User guide for REF-22K-GPD-INV-EASY3B

A reference design for a general purpose drive

About this document

Scope and purpose

This user guide provides an overview of the reference kit REF-22K-GPD-INV-EASY3B including its main features, key data, pin assignments, mechanical dimensions and corresponding control card. The reference kit REF-22K-GPD-INV-EASY3B is an industrial motor drive for three-phase 400 V AC grids and has a nominal power of 22 kW.

The REF-22K-GPD-INV-EASY3B includes the Easy3B power module FP100R12W3T7_B11, the current sensor TLIh4971-A120T5, the gate driver IC 1ED3240MC12H, the 1.7 kV SiC-MOSFET IMBF170R1K0M1 and the microcontrollers XMC4800-F144F2048 and XMC4300-F100K256. The combination of these Infineon products allows the customer to evaluate these products in one design and experience the interaction between the products.

Beside the hardware, the REF-22K-GPD-INV-EASY3B offers you software for control and communication. The inverter can be controlled via PC-GUI.

Note: *Please note that this product is not qualified according to the AEC Q100 or AEC Q101 documents of the Automotive Electronics Council.*

Intended audience

This user guide is intended for all technical specialists working on industrial drives and those interested in how the latest Infineon products like IGBT7, XENSIV™ current sensors, EICE™ drivers and CoolSiC™ MOSFETs perform under application conditions.

Reference Kit

The Infineon products are embedded on this PCB with functions and form factor close to a commercial design. PCB and auxiliary circuits are optimized for the final design.

Note: *Boards do not necessarily meet safety, EMI, quality standards (for example UL, CE) requirements.*

Important notice

"Evaluation Boards and Reference Boards" shall mean products embedded on a printed circuit board (PCB) for demonstration and/or evaluation purposes, which include, without limitation, demonstration, reference and evaluation boards, kits and design (collectively referred to as "Reference Board").

Environmental conditions have been considered in the design of the Evaluation Boards and Reference Boards provided by Infineon Technologies. The design of the Evaluation Boards and Reference Boards has been tested by Infineon Technologies only as described in this document. The design is not qualified in terms of safety requirements, manufacturing and operation over the entire operating temperature range or lifetime.

The Evaluation Boards and Reference Boards provided by Infineon Technologies are subject to functional testing only under typical load conditions. Evaluation Boards and Reference Boards are not subject to the same procedures as regular products regarding returned material analysis (RMA), process change notification (PCN) and product discontinuation (PD).

Evaluation Boards and Reference Boards are not commercialized products, and are solely intended for evaluation and testing purposes. In particular, they shall not be used for reliability testing or production. The Evaluation Boards and Reference Boards may therefore not comply with CE or similar standards (including but not limited to the EMC Directive 2004/EC/108 and the EMC Act) and may not fulfill other requirements of the country in which they are operated by the customer. The customer shall ensure that all Evaluation Boards and Reference Boards will be handled in a way which is compliant with the relevant requirements and standards of the country in which they are operated.

The Evaluation Boards and Reference Boards as well as the information provided in this document are addressed only to qualified and skilled technical staff, for laboratory usage, and shall be used and managed according to the terms and conditions set forth in this document and in other related documentation supplied with the respective Evaluation Board or Reference Board.

It is the responsibility of the customer's technical departments to evaluate the suitability of the Evaluation Boards and Reference Boards for the intended application, and to evaluate the completeness and correctness of the information provided in this document with respect to such application.

The customer is obliged to ensure that the use of the Evaluation Boards and Reference Boards does not cause any harm to persons or third party property.
The Evaluation Boards and Reference Boards and any information in this document is provided "as is" and Infineon Technologies disclaims any warranties, express or implied, including but not limited to warranties of non-infringement of third party rights and implied warranties of fitness for any purpose, or for merchantability.

Infineon Technologies shall not be responsible for any damages resulting from the use of the Evaluation Boards and Reference Boards and/or from any information provided in this document. The customer is obliged to defend, indemnify and hold Infineon Technologies harmless from and against any claims or damages arising out of or resulting from any use thereof.

Infineon Technologies reserves the right to modify this document and/or any information provided herein at any time without further notice.

Safety precautions

Note: Please note the following warnings regarding the hazards associated with development systems.

Table 1 Safety precautions

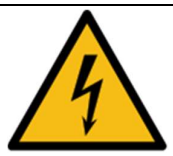
	**Warning:** *The DC link potential of this board is up to 1000 VDC. When measuring voltage waveforms by oscilloscope, high voltage differential probes must be used. Failure to do so may result in personal injury or death.*
	Warning: *The evaluation or reference board contains DC bus capacitors which take time to discharge after removal of the main supply. Before working on the drive system, wait five minutes for capacitors to discharge to safe voltage levels. Failure to do so may result in personal injury or death. Darkened display LEDs are not an indication that capacitors have discharged to safe voltage levels.*
	Warning: *The evaluation or reference board is connected to the grid input during testing. Hence, high-voltage differential probes must be used when measuring voltage waveforms by oscilloscope. Failure to do so may result in personal injury or death. Darkened display LEDs are not an indication that capacitors have discharged to safe voltage levels.*
	Warning: *Remove or disconnect power from the drive before you disconnect or reconnect wires, or perform maintenance work. Wait five minutes after removing power to discharge the bus capacitors. Do not attempt to service the drive until the bus capacitors have discharged to zero. Failure to do so may result in personal injury or death.*
	Caution: *The heat sink and device surfaces of the evaluation or reference board may become hot during testing. Hence, necessary precautions are required while handling the board. Failure to comply may cause injury.*
	Caution: *Only personnel familiar with the drive, power electronics and associated machinery should plan, install, commission and subsequently service the system. Failure to comply may result in personal injury and/or equipment damage.*
	Caution: *The evaluation or reference board contains parts and assemblies sensitive to electrostatic discharge (ESD). Electrostatic control precautions are required when installing, testing, servicing or repairing the assembly. Component damage may result if ESD control procedures are not followed. If you are not familiar with electrostatic control procedures, refer to the applicable ESD protection handbooks and guidelines.*
	Caution: *A drive that is incorrectly applied or installed can lead to component damage or reduction in product lifetime. Wiring or application errors such as undersizing the motor, supplying an incorrect or inadequate AC supply, or excessive ambient temperatures may result in system malfunction.*
	Caution: *The evaluation or reference board is shipped with packing materials that need to be removed prior to installation. Failure to remove all packing materials that are unnecessary for system installation may result in overheating or abnormal operating conditions.*

Table of contents

About this document **1**
Important notice **2**
Safety precautions **3**
Table of contents **4**
1 The reference board at a glance **6**
1.1 Delivery content 6
1.2 Block diagram 6
1.3 Main features 7
1.4 Board parameters and technical data 8
1.4.1 Overload profile normal duty 9
1.4.2 Heavy duty overload profile 9
1.5 Advantages of two level slew rate gate drivers 10
2 System design **11**
2.1 Schematics 11
2.2 Layout 11
2.2.1 Power board 11
2.3 Inverter cooling concept 13
2.4 Bill of material 14
2.5 Connector details 14
3 Software structure **17**
3.1 Overview 17
3.2 R-CPU Software 17
3.2.1 Real-time Domain 18
3.2.1.1 Auto-generated Code 19
3.2.1.2 Control-Framework 19
3.2.1.3 Measurement and Conditioning 20
3.2.1.4 Protection 21
3.2.1.5 Background Domain 22
3.2.1.6 Tasks 23
3.2.1.7 Main State Machine 23
3.2.1.8 UART Communication 25
3.2.1.9 XScope 25
3.2.2 Hardware Abstraction (DAVE) 25
3.2.3 Signals and Parameters 25
3.2.4 Folder and file reference 33
3.3 Inter-Processor Communication 34
3.3.1 Packet Structure and Code Layers 34
3.3.2 State Machine 35
3.3.3 Command Reference 36
3.4 C-CPU Software 37
3.4.1 Structure 37
3.4.2 Parameter Database 38
3.4.3 Folder and file reference 39
4 System and functional description **40**
4.1 Usage 40
4.1.1 Prerequisites and Installation 40

