depends on the processing and control requirements of the application.

Differences between the Cortex-A53 and Cortex-R5F cores

1. **Arm Cortex-A53**
   - **Purpose:** high-performance processing and general applications that require higher computational capacity.
   - **Architecture:** 64-bit processor, suited to advanced operations and high performance in multitasking environments.
   - **Recommended applications:** complex, high-level applications such as signal processing, closed-loop control of complex systems and execution of user interfaces. It is the preferred core for processing-intensive applications or those that require a more complex software interface.
2. **Arm Cortex-R5F**
   - **Purpose:** real-time, low-latency applications, prioritising reliability and fast response.
   - **Architecture:** 32-bit processor designed for environments where accuracy and speed of response are critical.
   - **Recommended applications:** real-time control tasks, such as communications management and monitoring of critical systems. It is ideal for direct management of hardware and for the synchronization of signals or peripherals.

Resource allocation in the buck converter project

In the buck converter demonstrator two cores are used, to illustrate a multiprocess application design (Figure 149):

- **R5_0:** runs the application that manages the communication between the SmartRCP and the desktop software interface. The application is delivered precompiled as `APP_RCP_USER_INTERFACE.elf`, configured exclusively to run on the R5_0 core. To take advantage of the RCP_TEMPLATE and the desktop application, the user must reserve the R5_0 core for this task.
- **A53_0:** used as the example of multiprocess development, running the Hello World application developed in the previous section.

The core assignment above is the one of the demonstrator files (Interface 2024.2). In the RCP_TEMPLATE release that accompanies the current *Getting started* guide, the interface application runs on core A53_0, which is then the reserved core, and the delivered files are named accordingly (`ADD_VITIS_A53_0`, `APP_RCP_USER_INTERFACE_A53_0.elf`). Check the folder names of your template and select the core that matches them; the procedure is otherwise identical.

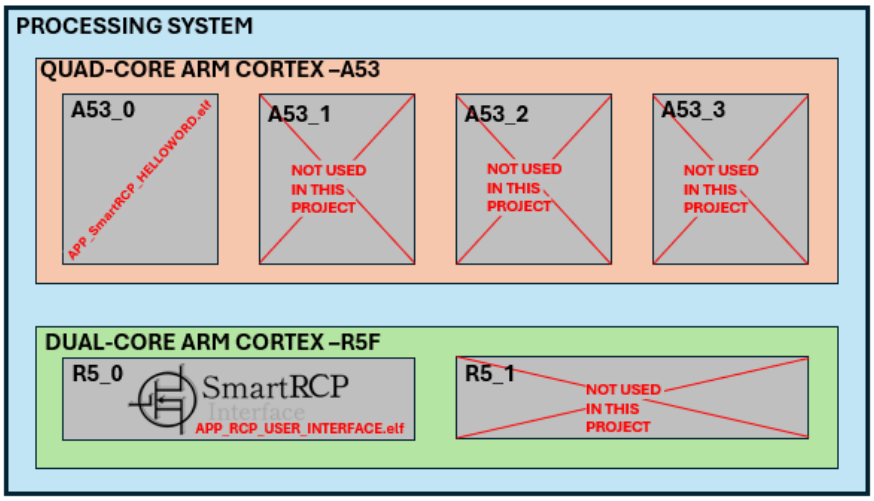


Figure 149: Core allocation of the demonstrator: the Hello World application on A53_0 and the interface application on R5_0.

Memory considerations for parallel applications

A fundamental consideration in parallel application development is the allocation of memory. Although each core operates independently, the memory where the code is loaded is shared, so each core must be given a unique memory region. If two applications share the same region, code can be overwritten, causing one of them, or both, to fail. Table 34 lists the memory address ranges available in the K26 SoC.

For the application that manages the communication with the software interface these memory settings are already established, and other applications must not interfere with them. Figure 150 details the memory locations used by `APP_RCP_USER_INTERFACE.elf`, to be taken into account when allocating memory to the user's own applications running in parallel on other microprocessors.

Example of memory configuration for parallel applications

The following steps make the execution of the Hello World process compatible with the SmartRCP software interface application. Although this is only an example, the same procedure applies to any multiprocess configuration. Follow the steps of Figure 151.

Address range	Memory
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 FFFF_FFFF	OCM

Table 34: Memory address ranges of the Zynq UltraScale+ MPSoC available to the applications.

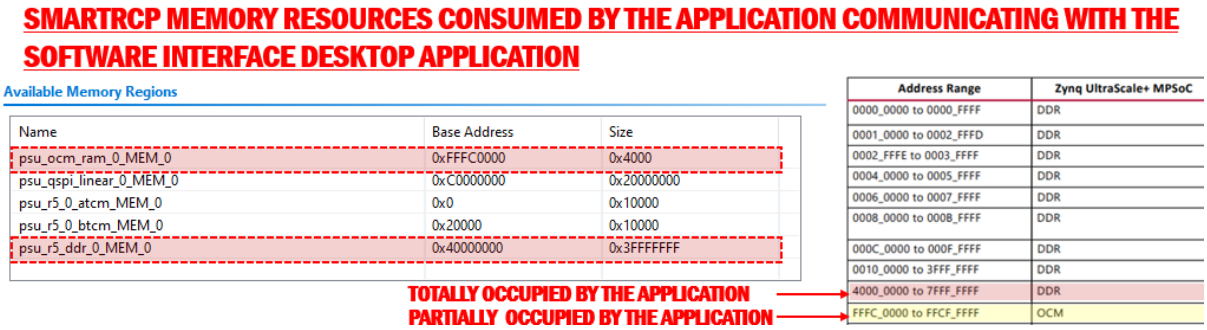


Figure 150: Memory regions consumed by the application that communicates with the software interface, and their place in the address map.

The memory constraints tell the compiler from which address the application code is stored in the shared memory of the system:

- **Base Address:** the lowest memory address at which the application code can begin to be stored.
- **Size:** the number of memory addresses available to the application, starting from the base address.

The complexity of the application directly influences the memory required (Size). The user must adjust this parameter to the needs of each application: if the assigned size is smaller than required, an error is generated during the build (step 5).

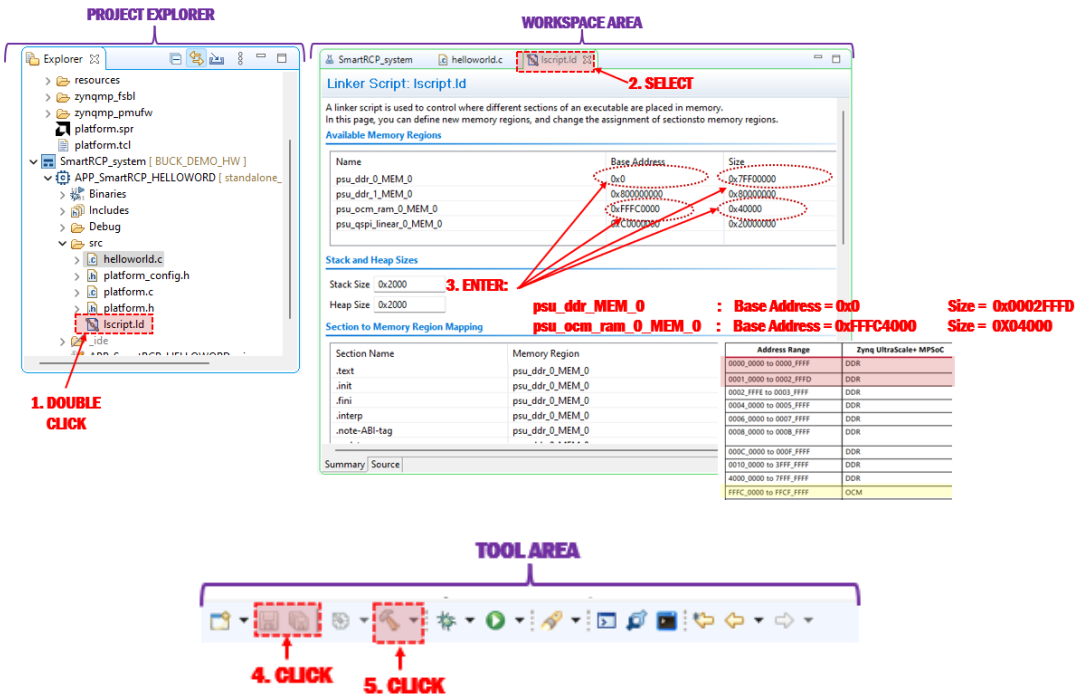


Figure 151: Memory configuration, steps 1 to 5: the linker script `lscript.ld`, the base address and size of the DDR and OCM regions, save and build.

7.7 Configuring the SmartRCP device: microSD and JTAG

This section presents the steps to configure the SmartRCP device by two methods: through the JTAG connection, and by generating a boot file for the microSD card.

Whenever the SmartRCP software interface application is used, a microSD card containing the network configuration file of the device (`PSC_TCP.txt`) is required. The file is available inside the RCP_TEMPLATE archive, at `RCP_TEMPLATE\RCP_VITIS_APP_USER_INTERFACE\ADD_uSD_CARD`.

7.7.1 Configuring the SmartRCP device via JTAG connection

This procedure needs a USB cable with a Micro-B connector for the SmartRCP side (connector J6), which links the user's computer directly with the SmartRCP device. This cable is the minimum requirement to use the JTAG programming method on any board programmed from Vitis IDE.

In this demonstration project an Ethernet cable is also required for the communication between the SmartRCP and the user's computer, as well as a microSD card. The card holds the configuration parameters that define the network connection preferences: whether the connection is established automatically by DHCP negotiation with

the router, or manually, with a specific IP address assigned by the user to the device. Before starting the configuration steps, make the connections of Figure 152.

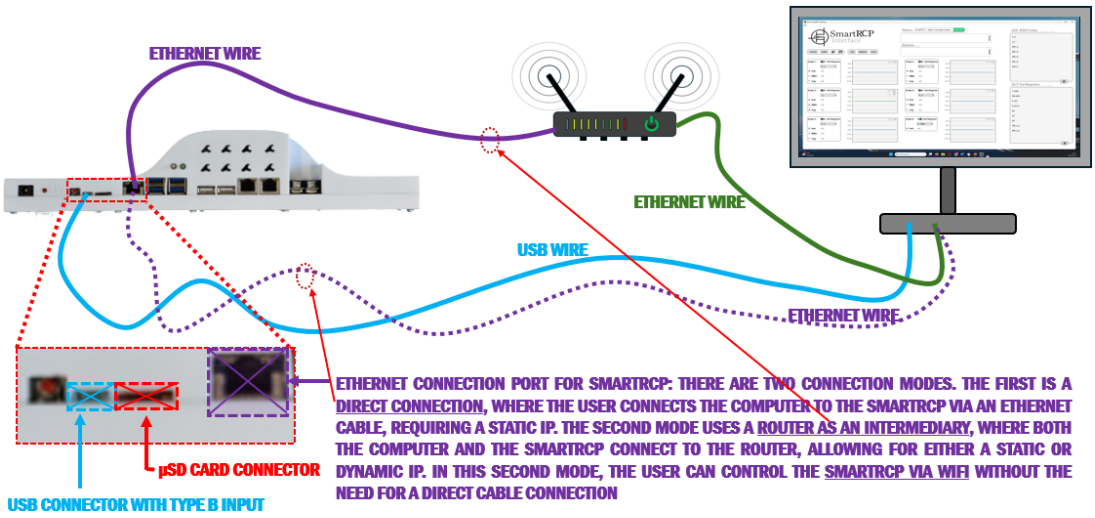


Figure 152: Connections for JTAG programming: USB cable to the computer, microSD card inserted, and the Ethernet connection either direct or through a router.

Proceed with the following steps:

1. Turn on the SmartRCP.
2. Make the connections described in Figure 152. To load the design on the board, only the USB connection between the developer's computer and the SmartRCP is strictly needed.

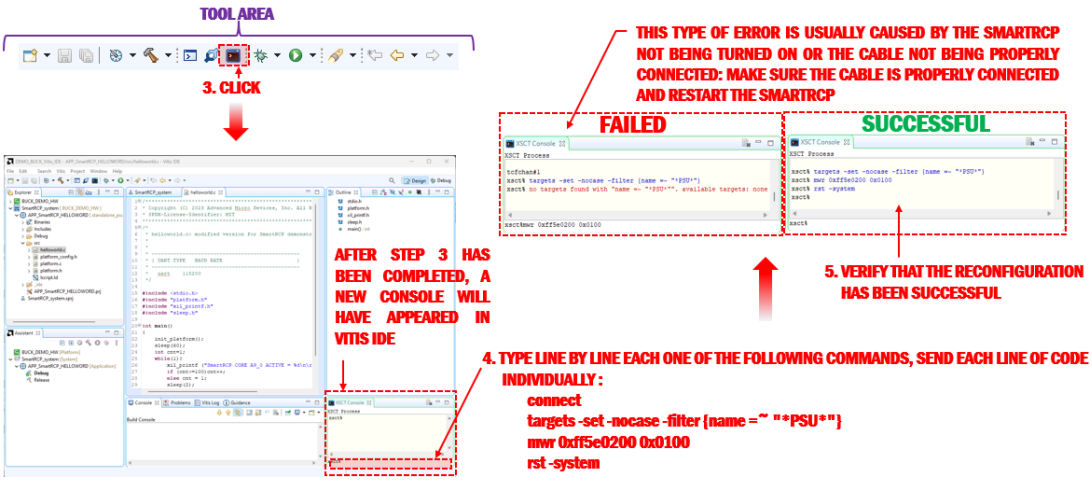


Figure 153: JTAG programming, steps 3 to 5: opening the XSCT console, the four commands typed one by one, and the console output when the reconfiguration fails or succeeds.