4.1.2 Installing the GPD software package ... 40
4.1.3 Installing PLECS ... 40
4.1.4 Installing the J-Link drivers for flashing ... 40
4.1.5 Installing DAVE ... 41
4.1.5.1 Installation ... 41
4.1.5.2 Compiling the R-CPU software package ... 41
4.1.5.3 Compiling the C-CPU software package ... 41
4.1.6 Installing the WinUSB drivers ... 41
4.1.7 Setting up the XScope software oscilloscope ... 42
4.1.8 Setting up the GUI ... 42
4.1.9 Setting up the GPDTarget and PLECS model ... 42
4.1.9.1 Setting up the GPDTarget ... 42
4.1.9.2 Setting up the PLECS model ... 42
4.1.9.3 Setting up the code generation and flashing ... 43
4.2 Graphical Programming and Workflow ... 43
4.2.1 Model configuration ... 43
4.2.2 Coder configuration ... 44
4.2.3 Parameter definition ... 45
4.2.4 Code generation ... 46
4.2.5 GPD target library reference ... 46
4.2.6 Example models ... 47
4.2.7 V/f ... 47
4.2.8 Debugging with XScope ... 50
4.3 GPD Operation ... 53
4.4 General tab ... 53
4.5 Diagnostics view ... 54
4.5.1 ParamDB tab ... 57
5 Commissioning ... 59
5.1 Description of the functional blocks ... 63
5.1.1 Hardware partitioning ... 63
5.1.2 Isolation coordination ... 65
5.1.3 Board interconnection schemes ... 66
5.1.4 Supply schemes ... 67
5.1.5 Component selection ... 67
5.2 Setup two level slew rate gate driver ... 69
6 System performance ... 71
6.1 Test results inverter start-up ... 71
6.2 Operation under rated conditions ... 72
6.3 Switching behavior of the IGBT – turn on/off ... 74
6.4 Thermal behavior of the inverter under nominal operation ... 75
6.5 Short-circuit measurement ... 77
7 References and appendices ... 80
7.1 Abbreviations and definitions ... 80
7.2 References ... 80
Revision history ... 81

1 The reference board at a glance

1.1 Delivery content

The reference kit is a general purpose drive developed for applications like pumps, fans, compressors, conveyor belts. The design has the look and feel of a typical drive and includes EMI filter, pre-charge and capacitor bank, isolated power supplies, power module, controls and heat sink with fan. It can be operated directly on a three-phase grid or by a external high voltage DC-supply, enabling a fast evaluation of Infineon's newest technologies like IGBT7, gate driver, current sensor and control in one system. This enables the unique opportunity to see the improvement by combining Infineon's newest technologies. You will see how the new IGBT7 modules work with the EICE™ gate driver, or test the accuracy of the current sensor.

The drive has been tested according to DIN EN IEC 61800-5-1. It passed the isolationstest according to the specified requirements of the shortcircuit powerterminals and shortcircuit SELV terminals.

1.2 Block diagram

The block diagram of the inverter REF-22K-GPD-INV-EASY3B is shown in Figure 1. The board consists of five boards: power board, EMI filter, high-voltage logic board, low-voltage interface board, and the DC-link board which are mounted in one housing. For more details see Section 5.

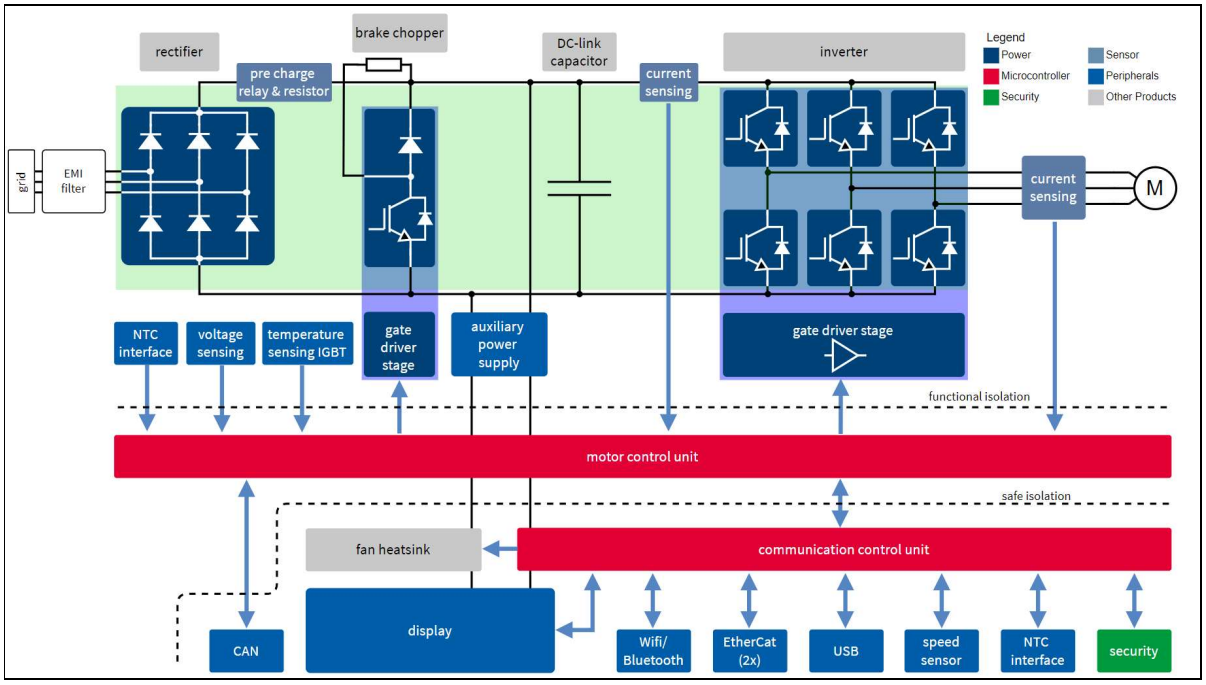


Figure 1 Block diagram of the inverter

The reference kit REF-22K-GPD-INV-EASY3B is a industrial drive inverter which can be connected to a three-phase AC input. The reference kit includes an input EMI-filter. The AC voltage is rectified via the uncontrolled diode full bridge. The inrush current is limited by pre-charge circuitry. The rectified AC voltage is stabilized by a DC-link capacitor bank. The IGBT six pack allows for the modulation of a three-phase output voltage/current which can be varied in terms of its amplitude and frequency. This three-phase voltage is used for controlling the speed and torque of the motor. During braking or deceleration of the motor, energy is transferred back into the DC-link capacitor increasing the DC bus voltage. Therefore, a brake chopper is included that absorbs this energy by switching an external brake resistor across the DC bus capacitors. The

The reference board at a glance

The Infineon EasyPIM™ 3B power module, FP100R12W3T7_B11, includes the three-phase input rectifier, the six pack as well as the brake chopper. All IGBTs are driven by the Infineon EiceDRIVER™ 1ED3240MC12H. The isolation coordination details of the unit can be found in Section 5.1.2.

The current measurement of the three-phase output current is done with TLI4971-A120T5 current sensors. The current signals are used for the motor control; additionally, these sensors are used to detect an external short circuit for the IGBT's. To protect the module against an internal short circuit, a fourth current sensor is integrated between the DC-link capacitor and the IGBT module.

The auxiliary voltage for the inverter is generated by a DC/DC switch mode power supply (SMPS) converter. The CoolSiC™-MOSFET IMBF170R1K0M1 is used in a flyback topology. More details about the flyback SMPS are listed in Section 5.1.5

The reference inverter uses two microcontrollers; one for control and one for communication. For control the XMC4800-F144F2048 is used, for communication the XMC4300-F100K256 AA is designed in. Both microcontrollers communicate via an UART interface. The communication microcontroller allows the inverter to be controlled via a touch-screen or a PC-GUI.