By executing the commands of Figure 153 (Listing 2), the developer temporarily reconfigures the boot mode of the SmartRCP. By default the device boots from the microSD card each time it is powered on or rebooted with the reset button. These commands

modify the SmartRCP registers so that, on receiving the reset instruction, the device starts in JTAG mode and can be programmed from Vitis IDE. The configuration is temporary: every time the power is turned off or the reset button is pressed, the SmartRCP reverts to the default boot mode (microSD) and is deprogrammed. In those cases the commands must be entered again in the console, one by one, pressing Enter after each line.

Listing 2: XSCT commands that switch the boot mode to JTAG.

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

Once the programming mode of the board has been switched to JTAG, the SmartRCP is ready to be programmed from Vitis IDE. Follow the steps of Figure 154.

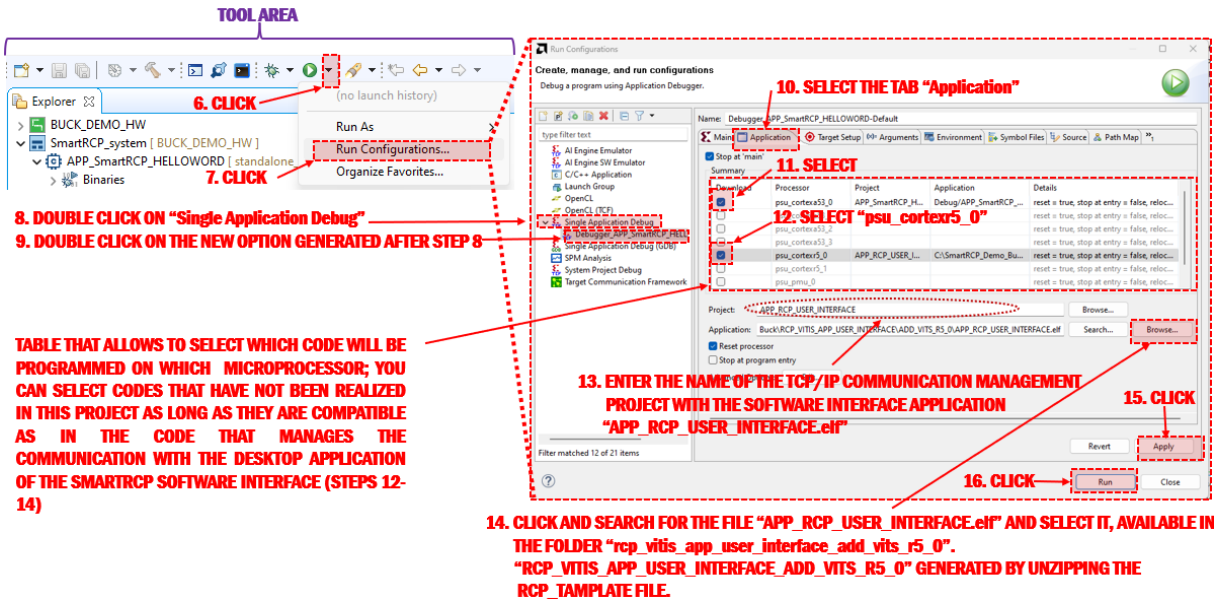


Figure 154: JTAG programming, steps 6 to 16: Run Configurations, Single Application Debug, and the Application tab where the interface application is assigned to core R5_0.

In step 12 a new microprocessor is added that is not managed by any of the processes previously created in this project. If, instead of a new microprocessor, it were another process already existing in the project, steps 13 and 14 would not be necessary. In step 13 a name is assigned to the process associated with the new microprocessor; in this example it is the application that manages the communication with the SmartRCP. In step 14 the location of the compiled code of that application is specified.

For multiprocess applications such as this one it is essential to manage the memory allocation properly: each application must be stored in a different memory area of the SmartRCP, avoiding conflicts between processes.

There is also the option of launching only the SmartRCP communication management application, unchecking all the other applications as shown in Figure 155. This is use-

ful when the user does not need any additional microprocessor in the project and the functionalities provided by the software user interface are sufficient.

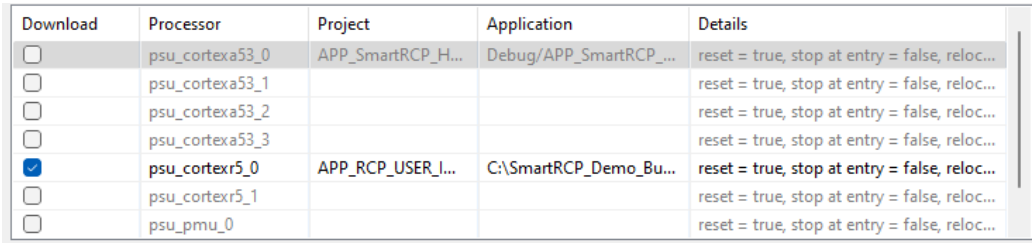


Figure 155: Run configuration with only the interface application selected, on core R5_0.

If the procedure has been executed correctly, the messages of Figure 156 appear in the console opened in step 3, confirming that the programming has succeeded and that the processes are running.

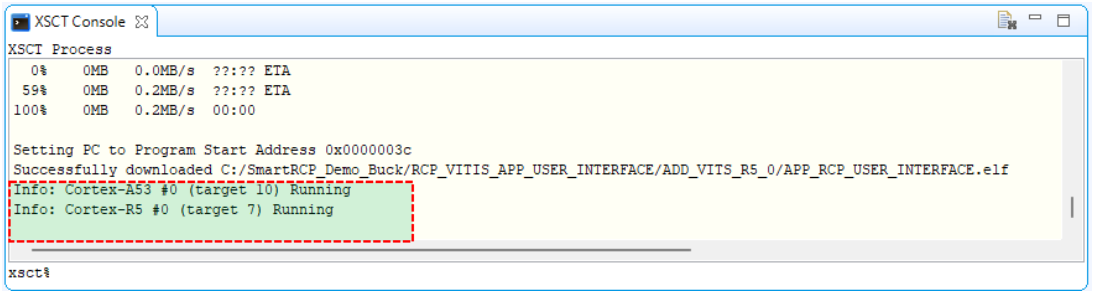


Figure 156: XSCT console after a successful download: the A53 and R5 targets report Running.

7.7.2 Generating the boot file for the microSD card

Booting from the microSD card is the default mode of the SmartRCP. Its main advantage is that the device starts automatically as soon as it is turned on or restarted, without manual reconfiguration. The method provides greater security and operational stability: after a reboot or a power failure the SmartRCP reloads the complete program by itself, FPGA design and microprocessor software included. For this configuration the user generates a binary file that is copied to the microSD card.

The steps to generate the `BOOT.bin` file of the buck converter demonstrator are shown below. The file, together with the network configuration file, is available in the `DEMO_BUCK_BOOT_BIN` folder of the `SmartRCP_Demo_Buck` archive, but the user can also generate it with these steps, which apply to any boot file.

Unlike the JTAG procedure, which is intended for debugging and requires a permanent active connection between the computer and the SmartRCP, this method does not require the SmartRCP to be connected while the file is generated. A programming file (`BOOT.bin`) is generated that contains both the digital design of the FPGA and the microprocessor software. The file is copied to the microSD card, which is then inserted

into the card reader of the SmartRCP: when the device is turned on or rebooted, it loads the program from the card automatically. Follow the steps of Figure 157 in Vitis IDE, starting from the project created in the previous sections.

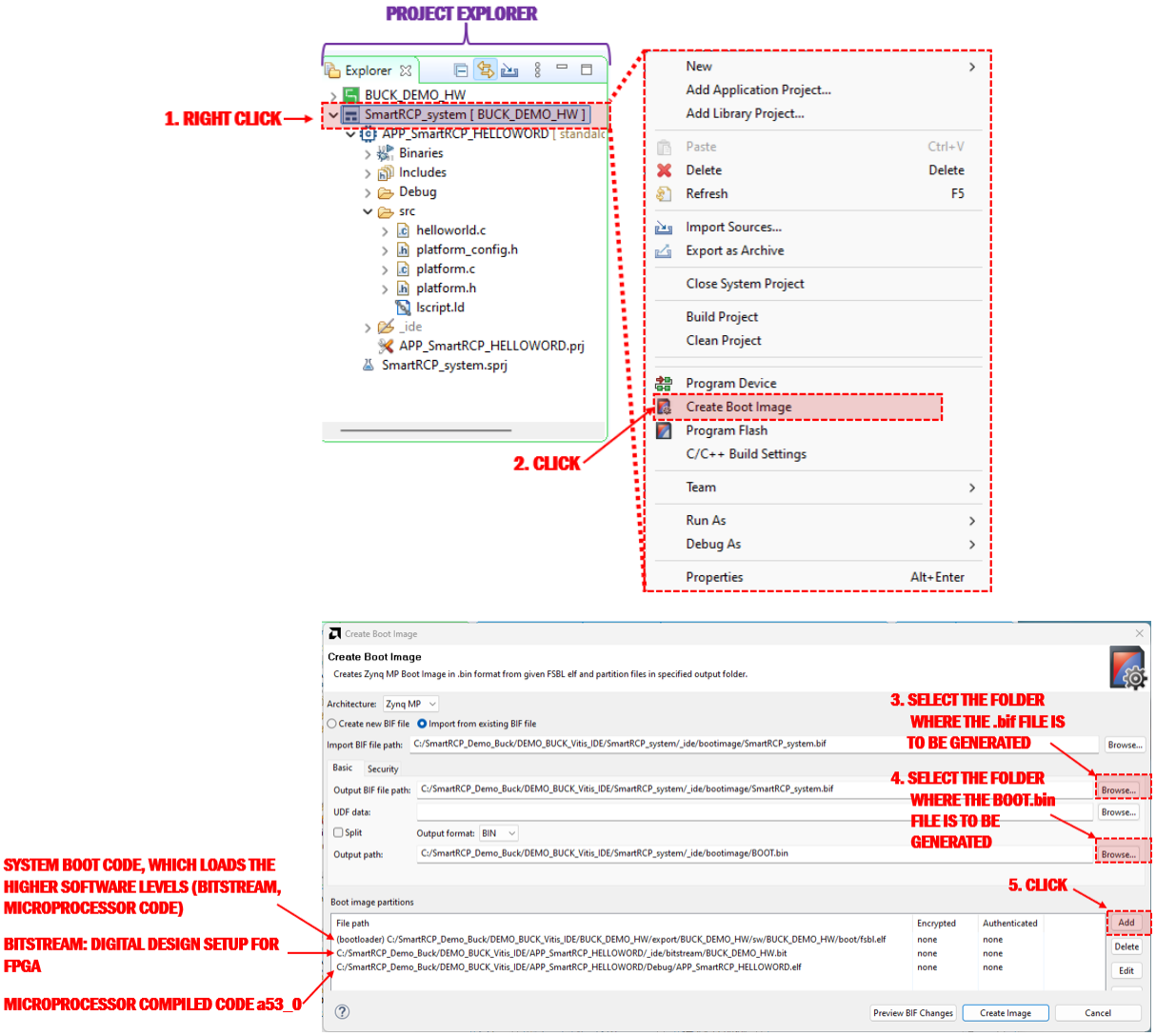


Figure 157: Boot image, steps 1 to 5: Create Boot Image from the system project, the output folders and the partition list.

A `BOOT.bin` file combines three types of file that make up the complete boot system:

1. **FSBL** (`fsbl.elf`). The First Stage Boot Loader initialises the system hardware, configuring the fundamental elements such as the memory and the peripherals. When the design includes the FPGA, the FSBL also loads the bitstream into it. It is the first step of the boot process and prepares the system to load the next level of software or the operating system.
2. **Bitstream** (`.bit`). The hardware design configuration of the FPGA. When loaded, it sets up the internal structure of the FPGA, configuring the logic and hardware resources needed for the design to work as specified in the Vivado project.

3. **ELF files (`.elf`).** One `.elf` file for each microprocessor that runs in parallel in the system, holding the executable code of each SoC core. If the design uses several processors, a specific `.elf` is required for each core, so that each one loads and executes its application independently and in coordination with the others.

The combination of these three elements in `BOOT.bin` lets the complete system be configured and started, with hardware and software started in the correct order and with the correct configuration.

As Figure 157 shows, when the Hello World project is used as the basis for the boot file, only the code of the microprocessor that runs the Hello World application is included. The code for the R5_0 core, reserved for the application that communicates with the SmartRCP desktop interface, must be added manually: include it in step 5 as shown in Figure 158.

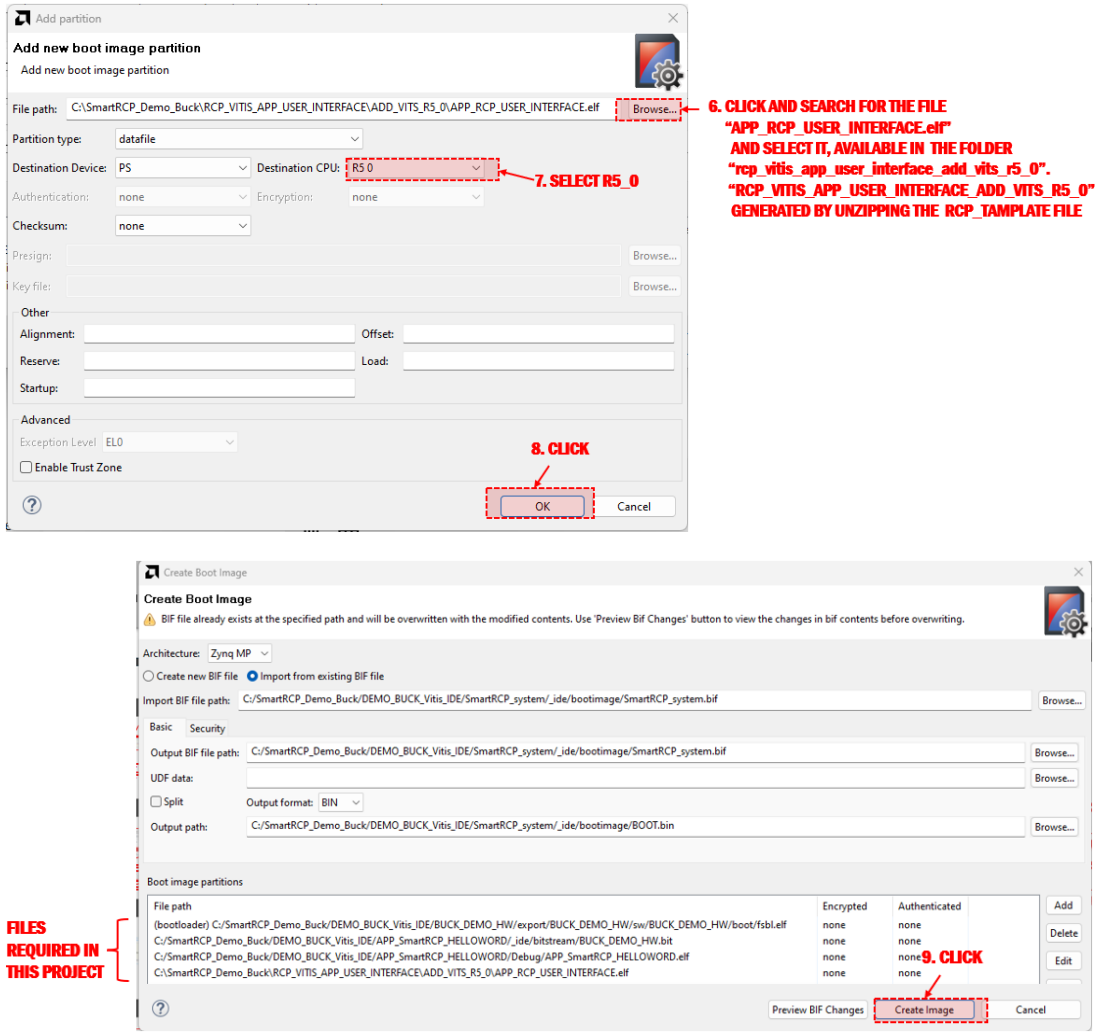


Figure 158: Boot image, steps 6 to 9: adding the interface application partition for core R5_0 and creating the image with the four files.

At the end of step 9 the `BOOT.bin` file has been generated in the folder specified in step 4. Copy it to the microSD card together with the Ethernet configuration file (`PSC_TCP.txt`).

From then on, every time the SmartRCP is turned on with the microSD card inserted, the device is programmed automatically with the buck converter project.

7.8 Customization of the SmartRCP software interface

This section explains how to customize the control console that manages the SmartRCP from the computer, focusing on the customizations specific to the buck converter project. For a more complete use of the interface and all its functionalities, consult the *SmartRCP Interface* manual. The customizations required in this project are:

- **ADC QUICK parameters.** ADC QUICK 1 measures the voltage at the output of the converter; ADC QUICK 2 measures the current in the converter inductance.
- **Write registers.** These registers control the converter. Representative names are assigned to each parameter and the corresponding values are defined, which makes them easier to identify and control.
- **Read registers.** In this project their function is mainly instructional: they show the results of the internal operations performed by the data processing IP block. It is explained how to make them visible in the interface and how to customize their names.
- **PL-DataLogger inputs.** How to customize the name of each input and how to configure the inputs as active probes that capture data in real time. For the potential and advanced options of the PL-DataLogger, see the *SmartRCP Interface* manual.

Finally, it is explained how to save and load the whole configuration, so that it is available whenever the user needs it.

Set the screen scaling to 100 %. Otherwise the interface may display incorrectly and leave some buttons out of the user's reach.

Run the SmartRCP Interface application, located in `SmartRCP_Demo_Buck\RCP_USER_SOFTWARE_APP` of the unzipped folder (executable `SmartRCP Interface 2024.2`). Then configure the ADC parameters with the steps of Figure 159 and the values of Table 35.

Configure the write registers that control the buck converter with the steps of Figure 160.

Configure the read registers with the steps of Figure 161, and the PL-DataLogger with the steps of Figures 162 and 163.

At the end of step 23 the desktop application is customized to control the buck converter directly from the computer: it allows monitoring the waveforms generated on



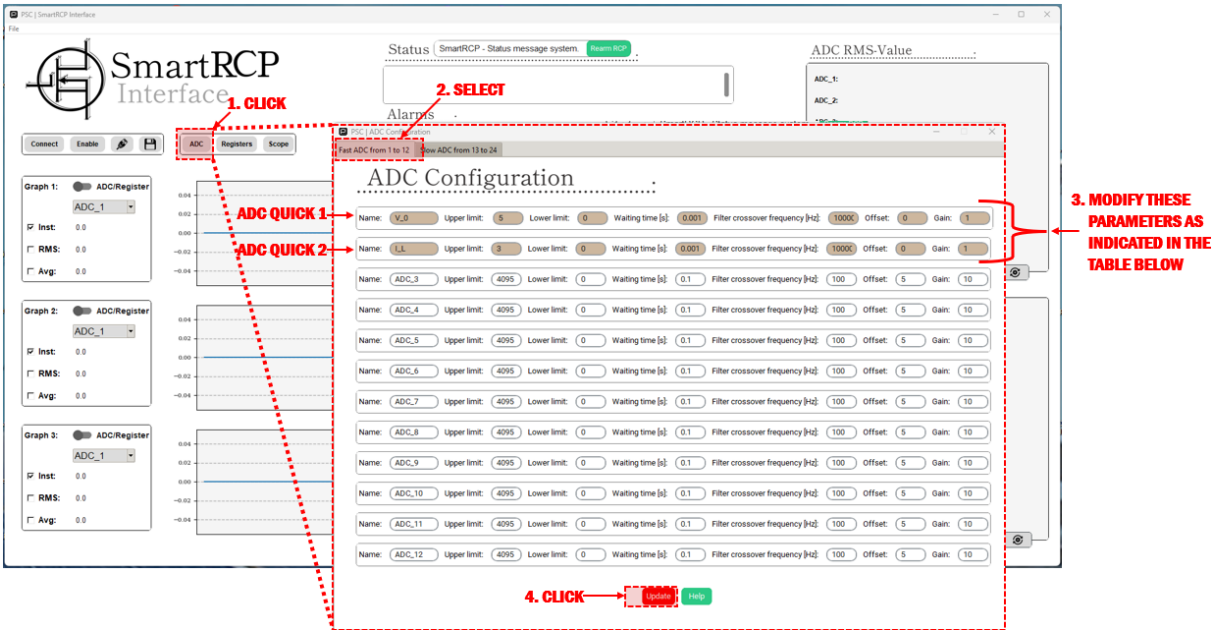


Figure 159: Interface customization, steps 1 to 4: the ADC configuration window with the two ADC QUICK channels and the Update button.

Parameter	ADC QUICK 1	ADC QUICK 2
Name	`V_0`	`I_L`
Upper limit	5 [V]	3 [A]
Lower limit	0 [V]	0 [A]
Waiting time	0.001 [s]	0.001 [s]
Filter crossover frequency	1 000 000 [Hz]	1 000 000 [Hz]
Offset	0 [V]	0 [A]
Gain	1	1

Table 35: ADC QUICK parameters of the buck converter project.

the device and adjusting, as necessary, the control variables for all the tests required during development.

The next step is to save the configuration, so that the next time the user needs to perform tests the configuration file can simply be loaded, instead of repeating the whole process, and all the variables set before are loaded automatically into the application. Follow the steps of the upper part of Figure 164.

All the software interface settings made in this tutorial are available in the archive of the buck converter demonstrator, in the `SmartRCP_Demo_Buck\DEMO_BUCK_SMARTRCP_APP_CONF` folder, saved in the file `SmartRCP_CONF.txt`. To load a saved configuration into a session of the desktop application, follow the steps of the lower part of Figure 164.

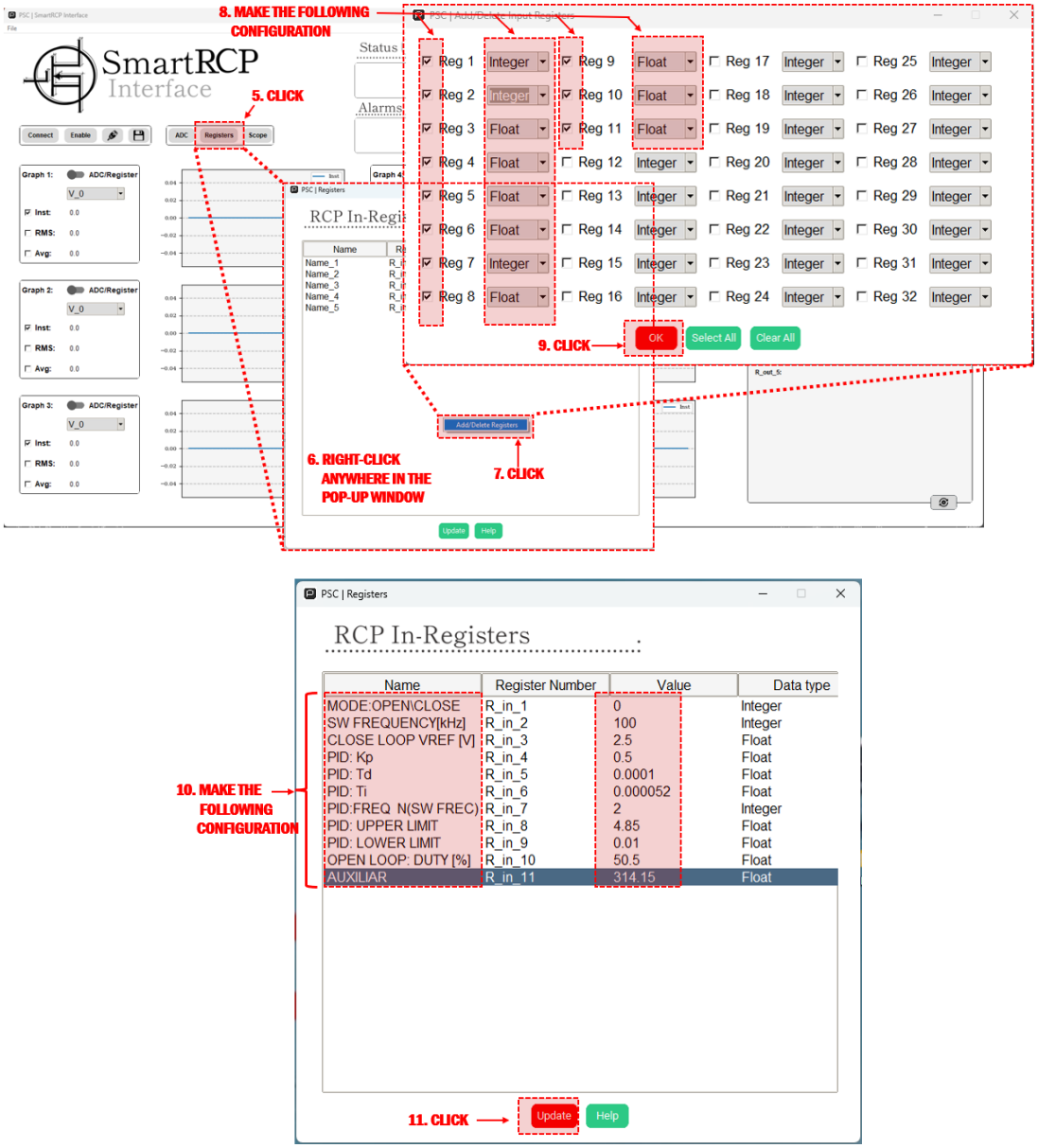


Figure 160: Interface customization, steps 5 to 11: enabling registers 1 to 11 with their data types, then the names and initial values of the write registers.