A simplified block diagram is shown in Figure 2. This block diagram shows the main components of the boards, the DC/DC SMPS converter, the low-voltage interface board (SELV board), high-voltage logic board (FELV board), the IGBT module incl. gate driver ICs, and the current measurement of the AC output. The main Infineon components used in the specific blocks are listed in the diagram.

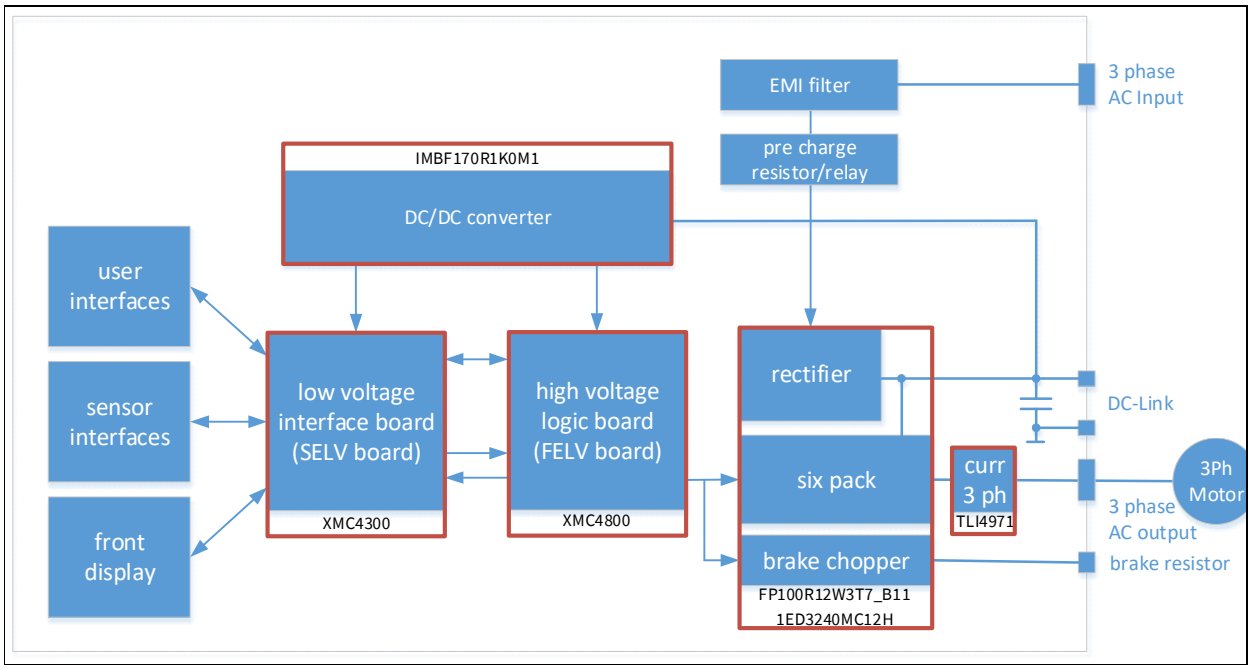


Figure 2 Simplified block diagram of the inverter

1.3 Main features

- FP100R12W3T7_B11: EASY3B IGBT7 module for high-current and high-power density [3]
- TLI4971-A120T5: XENSIV™ current sensor for measuring high currents with minimal power loss [4]
- 1ED3240MC12H: Gate driver for optimal EMI performance and reduced power losses [5]
- IMBF170R1K0M1: Infineon CoolSiC™ MOSFET 1700 V enables direct drive by most flyback controllers [6]
- XMC4800-F144F2048 and XMC4300-F100K256 AA: XMC microcontrollers for inverter control and communication [7]

1.4 Board parameters and technical data

The key parameters of the REF-22K-GPD-INV-EASY3B are shown in Table 2.

Table 2 Parameters of REF-22K-GPD-INV-EASY3B

Parameter	Symbol	Conditions	Value	Unit
Input line voltage	V_{in}	Three-phase AC ± 10%	380 … 480	V
Input frequency	f_{in}	± 3 Hz	50 … 60	Hz
Output voltage	V_{out}	three-phase AC	0 V … 0.95 x input voltage	V
Output frequency	f_{out}		0 … 599	Hz
Switching frequency of motor output	f_{sw}	Factory setting 4 kHz	4	kHz
Rated power	P_{rated}		22	kW
Rated current	I_{rated}		45	A
Low overload - base load power	P_{LO}	T_{amb} <= 35°C	22	kW
Low overload - base load current	I_{LO}		45	A
High overload - base load power	P_{HO}		18.5	kW
High overload - base load current	I_{HO}		38	A
Power losses	P_{loss}		< 700	W
Weight	m_{inv}		< 10	kg
Ambient temperature	T_{amb}	Relative humidity RH; 30% < RH< 80%	10 to 35	°C

1.4.1 Overload profile normal duty

The normal duty (ND) load cycle assumes a uniform base load with low requirements during short acceleration times at high torque; see Figure 3. Typical applications when designing according to normal duty include:

- Pumps, fans and compressors
- Wet or dry blasting technology
- Mills, mixers, kneaders, crushers, agitators
- Basic spindles
- Rotary kilns
- Extruders

Typical converter load cycle:

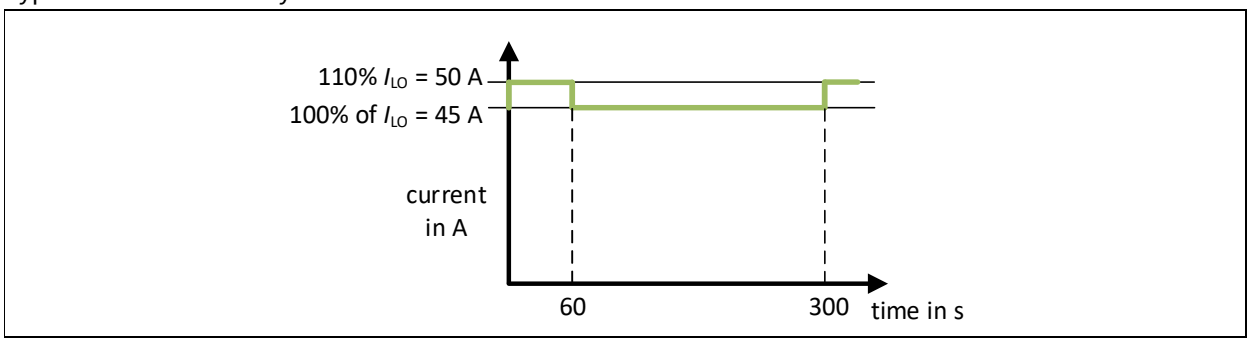


Figure 3 Overload profile normal duty

1.4.2 Heavy duty overload profile

The heavy duty (HD) load cycle permits dynamic accelerating phases at a reduced base load; see Figure 4. Typical applications when designing according to heavy duty include:

- Horizontal and vertical conveyor technology (conveyor belts, roller conveyors, chain conveyors)
- Centrifuges
- Escalators/moving stairways
- Lifters
- Elevators
- Gantry cranes
- Storage and retrieval machines

Typical converter load cycle:

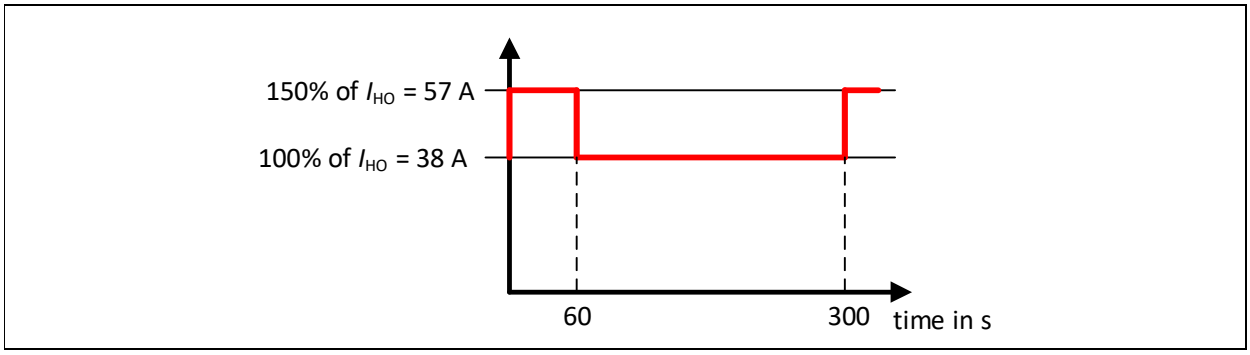


Figure 4 Overload profile heavy duty

1.5 Advantages of two level slew rate gate drivers

The dvCE/dt level of an IGBT turn-on event is usually higher at low temperatures and low currents. This means that if the dvCE/dt level is to be limited, the value of the gate resistor needs to be increased. Such limitations of dvCE/dt are common in motor drives, where the motor current also contains zero crossings. That is why the gate resistance should be dimensioned at 0 for example, or perhaps at 1/10 of the rated current of the power module at room temperature. It is known that high dvCE/dt has detrimental effects on motors. These effects include voltage doubling at the motor windings with longer motor cables and high-step voltage changes across the stator end-windings. [8]

While allowing the dvCE/dt levels to be capped, the higher gate resistor value causes higher switching losses, and thus, reduces the efficiency of the converter. It also causes higher junction temperatures, reduces the lifetime of the power modules, and increases cooling requirements.

If, however, two gate resistors can be used, higher gate resistor values can be applied for turn-on at low currents and low temperatures, thus limiting dvCE/dt levels. Also, lower gate resistor values can be used for turn-on at higher currents and temperatures to minimize losses.

On the other hand, the inductive turn-off overshoot may violate the power transistor's breakdown voltage Vbr. The turn-off at short circuit or overcurrent is especially critical. It is possible to mitigate this risk by increasing the gate resistance for turn-off. This results in higher switching losses, which in turn increases junction temperatures and cooling requirements.

Figure 5 shows a schematic diagram for the proposed gate drive circuit using the 2L-SRC driver IC 1ED3240MC12H. The driver IC has a single control input for fast or slow switching (/INF), and a time in parallel, so that the IGBT gate is driven by the correlated combination of R1, R2, R3, and R4 in parallel. During slow operation, the IGBT is driven only by a combination of R1 and R1 in parallel with R2.

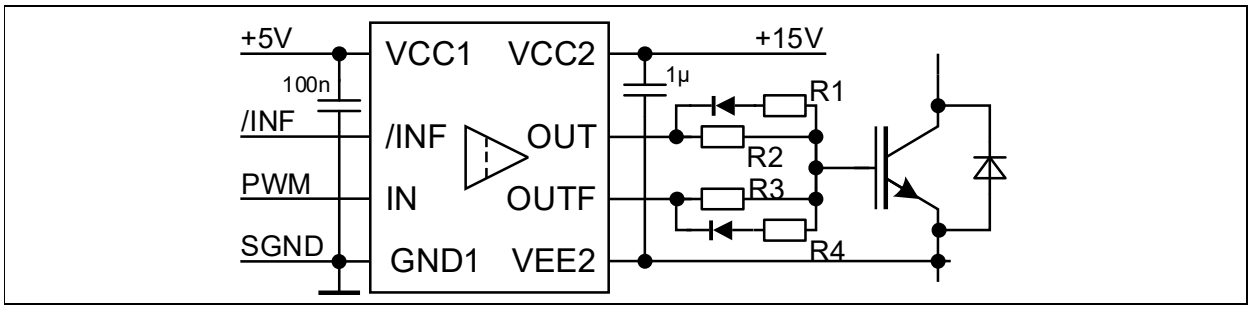


Figure 5 Schematic diagram of the proposed dual-gate resistor gate-drive circuit using 1ED3240MC12H

(IN) for fast operation, both output terminals OUT and OUTF switch at the same time in parallel, so that the IGBT gate is driven by the correlated combination of R1, R2, R3, and R4 in parallel. During slow operation, the IGBT is driven only by a combination of R1 and R1 in parallel with R2.

2 System design

2.1 Schematics

The schematics of the design are available via Infineon.com. Keep in mind that the inverter consists of five boards, hence you will find one schematic for each board.

Table 3 Schematic overview

Board name	File name in zip folder
Power board	22kW_Inverter_Power_Board_02.11.2022.pdf
DC-link board	U109_GPD_DCLinkCaps_01.11.2022.pdf
EMI filter	U109_22kWGPD_EMI-filter_02.11.2022.pdf
High voltage logic board	22kW_Inverter_Logic_02.11.2022.pdf
Low voltage interface	22kW_Inverter_Interfaces_02.11.2022.pdf
Connection between high voltage logic board and low voltage Interface	U109_GPD_FlexPCBConnector_02.11.2022.pdf
Isolation sheet between power board and heat sink	U109_GPD_HeatsinkIsolation_01.11.2022.pdf
Gasket to separate clean and dirty air	U109_GPD_Sealing_01.11.2022.pdf

The Altium™ project files are available on request.

2.2 Layout

2.2.1 Power board

The layout of the power board is shown in Figure 6; in Table 4 you will find the names of each block.

The AC input connector (1), AC output connector (2), the brake resistor connector (3) as well as the DC-link connections (4) are located on the left hand side of the power board. The EMC filter is connected to the board in the area marked (5). The high-side gate drivers are located in the area marked (6) and the low-side gate driver are marked in section (12).

The interface to the high-voltage logic board (FELV board) is marked (7); this board has the interface connectors. The output phase current measurement is marked (8); all three-phase currents are measured, also the current between the DC-link capacitor and IGBT module is measured.

The IGBT module will be pressed in the area marked (9). The auxiliary power supply is located in the area marked (10), the SiC MOSFET is soldered to the PCB in the area (11). The DC-link board is screwed to the power board in the area marked (12).

The interface to the low-voltage interface board (SELV board) is marked (13). The safe isolation communication between the high-voltage logic board (FELV board) and low-voltage interface board (SELV board) is realized with the IC located on the PCB in area (14). The position of the pre-charge relay is shown at (16) and the pre-charge resistors are shown at (17).