All the current interface settings are overwritten when a configuration file is loaded: every existing parameter is replaced by the value specified in the loaded file.

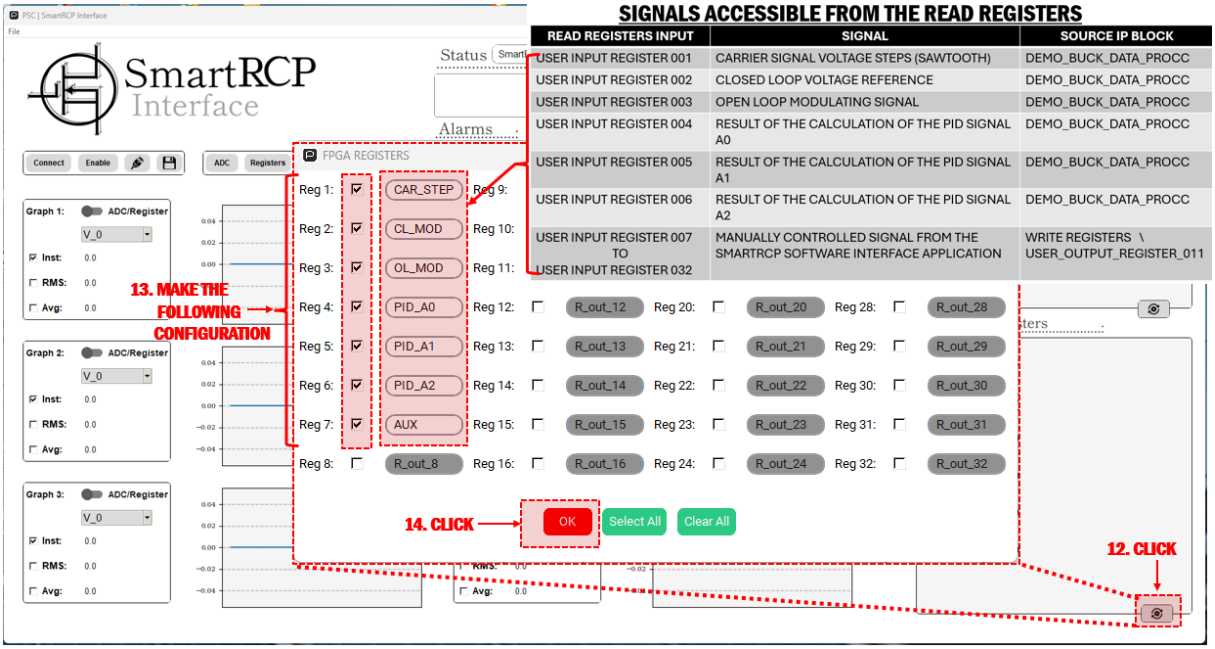


Figure 161: Interface customization, steps 12 to 14: the read registers enabled and named after the signals of Table 31.

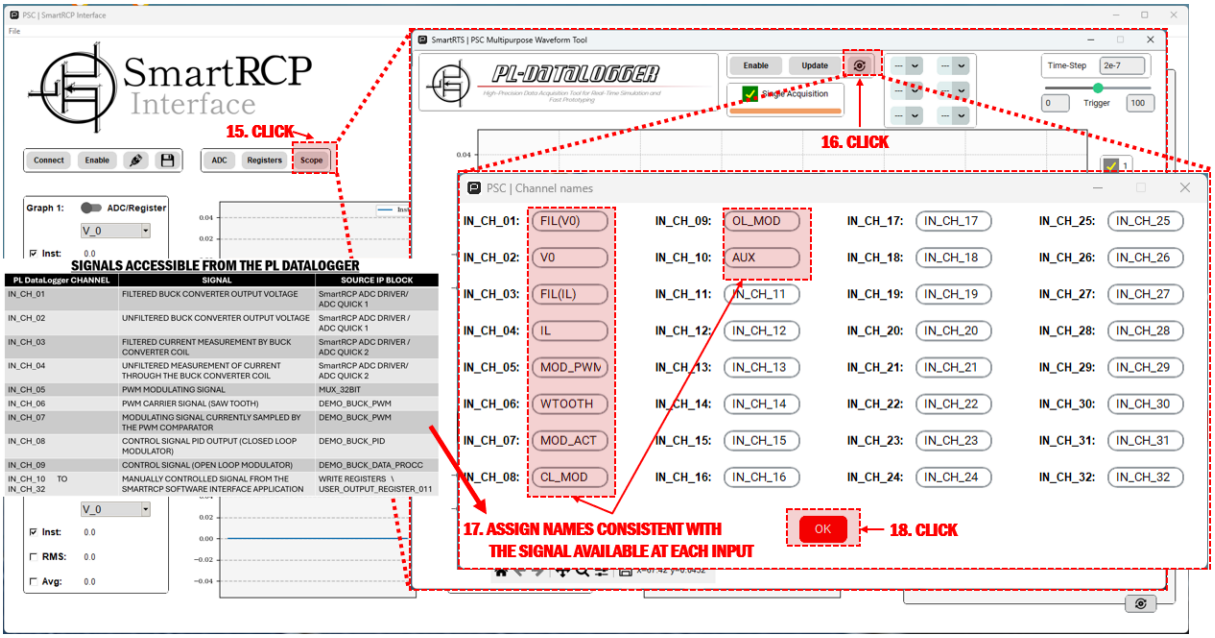


Figure 162: Interface customization, steps 15 to 18: the PL-DataLogger channel names, consistent with Table 30.

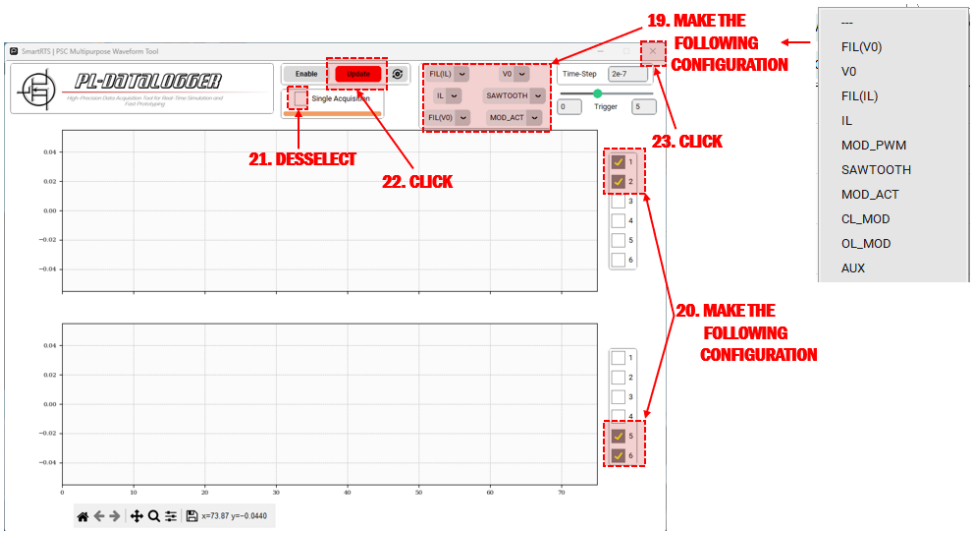


Figure 163: Interface customization, steps 19 to 23: probe selection and time step of the PL-DataLogger.

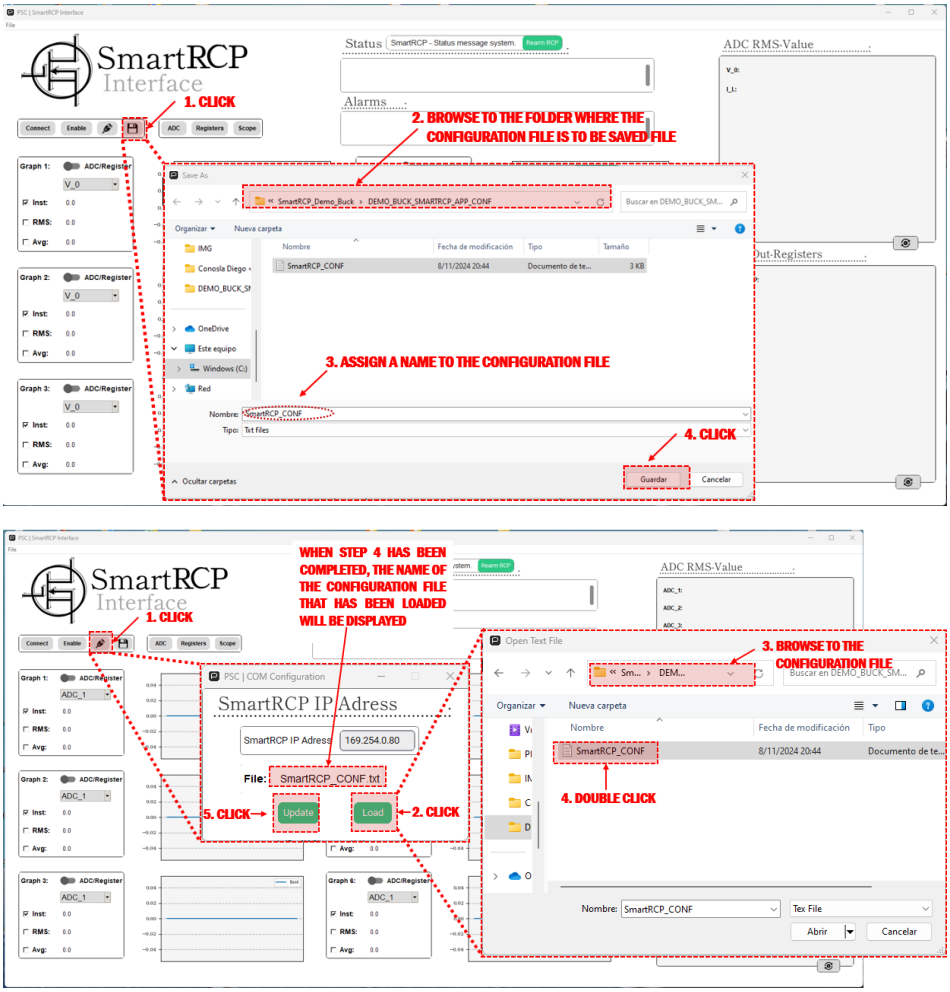


Figure 164: Saving (top, steps 1 to 4) and loading (bottom, steps 1 to 5) an interface configuration: the save button and the file dialog; the COM configuration window, the Load button and the selection of the file.

7.9 Final results: control of the buck converter

This final section of the example project, which demonstrates the use and customization of the SmartRCP template to control a buck converter, covers the following points:

- **List of materials and equipment connections.** The material needed to carry out the tests, and the connection scheme of the equipment involved.
- **Guided functional test.** A guided test with which the user verifies the correct operation of the design: both the control of the buck converter and the response of the SmartRCP software interface to real-time monitoring and adjustments.

7.9.1 List of required materials and equipment connections

To perform this test the user needs the following materials:

1. SmartRCP (board and charger).
2. Buck converter hardware set: one buck converter PCB, one USB power cable, one charger, two sensing cables and one Ethernet cable for the trigger signal of the converter.
3. Computer (the user's computer).
4. Oscilloscope (optional).
5. Ethernet cable for the TCP/IP communication with the user's computer.
6. USB cable with a Micro-B connector for the SmartRCP and, at the other end, the connector that fits the port available on the user's computer (or an adapter).
7. microSD card.
8. The compressed folder of the finished project. In this example the finished project provided in a compressed file is used; if the user has followed all the steps of this tutorial, the project generated from the RCP_TEMPLATE can be used instead. In both cases the final result should be identical. Figure 165 maps the contents of the archive.

Many computers have no built-in Ethernet port. In that case use an adapter, or a router with a Wi-Fi connection. This example only considers the direct connection between the SmartRCP and the user's computer.

Make the connections shown in Figure 166.



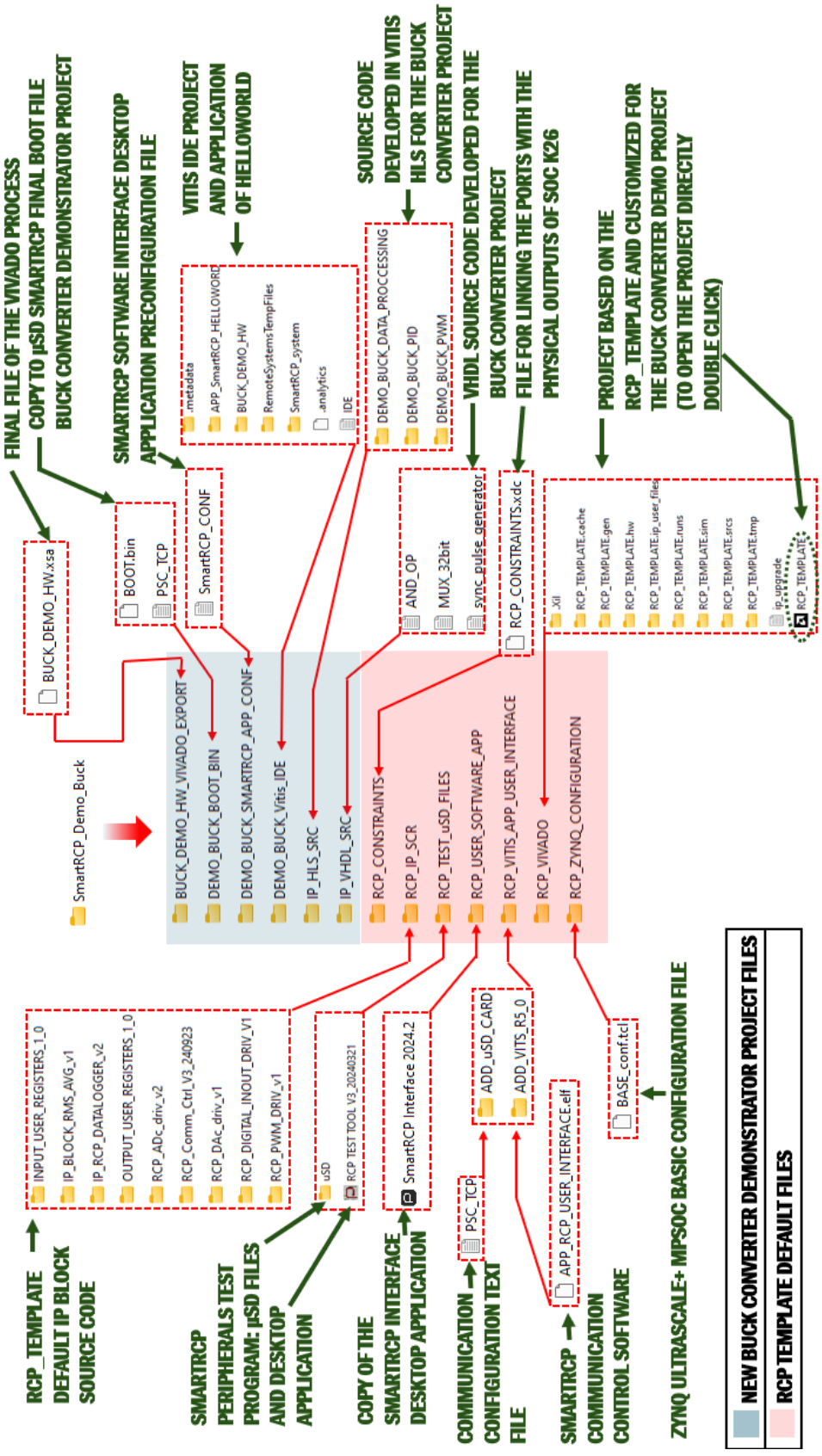


Figure 165: Contents of the SmartRCP_Demo_Buck archive: the RCP_TEMPLATE default files and the files added by the buck converter demonstrator.

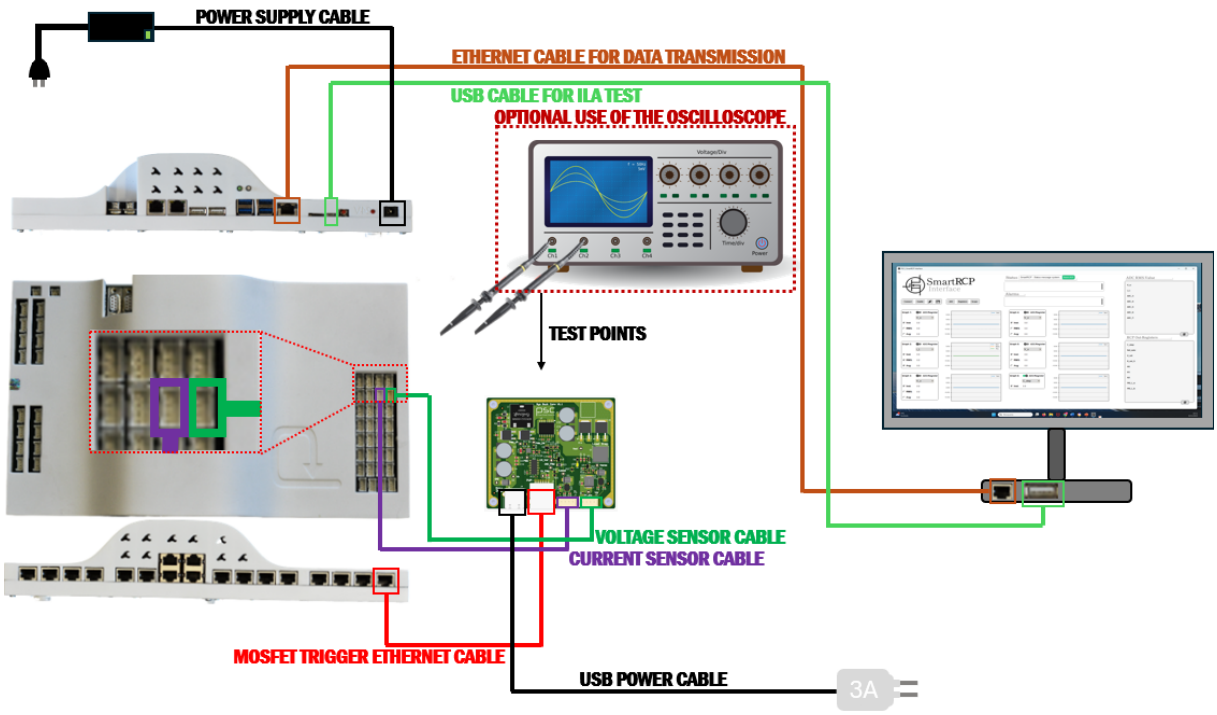


Figure 166: Test setup: power supply, Ethernet link to the computer, USB cable for the ILA test, sensing cables, PWM trigger cable, USB power of the converter and the optional oscilloscope on the test points.

7.9.2 Guided test for functionality verification

In this section the user is guided through a structured test to evaluate the developed control and to explore some of the advanced functionalities of the SmartRCP desktop application. The test has three parts:

1. **Connecting the SmartRCP to the desktop application.** The SmartRCP is connected directly to the software application over Ethernet, without a router, which allows the tutorial to be performed in a simplified environment.
2. **Modification of control parameters and monitoring of results.** Once the connection is configured, parameters such as the closed-loop reference voltage and the switching frequency are modified. Their effect is observed in real time on the PL-DataLogger screen and compared with the results obtained simultaneously on a physical oscilloscope, focusing on the current waveform through the converter inductance. Then the cut-off frequency of the low-pass filter (initially 1 MHz, Table 35) is adjusted to analyse its effect on the measured and filtered voltage signal. Finally, the alarm parameters are modified to force their activation and observe the response of the system.
3. **Demonstration of the Vivado ILA functionality.** To close the test, a brief demonstration of the Integrated Logic Analyzer (ILA) of Vivado, a very useful tool for debugging and verifying the operation of the design blocks.

Direct connection between the SmartRCP and the interface

Follow the steps of Figure 167 to prepare the microSD card, then continue with steps 5 and 6 below.

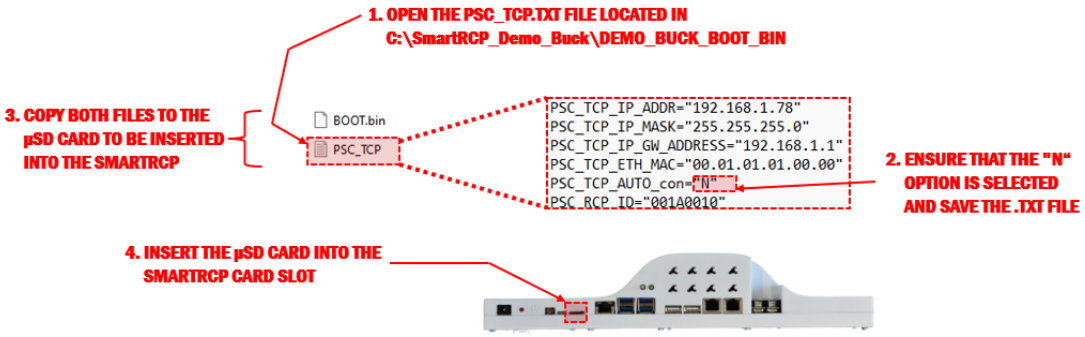


Figure 167: Direct connection, steps 1 to 4: the `PSC_TCP.txt` file with the manual IP configuration, copied with `BOOT.bin` to the microSD card and inserted into the SmartRCP.

5. Set the computer to assign its IP address manually. Make sure the subnet mask and gateway match those of the configuration file (`PSC_TCP.txt`). These parameters must be the same on the SmartRCP and on the computer; they can be modified as long as they remain consistent on both. Choose an IP address for the computer in the same range but different from the one assigned to the SmartRCP. With the default values of the configuration file, the following settings can be used:
   - Subnet mask: 255.255.255.0
   - Gateway: 192.168.1.1
   - Preferred DNS: 192.168.1.1
   - Computer IP address: 192.168.1.100
6. Launch the SmartRCP software interface desktop application from its executable, and continue with the steps of Figure 168.

11. Connect the SmartRCP charger to the SmartRCP and then to the power outlet.
12. Connect the USB charger to the buck converter and then to the power outlet.

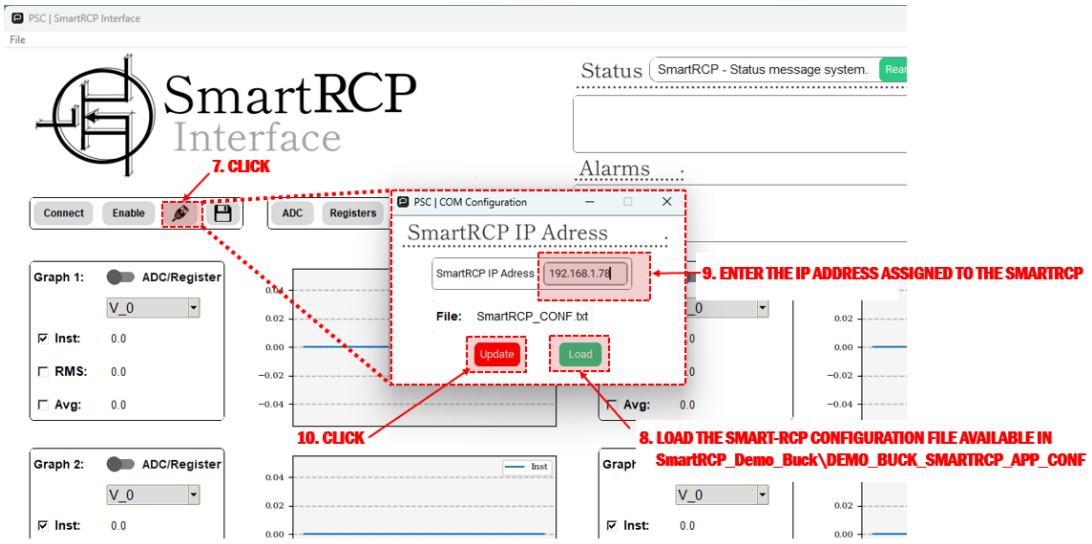


Figure 168: Direct connection, steps 7 to 10: the COM configuration window, the IP address of the SmartRCP, loading the saved configuration and Update.

Control parameters and monitoring

Continue with the steps of Figures 169 to 172.

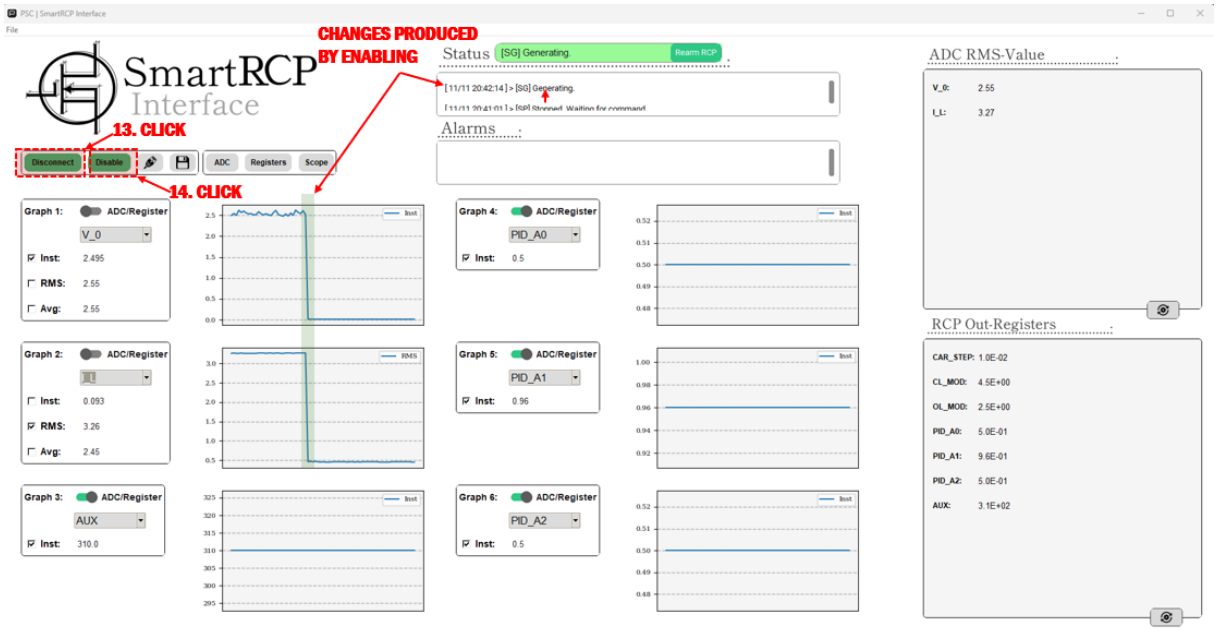


Figure 169: Guided test, steps 13 and 14: Connect and Enable, and the values that appear in the ADC readouts and the read registers once the converter is running.