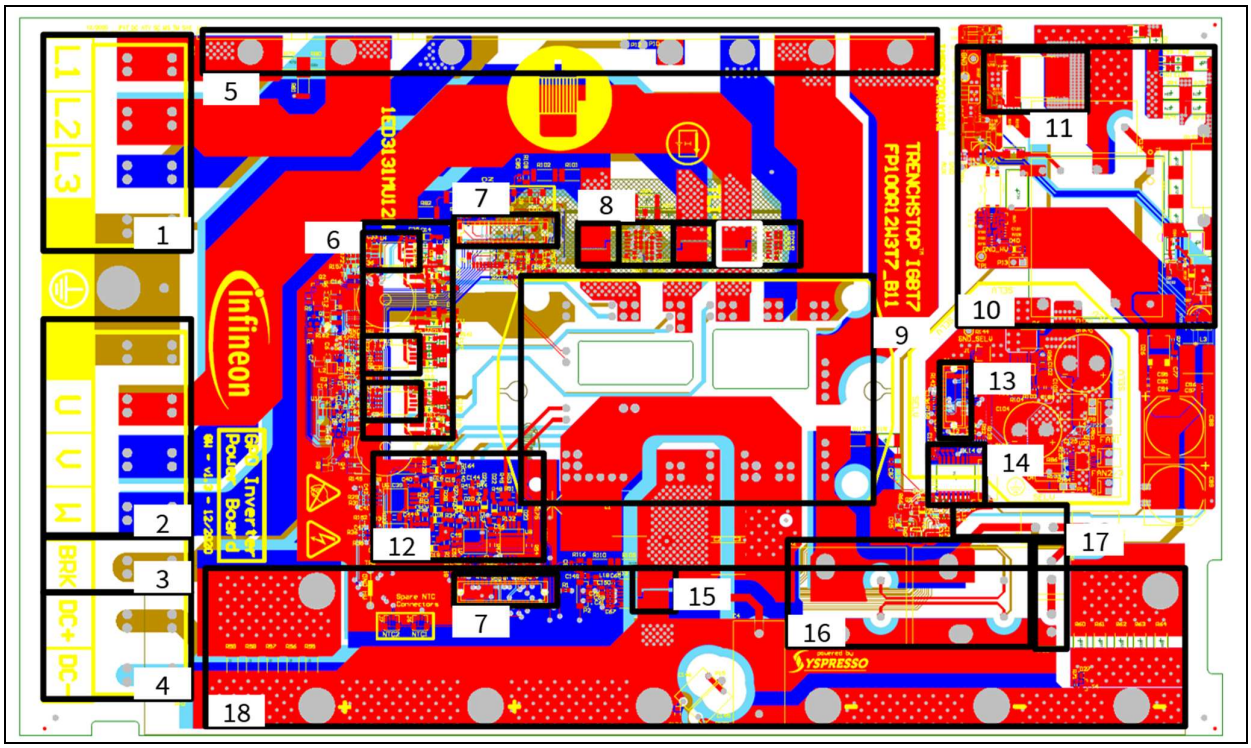


Figure 6 Layout of the power board

Table 4 Functional block - power board layout

Block number	Function
1	AC input connector
2	AC output connector
3	Brake resistor connector
4	DC-link connector
5	EMI-filter connections
6	High-side gate driver with gate driver IC 1ED3240MC12H
7	Connector for motor control board
8	Output phase current sensor measurement with TLI4971-A120T5
9	IGBT module with IGBT7 - FP100R12W3T7_B11
10	Auxiliary power supply
11	CoolSiC™ MOSFET IMBF170R1K0M1 of the auxiliary power supply
12	Low-side gate driver with gate driver IC 1ED3131MC12H
13	Connector for communication interface board
14	Isolation barrier for communication between high-voltage logic board (FELV board) and low voltage interface board (SELV board)
15	DC-link current sensor TLI4971-A120T5
16	Pre-charge relay
17	Pre-charge resistors
18	DC-Link

System design

The gate layout of the high-side IGBT of the phase U is shown as an example in Figure 7. The traces of the emitter and gate trace are on top of each other, which minimizes any parasitic inductances.

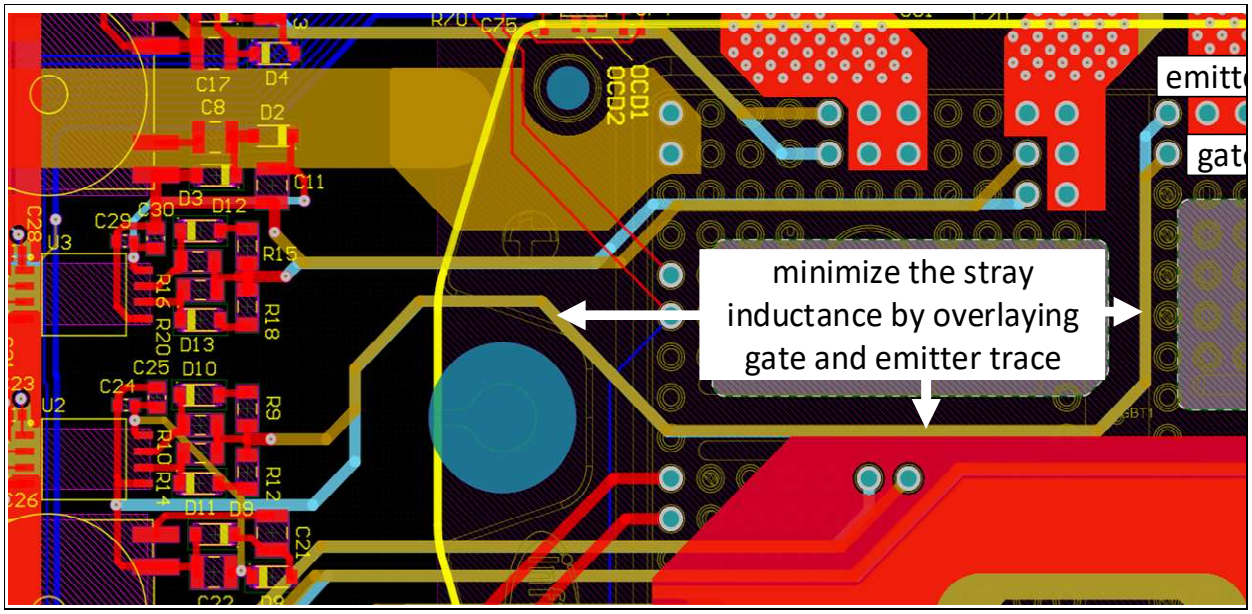


Figure 7 Gate layout of the high-side IGBT of phase U

2.3 Inverter cooling concept

The main cooling of the IGBT is done by a heat sink and an integrated 92 mm fan with a temperature controlled flowrate of 0-270 m^3/h. The air is drawn in the bottom of the unit and blown up through the heat sink to the outlet at the top of the enclosure, assuming a normal wall mounting position.
A 52 mm auxiliary fan cools the DC-link capacitors and operates at a constant speed with a flow rate of 20m^3/h. The air is blown into an air channel at the bottom of the capacitor array. To provide an improved cooling over the entire surface of each capacitor the air is directed up one side and then channeled via small slits between each capacitor and then finally along the side of the heat sink to the air outlet at the top of the enclosure.
A second 52 mm fan, also operating at a constant speed, cools a third channel containing the EMI filter. The air flows beside the heat sink and over the indictors and capacitors of the EMI filter. This cooling channel has a separate air outlet at the top of the enclosure.

2.4 Bill of material

The complete bill of material is available on the download section of the Infineon homepage. A log-in is required to download this material.

Table 5 BOM of the most important/critical parts of the evaluation or reference board (example)

Board name	File name in zip folder
Power board	BOM_22kW_Inverter_Power_Board_v1.3_01.11.2022.xlsx
DC-link board	BOM_U109_GPD_DCLinkCaps_v1.0_01.11.2022.xlsx
EMI filter	BOM_U109_22kWGPD_EMI-filter_V2_v1.0_01.11.2022.xlsx
High voltage logic board	BOM_22kW_Inverter_Logic_v1.2_01.11.2022.xlsx
Low voltage interface	BOM_U109_22kW_Inverter_Interfaces_v1.2_01.11.2022.xlsx
Connection between high voltage logic board and low voltage Interface	BOM_U109_GPD_FlexPCBConnector_v1.0_01.11.2022.xlsx

2.5 Connector details

Connectors of the power board:

Table 6 Connector PH1

PIN	Label	Function
1	Earth	Earth potential for safe operation always connect to earth
2	L3	Line feeder cable phase L3
3	L2	Line feeder cable phase L2
4	L1	Line feeder cable phase L1

Table 7 Connector PH2

PIN	Label	Function
1	Earth	Earth potential for safe operation always connect to earth
2	DC-	DC-minus voltage
3	DC+	DC-plus voltage, the brake resistor must be connected here
4	brake	The brake resistor must be connected here, it is connect between DC+ and brake
5	W	Motor feeder cable phase W
6	V	Motor feeder cable phase V
7	U	Motor feeder cable phase U

System design

Connectors of the user interface board:

Table 8 Connector P1

PIN	Label	Function
1	SPEED_1	Input signal speed sensor
2	V9_SELV	9 V power supply

Table 9 Connector P2

PIN	Label	Function
1	Input	Input signal NTC
2	GND	Ground

Table 10 Connector P4

PIN	Label	Function
1	VCC	Supply voltage
2	D-	Data signal -
3	D+	Data signal +
4	GND	Ground

The USB port for remote control via the PC-GUI can be found at the front side of the inverter.

Table 11 Connector P14

PIN	Label	Function
1	VCC	Supply voltage
2	VCC	Supply voltage
3	GND	Ground
4	STOA	Safe torque off Channel A
5	STOB	Safe torque off Channel B
6	Chassis_GND	Chassis GND

System design

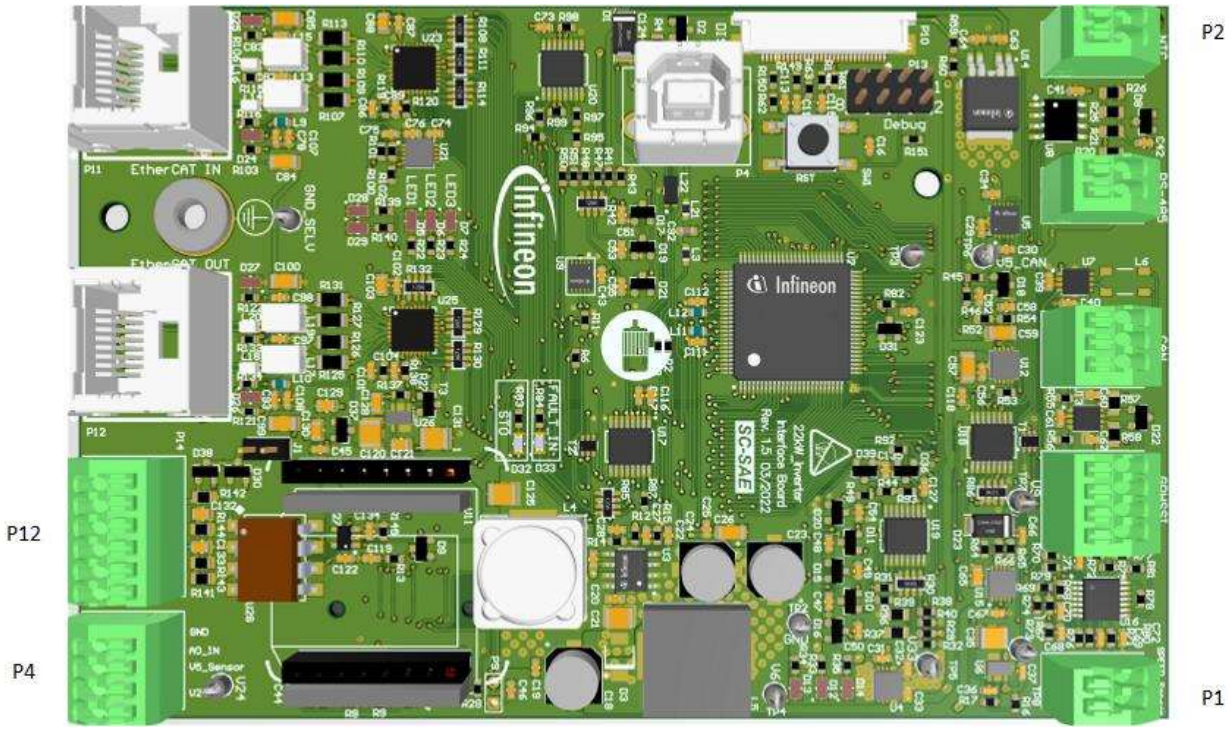


Figure 8 Interfaceboard

3 Software structure

3.1 Overview

The system depicted can be simplified to the structure in Figure 9 from the perspective of the interacting software applications. The system contains two central processing units (CPUs) which both are connected to each other and to a computer.
The R-CPU (real-time CPU) executes the control of the connected power electronic setup. It is connected to C-CPU (communication CPU) to exchange parameters and to respond to commands from the user. Additionally, it is connected to the computer to support a live view of internal quantities of the software for debugging purposes.
The C-CPU connects the user interface running on the computer to the R-CPU handling the control. It provides access to the parameter database for the user and the R-CPU.
The computer is used to execute the graphical user interface (GUI) provided to control and parameterize the connected system. It can optionally run the software oscilloscope XScope to allow inspection of internal quantities of the R-CPU software.
The software structure of both CPUs and the involved software components for communication are described in more detail in the subsequent sections.

The USB connection between R-CPU and computer needs a isolation ! (see Section 4.1.7)

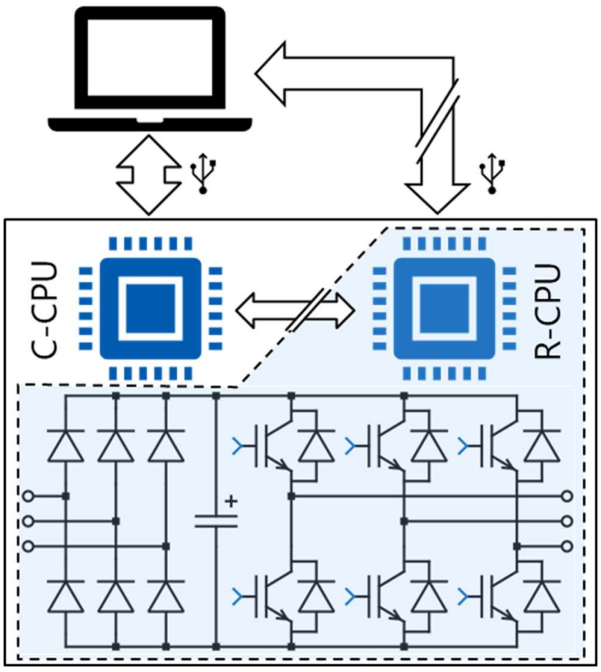


Figure 9 System abstraction focusing on communication

3.2 R-CPU Software

The structure of the software for the R-CPU is depicted in Figure 10 as a layered structure. The software is divided into three main parts or domains. The *Real-time Domain* takes care of all relevant functions that require real-time performance such as closed-loop control or system protection. The *Background Domain* handles all other functions like communication or the main state machine of the system. These layers are implemented ideally as completely abstract and platform independent code.

Software structure

The third layer therefore provides an abstract application programming interface (API) to the other layers and implements the platform specific code. This *Hardware Abstraction* layer is mostly provided by the toolchain DAVE.
The connection between the layers is achieved by program flow, by signals and by parameters.
Program flow defines which component executes what part of the software at a given time. Execution is passed at program start to the initialization code which organizes how the runtime is distributed between the components. Mostly tasks of a real-time operating system (RTOS) and interrupts (IRQs) are used to structure the available runtime.
Signals are implemented as a globally available structure of variables which can be accessed by all code parts and reflect the overall state of the software and are thus volatile in nature.
Parameters on the other hand are used to store quantities which are non-volatile and parameterize the behavior of the software.
The subsequent sections give more detail for each layer of the software and the folder structure of the code base.