The user can modify the control parameters and observe their effect on the current and voltage waveforms. Figure 173 compares, at two operating points, a current capture obtained with a bench oscilloscope with the values recorded by the SmartRCP.

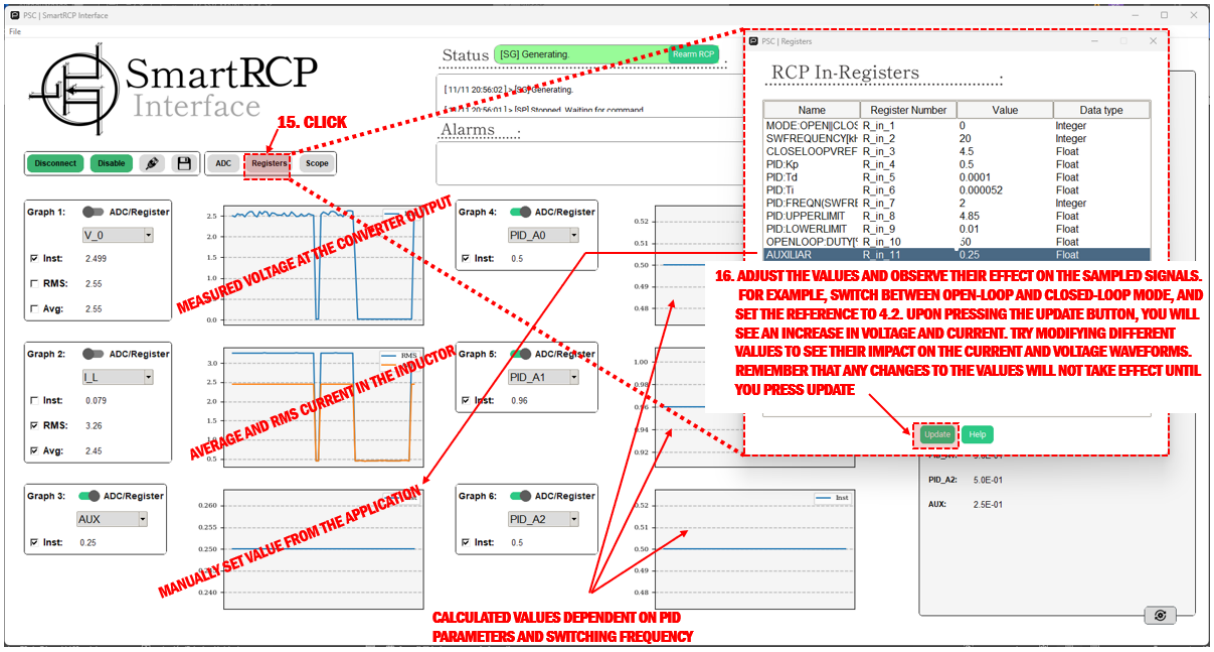


Figure 170: Guided test, steps 15 and 16: the write registers window; the effect of the loop mode and reference changes on voltage and current.

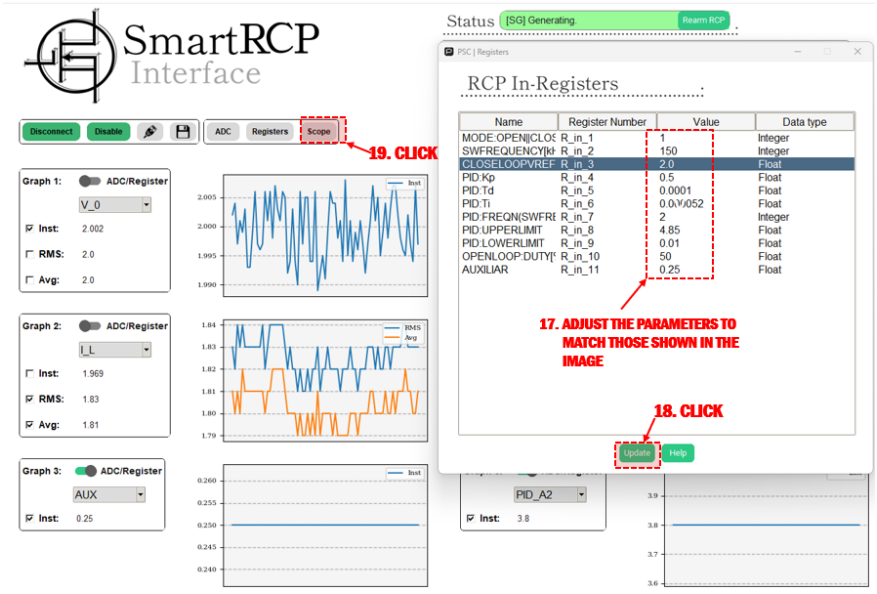


Figure 171: Guided test, steps 17 to 19: register values of the test, Update and the Scope button.

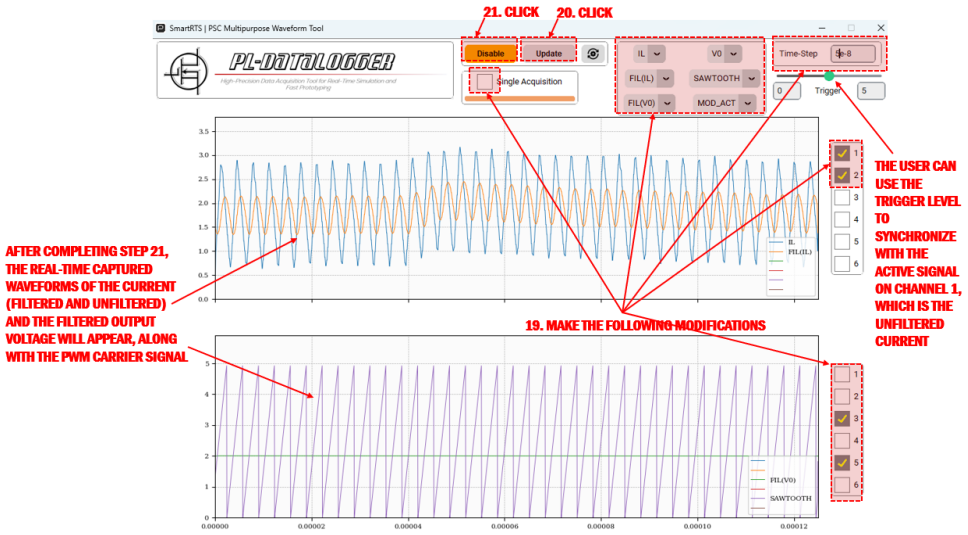


Figure 172: Guided test, steps 19 to 21: PL-DataLogger configuration and the real-time capture of the filtered and unfiltered current, the filtered output voltage and the PWM carrier.

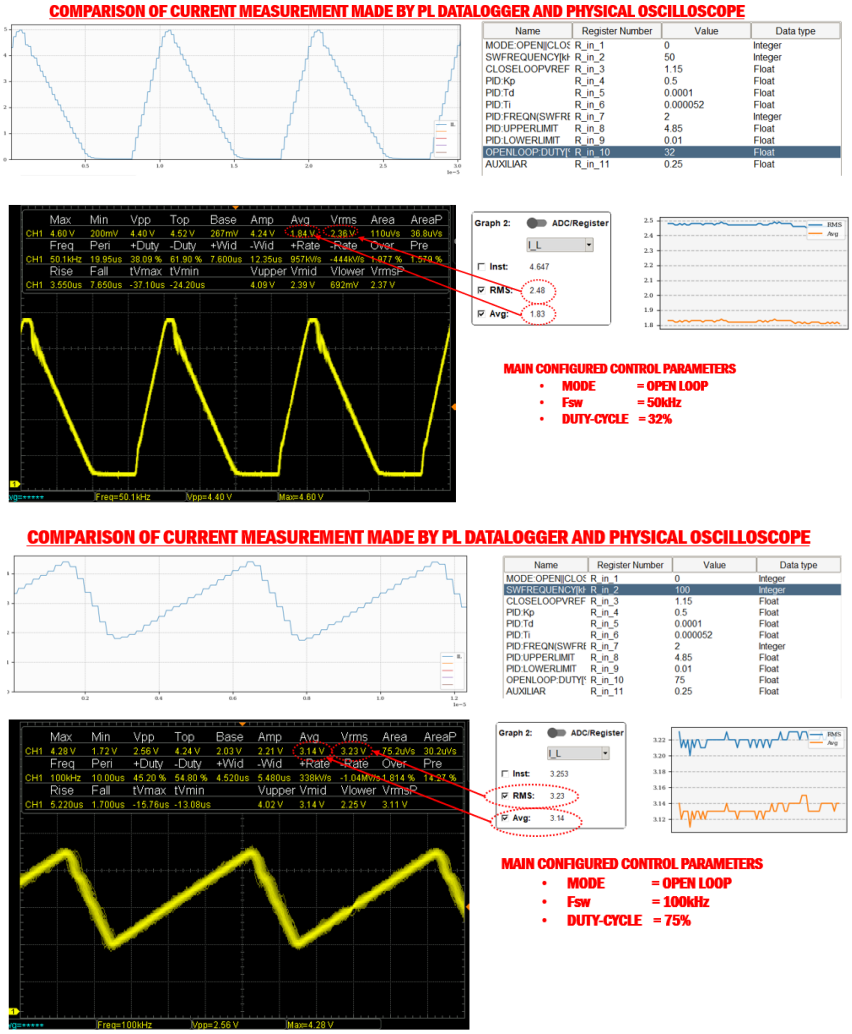


Figure 173: Inductor current measured by the PL-DataLogger and by a bench oscilloscope in open loop: top, 50 kHz and 32 % duty cycle; bottom, 100 kHz and 75 % duty cycle.

Alarm test

Next, set the maximum current limit so as to trigger the alarm event and observe how the SmartRCP protections act, stopping the signal generation automatically. Before starting, switch to closed-loop control with the parameters of the previous test, then follow the steps of Figures 174 and 175. Figure 176 shows the waveforms as the reference voltage is raised, and Figure 177 the interface once the alarm has tripped.

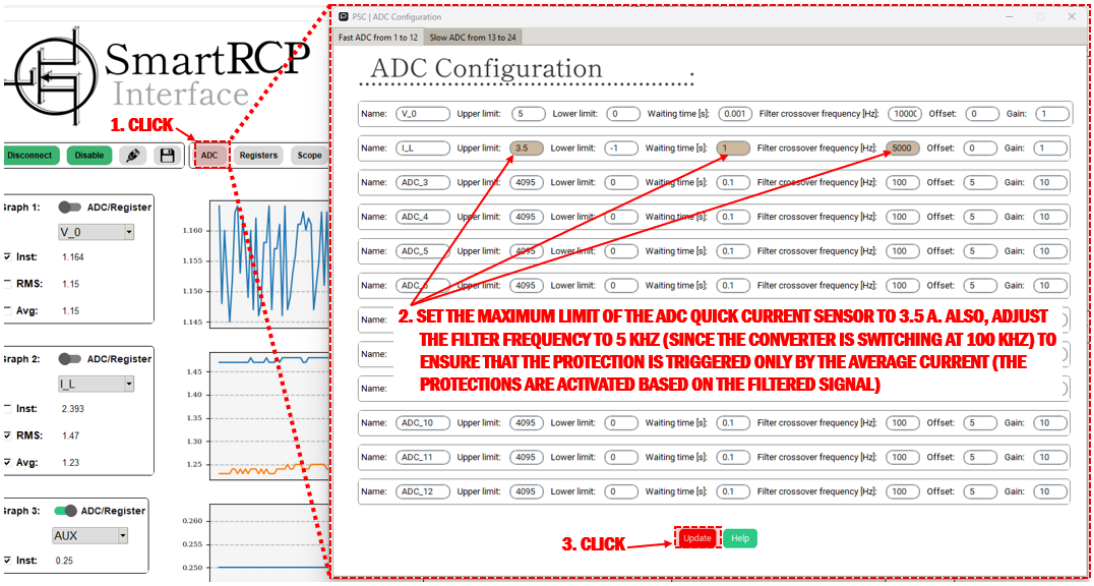


Figure 174: Alarm test, steps 1 to 3: current limit of ADC QUICK 2 set to 3.5 A and its filter to 5 kHz, then Update.