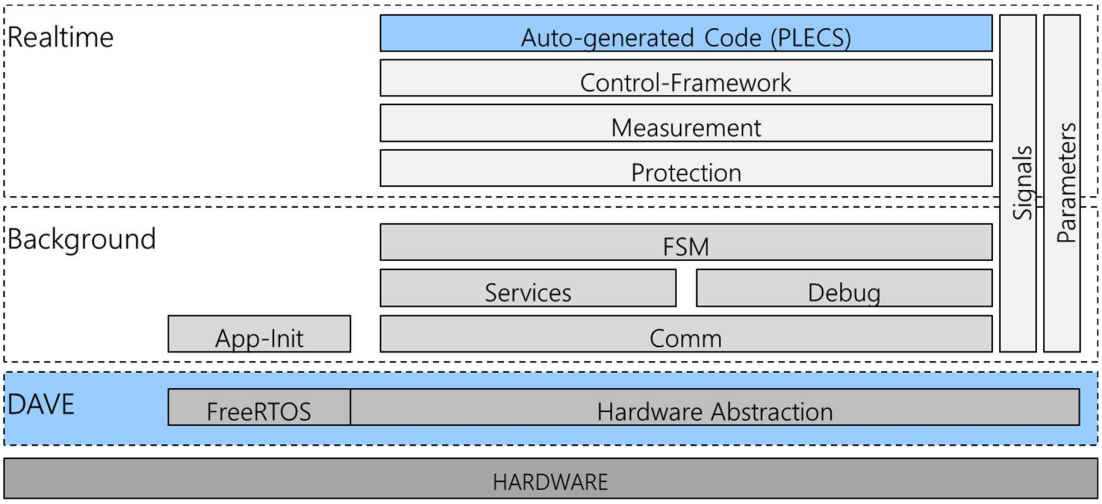


Figure 10 Structural representation of the implemented software

3.2.1 Real-time Domain

The *Real-time Domain* contains all components which are required for the real-time functions of the software. The main functions are being realized by model-based development within the simulation tool PLECS and exported as generated C code. The code generation is controlled by the target support package *GPDTarget* within PLECS which connects the code to the control framework of the software structure. This framework integrates the generated code into the overall code structure in terms of program flow and signal flow.
Figure 11 depicts the program flow of the *Real-time Domain*. The swim lanes describe which part of the real-time domain is responsible for the processing of the involved steps.
The real-time behavior is entirely realized within an interrupt handler which is triggered by the signal generation of the hardware platform, e.g. the space vector modulator (SVM) or – used here – the analog-to-digital conversion (ADC). The execution rate is an integer divider of the SVM carrier frequency. The IRQ handler therefore is the first part of the code that is being executed and starts the processing. The step *Measurement & Conditioning* acquires and pre-processes inputs required for *Protection* and *PLECS model*, mostly analog and digital inputs to the hardware. The step *Protection* realizes the safe operation of the system by checking measurements and internal quantities against trip levels. The control code is executed as part of the step *PLECS Model* and yields output signals which are then being distributed to the respective destinations, mainly the SVM generator. Part of the *Protection* is to supervise the calculation time of the real-time IRQ.

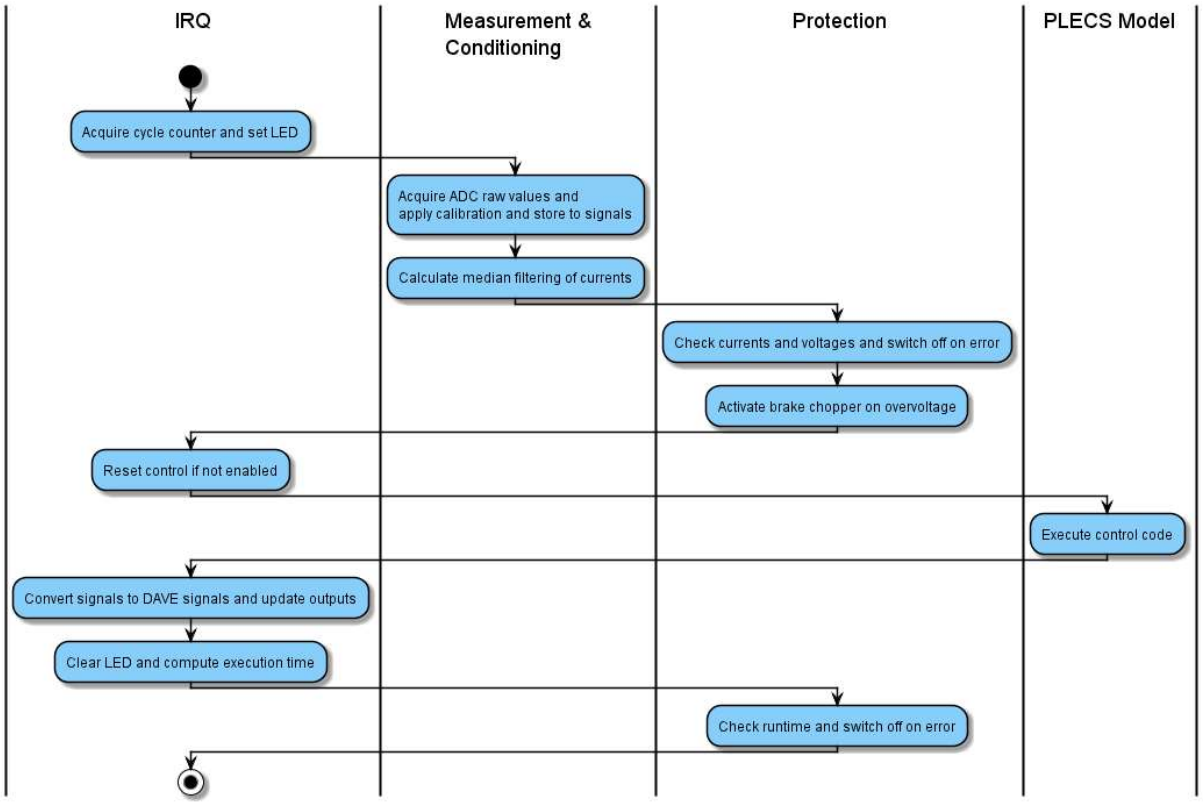


Figure 11 Program flow of the real-time control framework

The following sections describe details of the structural components as shown in Figure 11.

3.2.1.1 Auto-generated Code

The auto-generated code is compiled as part of the complete project. It consists of one main code block (.h/.c file) representing the top-level PLECS model, possibly additional model files of subcomponents, and additional files which make sure that the model-based code can be seamlessly integrated into the framework. Table 12 lists the files of the sensorless Field Oriented Control (FOC) as an example for the generated files.

Table 12 Auto-generated files of the GPDTarget

File	Description
VfControl (.h/.c), …	Main file of the control model; name and content are model specific
parameterentry.h	Declaration of the C struct of a parameter entry
parameterlist (.h/.c)	List of parameters which can be manipulated; This list is being sent to the C-CPU and made available in the parameter database.
realtime (.h/.c)	Main interface and entry point of control flow; Name and content are model-agnostic and offer a fixed interface to the remaining code.
debugsignallist (.h/.c)	Signals made available by the Debug Blocks of the model; These signals are available within the XScope software scope.

3.2.1.2 Control-Framework

The purpose of the *Control Framework* is to connect the auto-generated code of the PLECS model to the other software components. It contains the code which sets up the real-time IRQ handler based on the ADC module, the handler itself and other required functions.

Software structure

The main interfaces to the auto-generated code of the PLECS model are program flow, signals and parameters. The *GPDTarget* itself is able to directly generate access to the global *Signals* and to provide a list of *Parameters*. The program flow, however, is explicitly provided by the integration of function calls to the IRQ handler as part of the *Control Framework*.

3.2.1.3 Measurement and Conditioning

The component *Measurement* is responsible for acquiring all required digital and analog input signals of the system. Upon execution, it reads all inputs from the hardware via the *Hardware Abstraction layer (DAVE)* and applies a two-point calibration in form of a gain and an offset parameter to convert the raw signals from the analog-to-digital conversion to SI-based quantities like currents in Ampere or voltages in Volt.
Optionally, the component *Conditioning* provides filters to incoming signals such as low-pass filtering of noisy measurements.