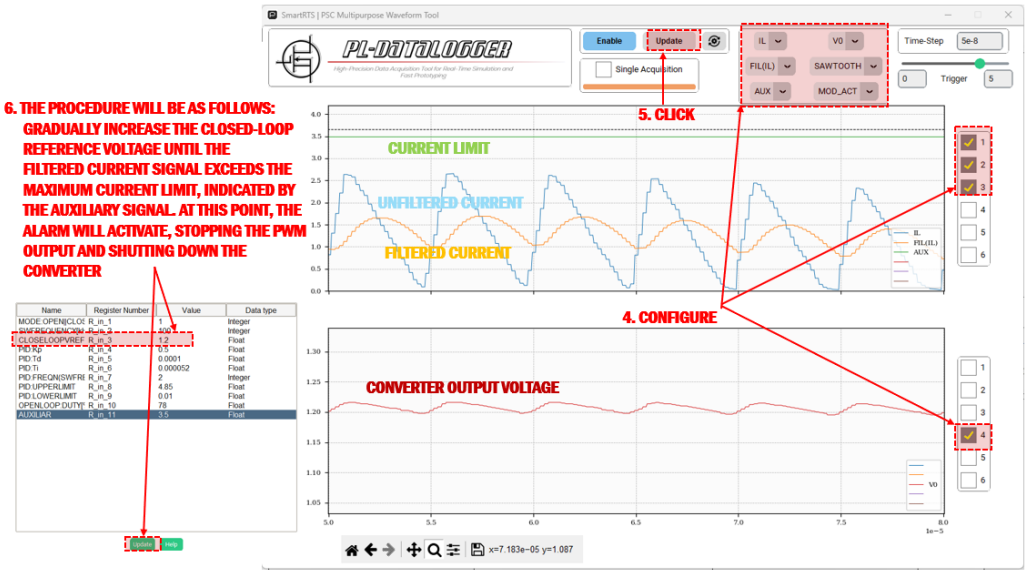


Figure 175: Alarm test, steps 4 to 6: PL-DataLogger channels for the current limit, the unfiltered and filtered current and the output voltage, while the closed-loop reference is raised.



Figure 176: Alarm test: current and voltage for reference voltages of 2, 3, 4 and 4.25 V; at 4.25 V the alarm trips, the PWM stops and voltage and current drop to zero.

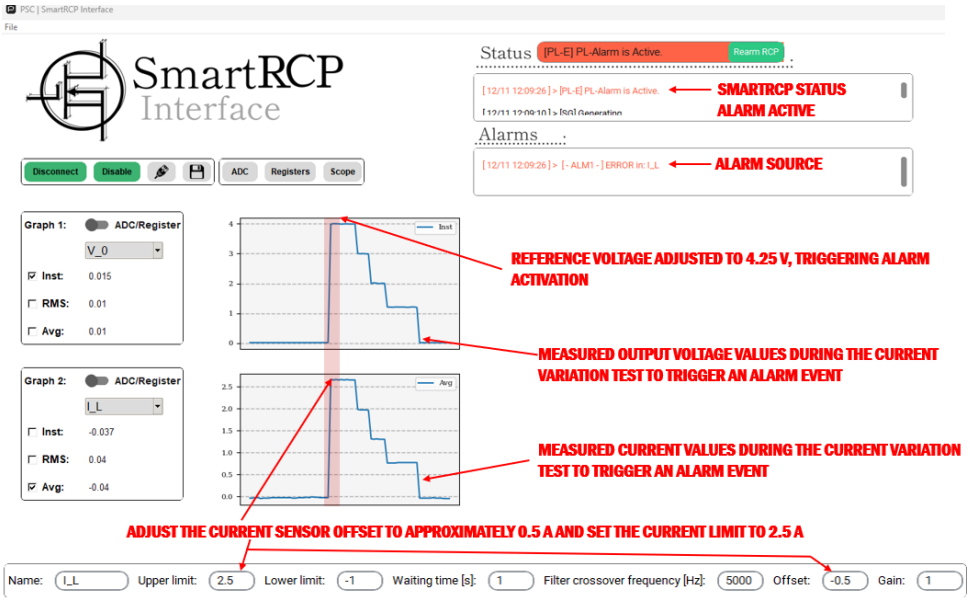


Figure 177: Alarm test: the interface with the alarm active, its source in the alarm log, and the voltage and current recorded during the test.

Debugging with the ILA

To conclude the tutorial, the use of the Integrated Logic Analyzer (ILA) for advanced debugging is demonstrated. First open Vivado and make sure that the SmartRCP board is connected to the computer with the USB cable. Then follow the steps of Figure 178 to start the ILA analysis.

After step 5 the Vivado Hardware Manager of the SmartRCP opens. This is where the analysis with the ILA is performed. The Hardware Manager also provides other data

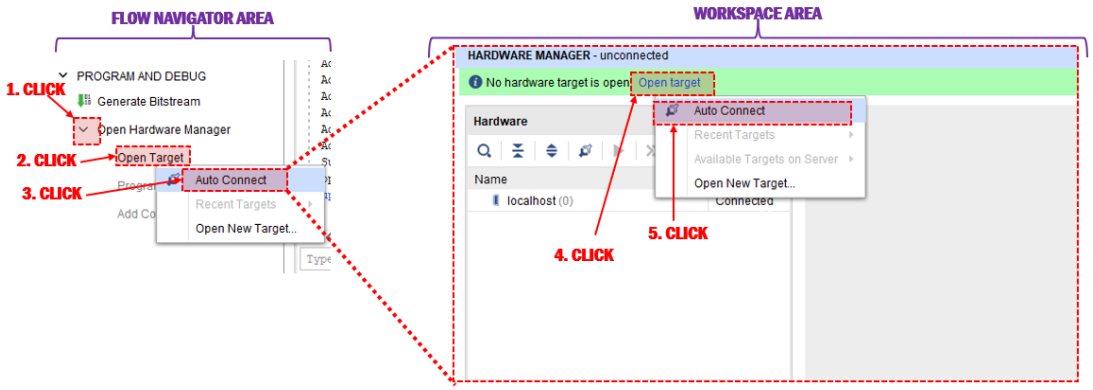


Figure 178: ILA, steps 1 to 5: Open Hardware Manager, Open Target and Auto Connect.

of interest about the SmartRCP, such as the unique serial number of the board, which may be needed when requesting technical support from the SmartRCP manufacturer. Figure 179 highlights the key areas of the Hardware Manager the user should know as a minimum to use the ILA, when it has been integrated in the design. Pressing the start-capture button acquires the values of the signals connected to the ILA (Figure 180).

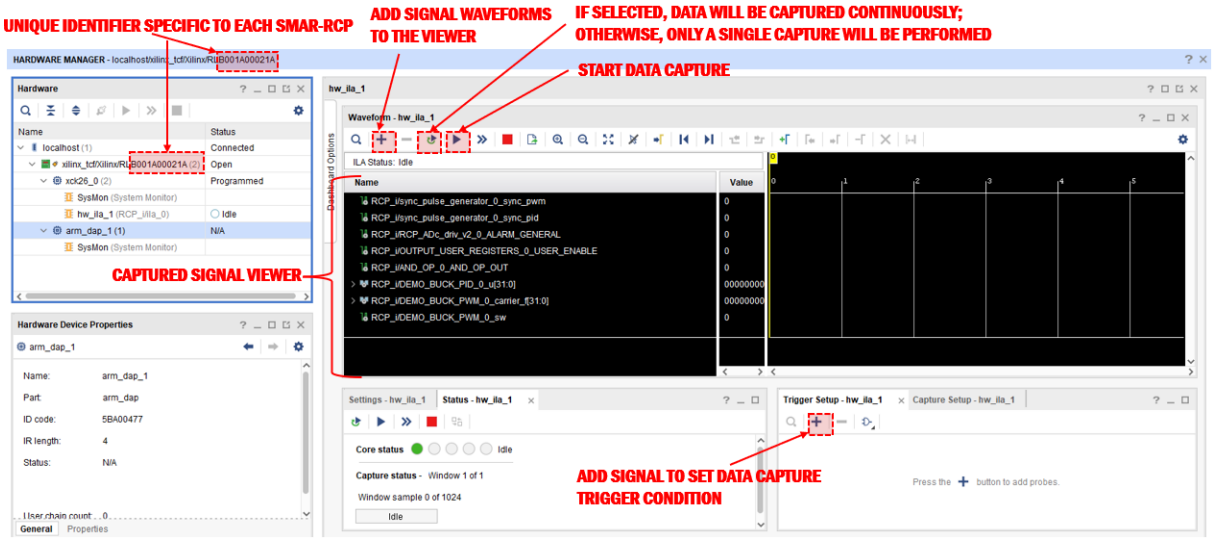


Figure 179: The Hardware Manager: unique identifier of the SmartRCP, the captured-signal viewer, the capture controls and the trigger setup.

This concludes the guided example of the development of a control for a buck converter. To take full advantage of the Xilinx tools (Vivado, Vitis, ILA), refer to the documentation available on the Internet. Likewise, to maximise the use of the SmartRCP software interface desktop application, the user is advised to review the *SmartRCP Interface* manual.

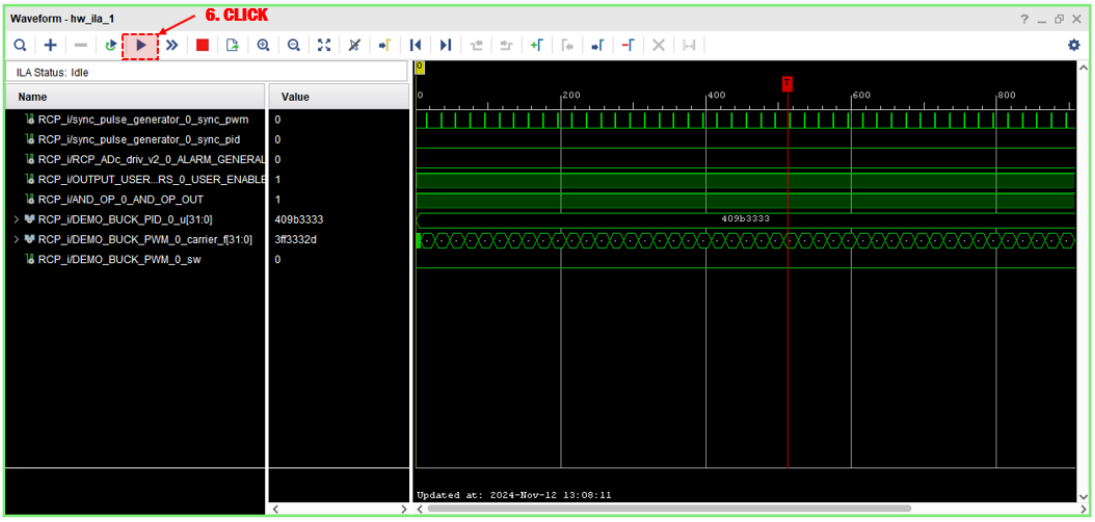


Figure 180: ILA, step 6: a capture of the eight probes of the demonstrator.

8

Terms of Use, Limitations and Warranty

8.1 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.



8.2 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.

8.3 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, Nº 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.

8.4 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.

9 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

www.powersmartcontrol.com

Copyright © 2026 Power Smart Control S.L. All rights reserved.

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

A.1 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.

Figure 181 presents a simplified schematic of an NTC sensor, linearized and with the total gain controlled by an Operational Amplifier (OpAmp):

The NTC reference shall be selected from a catalog to fit the measuring range required. Rlineal is used to ensure the relationship between temperature and voltage drop in the

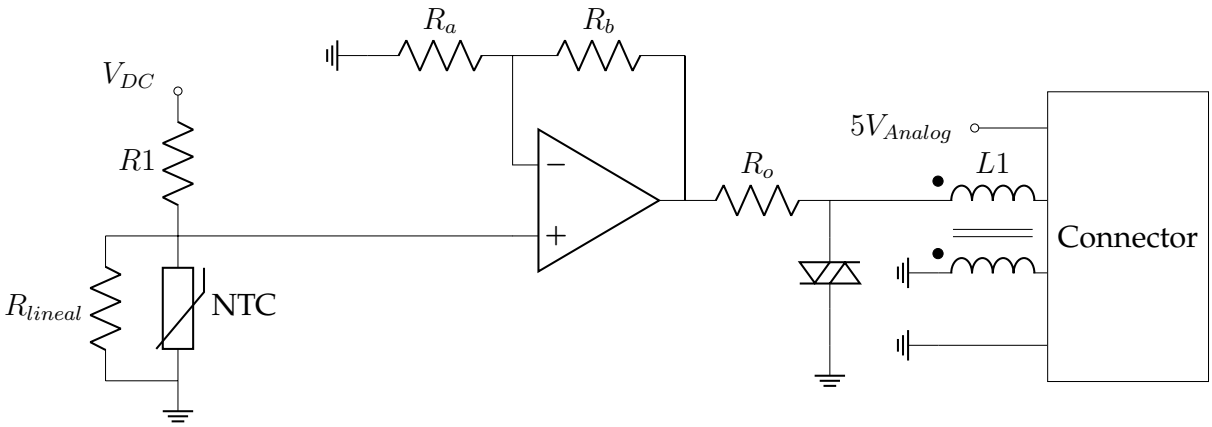


Figure 181: NTC temperature sensing analog chain.

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-Rlineal system and increase it as much as possible to fill the OpAmp input span (typically 0-5V).

Ra and Rb 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 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:

- NTC model: NTCALUG01A103JA
- TVS model: SMAJ6.0A
- Operational amplifier model: OPA192
- Choke model (L1): Wurth Elektronik 744235801
- Connector model: Molex 53375-0410
- Rlineal=910Ohm as R(NTC) at 90 °C = 910Ohm
- R1 = 10kOhm
- Ra=5kOhm, Rb=55kOhm
- Vout (0 °C) = 4.92V

- Vout (50 °C) = 4.08V
- Vout (100 °C) = 2.28V
- Ro = 100R
- Power is generated by additional power supply, 5V_ana is left unconnected.
- **Maximum output analog span must be between 0V and 5V**

Figure 182 provides a complete schematic for this sensor.

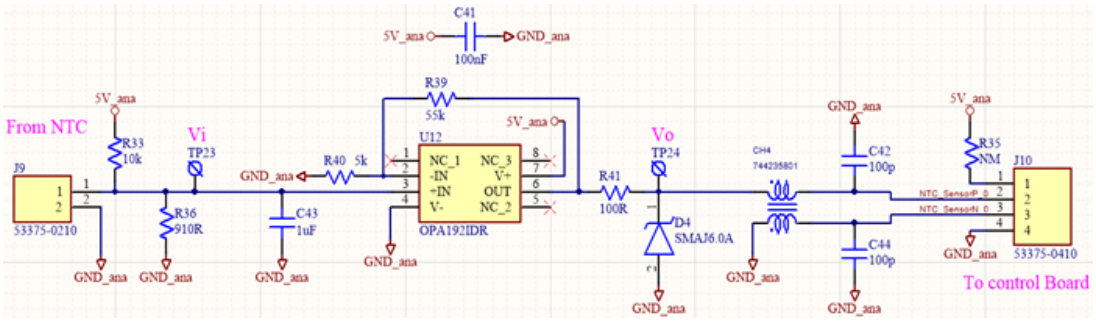


Figure 182: NTC temperature sensing schematic.

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.

A.2 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 183 provides the simplified scheme for a LEM current sensing stage:

Rload and Rin 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.

Ra and Rb 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.

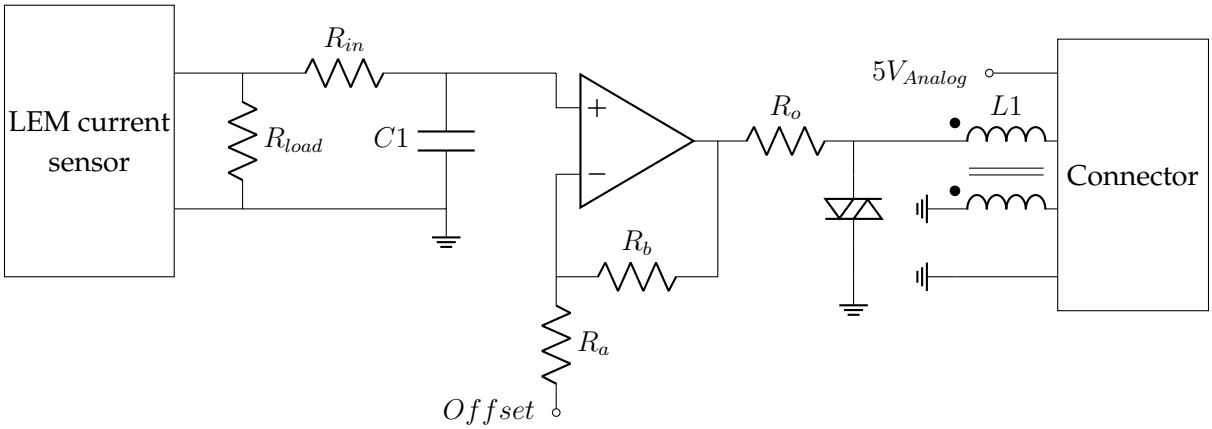


Figure 183: LEM current sensing analog chain.

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): Wurth Elektronik 744235801
- Connector model: Molex 53375-0410
- Rload=10kOhm
- Rin=22R
- C1=10nF
- Ra=10kOhm / Rb=30kOhm / Voffset=2.5V
- Vout (-200Amp) = 0V
- Vout (0Amp) = 2.5V
- Vout (200Amp) = 5V
- Ro = 100Ohm
- Power is generated by additional power supply, 5V_analog is left unconnected.
- **Maximum output analog span must be between 0V and 5V**

Figure 184 provides a detailed view of the schematic of a LEM current sensor:

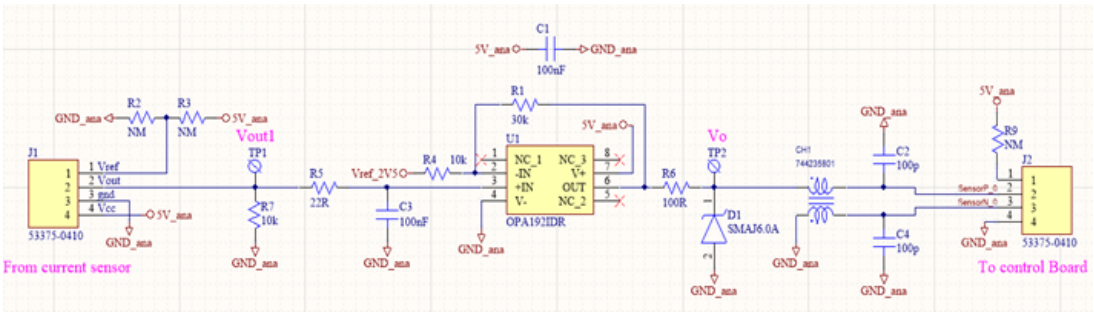


Figure 184: LEM current sensing schematic.

A.3 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. Figure 185 shows a simplified view of the analog chain.

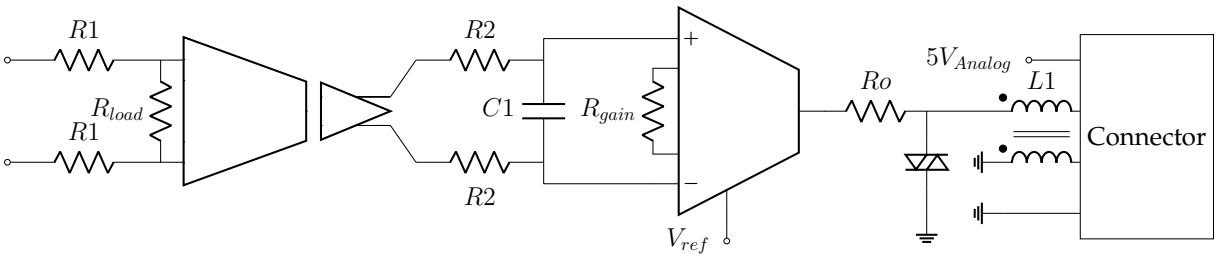


Figure 185: DC voltage sensing analog chain.

R1 and Rload shall be selected to drain the smallest amount of current (power) from the converter while ensuring the voltage drop on Rload is admissible by the isolated OpAmp input span. Due to dissipation or electrical security, R1 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=3.28MOhm (four 820KOhm resistors in series)
- Rload=10kOhm
- R2=4.7kOhm
- C1=10nF
- Rgain=33kOhm
- Vref= 0V (ground connection)
- Ro=100Ohm
- Instrumentation OpAmp: INA826
- Vo (Vin=0V) = 0V
- Vo (Vin=800V) = 3.02V
- Vo (Vin=1314V) = 5V
- Power is generated by additional power supply, 5V_analog is left unconnected.
- **Maximum output analog span must be between 0V and 5V**

Figure 186 provides a detailed view of the schematic for a DC isolated voltage sensor analog chain.



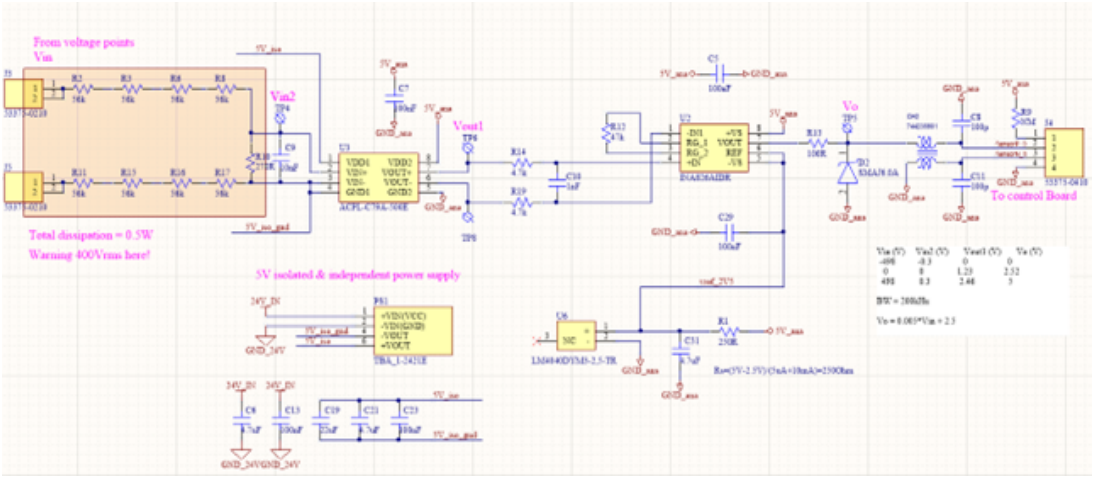


Figure 186: DC voltage sensing schematic.

A.4 Designing an AC isolated voltage sensor analog chain

Same precautions as in DC voltage sensing case shall be followed. Figure 187 provides a simplified schematic.

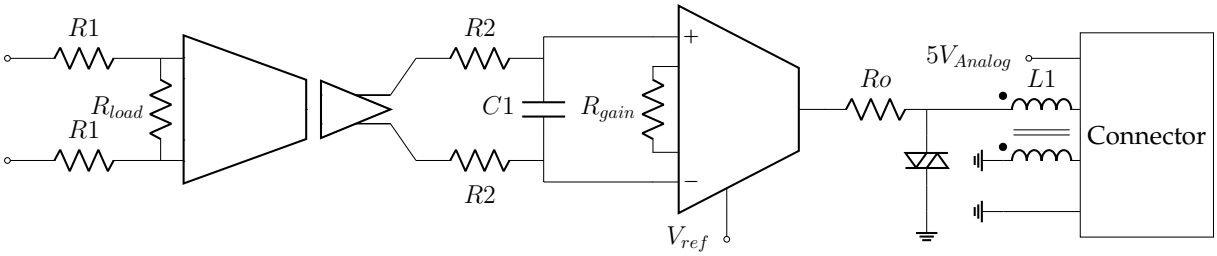


Figure 187: AC voltage sensing analog chain.

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 span. Due to dissipation or electrical security, R1 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.

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 resistors in series)
- Rload=270Ohm
- R2=4.7kOhm
- C1=10nF
- Rgain=Not Mounted
- Vref= 2.5V (generated with LM4040)
- Ro=100Ohm
- Instrumentation OpAmp: INA826
- Vo (Vin=-498V) = 0V
- Vo (Vin=0V) = 2.52V
- Vo (Vin=498V) = 5V
- Power is generated by additional power supply, 5V_analog is left unconnected.
- **Maximum output analog span must be between 0V and 5V**

Figure 188 provides a detailed schematic of an AC isolated voltage sensor analog chain.

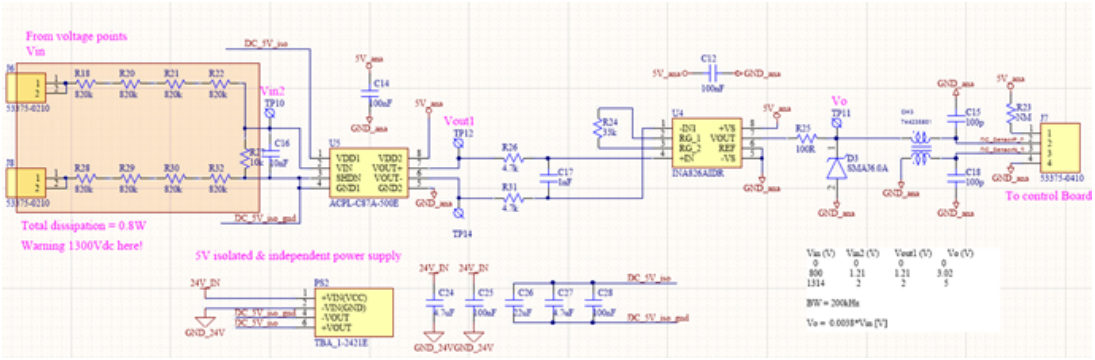


Figure 188: AC voltage sensing schematic.

References

[1] AMD Xilinx, *Kria K26 System-on-Module*, product page. https://www.xilinx.com/products/som/kria/k26c-commercial.html

[2] AMD Xilinx, *Kria K26 SOM Data Sheet* (DS987), PL Resources table.

[3] L. H. Crockett, D. Northcote, C. Ramsay, F. D. Robinson and R. W. Stewart, *Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications* (the Zynq MPSoC Book), Strathclyde Academic Media, 2019. Figure 3.2, p. 34.