Table 13 List of measurements

Input	Signal name	Description
	Signals.Measurement.Iph_U	Pseudo-differentially(1) determined phase current in Ampere.
	Signals.Measurement.Iph_V	Pseudo-differentially(1) determined phase current in Ampere.
	Signals.Measurement.Iph_W	Current of phase W is determined by Kirchhoff's law: IW=-IU-IV.
	Signals.Measurement.IDC	Pseudo-differentially(1) determined DC-link current in Ampere.
	Signals.Measurement.VDC	DC-link voltage in Volts.
	Signals.Measurement.VACR	Voltage after Full Bridge Rectifier (before Relay) in Volts.
	Signals.Measurement.VOUT_U	Phase voltage of the AC output in Volts. This is a slow(3) measurement.
	Signals.Measurement.VOUT_V	Phase voltage of the AC output in Volts.
	Signals.Measurement.VOUT_W	Phase voltage of the AC output in Volts.
	Signals.Measurement.offsetI	Average of the three reference voltages of the phase current sensors.
	Signals.Measurement.NTC1	Placeholder(2) for NTC1 temperature reading.
	Signals.Measurement.NTC2	Temperature measurement in °C of sensor NTC2. Gain and offset calculated at approx. 25°C and 80 °C. This is a slow(3) measurement.
	Signals.Measurement.NTC3	Temperature measurement in °C of sensor NTC3. Gain and offset calculated at approx. 25 °C and 80 °C. This is a slow(3) measurement.
	Signals.Measurement.NTC_cool	Placeholder(2) for NTC_cool temperature reading.
	Signals.Measurement.V15	Measurement of the 15 V supply rail in Volts. This is a slow(3) measurement.
	Signals.Measurement.V5	Placeholder(2) for 5 V supply rail in Volts.
	Signals.Measurement.NTC_ambient	Temperature measurement in °C of sensor NTC_ambient. Gain and offset calculated at approx. 25 °C and 80 °C. This is a slow(3) measurement.

	Signals.Measurement.NTC_dirty	Temperature measurement in °C of sensor NTC_dirty for hardware revision 0 or gate driver supply readback in Volts for hardware revision 2.
	Signals.Measurement.NTC_IGBT	Temperature measurement in °C of sensor NTC_IGBT. Gain and offset calculated at approx. 80 °C and 120 °C. This is a slow(3) measurement. A low-pass filter is being applied to this value.
	Signals.Measurement.Fil_Iph_U	Median3-filtered version of the unfiltered, calibrated signal.
	Signals.Measurement.Fil_Iph_V	Median3-filtered version of the unfiltered, calibrated signal.
	Signals.Measurement.Fil_Iph_W	Median3-filtered version of the unfiltered, calibrated signal.
	Signals.Measurement.Fil_IDC	Median3-filtered version of the unfiltered, calibrated signal.
	Signals.Measurement.Fil_VDC	Median3-filtered version of the unfiltered, calibrated signal.
	Signals.Measurement.Fil_VACR	Median3-filtered version of the unfiltered, calibrated signal.

(1) Pseudo-differential means: both terminal voltages (current signal and reference voltage) of the current sensor are acquired and the difference is being calculated in software.
(2) Not all physical sensors are available in the software due to ADC sampling restrictions of the underlying low-level driver and peripheral units of the controller.
(3) Due to the sampling structure of the peripheral modules, some values are not sampled for each control event.

3.2.1.4 Protection

The component *Protection* provides a safety mechanism (in terms of intrinsic safety, not functional safety) by checking if all relevant quantities are within the specified boundaries. If a quantity crosses a predefined and parameterizable threshold, an error is raised which results in executing an emergency shutoff and storing the cause for the error. Additionally, the main state machine is triggered to change state into the error state. The emergency shutoff forces all power electronic switching signals to their inactive state and opens the precharge relay as fast as possible to protect the system from overvoltages and overcurrents. Table 14 lists all protective measures, the respective signals and parameters.

Table 14 List of protective measures

Measures against	Signal name(s)	Parameter	Display in GUI
IRQ calculation time	Signals.Status. IRQTimeTotal	Parameters.Protection.IRQMax	ADC_BIST
Overcurrent phases	Signals.Measurement. Fil_Iph_{U, V, W}	Parameters.Protection.IOut	offset_Isensor{U,V, W}
Overcurrent DC link	Signals.Measurement. Fil_IDC	Parameters.Protection.IDC	offset_IsensorDC
Overvoltage DC link	Signals.Measurement. Fil_VDC	Parameters.Protection.VLinkMax	VDCL_OV(1)
Undervoltage DC link	Signals.Measurement. Fil_VDC	Parameters.Protection.VLinkMin	VDCL_UV

Overvoltage Grid	Signals.Measurement.Fil_VACR	Parameters.Protection.VGrid	ERROR(2)
Overvoltage phases	Signals.Measurement.VOUT_{U, V, W}	Parameters.Protection.VOut	ERROR(2)
Unbalance between VGrid and VDC	Signals.Measurement.Fil_VDC, Signals.Measurement.Fil_VACR	Parameters.Protection.VLinkUnbalance	VACR_Diff
Overtemperature(3)	Signals.Measurement.NTC2	Parameters.Protection.NTC2_Trip	TSENS_OOR
Overtemperature(3)	Signals.Measurement.NTC3	Parameters.Protection.NTC3_Trip	TSENS_OOR
Overtemperature(3)	Signals.Measurement.NTC_ambient	Parameters.Protection.NTC_ambient_Trip	TSENS_OOR
Overtemperature(3)	Signals.Measurement.NTC_IGBT	Parameters.Protection.NTC_IGBT_Trip	OVT_IGBT
Overtemperature(3)	Signals.Measurement.NTC_IGBT	Parameters.Protection.NTC_IGBT_Warning	TW_IGBT
Supply failure(3)	Signals.Measurement.V15	Parameters.Protection.V15OV	V15_OV
Supply failure(3)	Signals.Measurement.V15	Parameters.Protection.V15UV	V15_UV
Communication timeout(3)	Signals.Status.UARTPacketTimeout	N/A	ERROR(2)
OCD(4)	N/A	N/A	OCD1

(1) This error activates the brake chopper.
(2) ERROR represents a collective error, i.e. this indicator is active if any error occurs.
(3) These checks are being processed in the *Background Domain*, i.e. in a task.
(4) This check is realized by an interrupt handler, acting upon a digital input.

3.2.1.5 Background Domain

The *Background Domain* contains all code parts which are not time critical. The code is structured in tasks as provided by the underlying real-time operating system (RTOS, here FreeRTOS). This way, a blocking

programming paradigm can be used which leads to small and portable code fragments. The main components of the background processing are described in the subsequent sections.

3.2.1.6 Tasks

Tasks are the main building blocks of the application. Each task handles a specific part of the software. Since tasks can be blocked, the programming paradigm of synchronous operation can easily be used to control all required functions with a small code base. Table 15 lists the tasks of the R-CPU software and explains their purpose.

Table 15 Tasks of the R-CPU software

Name	Description	Priority
StateMachineTask	The statemachine task handles the main control flow on system level. The main states are Initialization, Configuration, Precharge, Active an Error. The state transitions are mainly controlled by events and timing. Events are detected by Boolean flags stored as Signal.	5
UARTInterfaceTask	This task handles high-level functionality of the communication interface. It is implemented as a state machine as well and handles the phases synchronization, parameter exchange, error handling and the actual operation. State transitions are triggered by reception of a data packet or by Boolean flags stored as Signals. Data packet reception is realized by a queue. Data packet transmission is done directly.	4
UARTLowLevelTask	This task takes care of the low-level data reception. Packet parsing and checksum calculation as well as verification is done here. The chosen approach utilizes an IRQ-less implementation where the physical reception rate and the built-in FIFO depth of the hardware unit is carefully balanced to prevent data overflows even without IRQs. Once a data packet is correctly received it is handed over to the UARTInterfaceTask using a queue.	5
XScopeTask	The XScopeTask is used to send the accumulated data of the XScope during control IRQs to the connected computer using the WinUSB driver of the hardware abstraction.	4

(Priority: high numbers mean high priority)

3.2.1.7 Main State Machine

The main state machine determines the behavior of the software and therefore the whole setup on system level.
Usually, after starting from reset, multiple steps are required in a specific order to reach the operational state. Additionally, certain inputs to the system might be required before operation is possible. The state machine handles all this on a structured, much more abstract level than pure program flow does. Figure 12 depicts the main state machine in the R-CPU software and thus describes the high-level software functions. It is mainly controlled by the user via the graphical user interface, sending commands to the C-CPU to forward them finally to the R-CPU and the state machine. Although the states themselves execute on the CPU in a cyclic fashion to allow both asynchronous and synchronous programming paradigms, they usually are written in a way that the program flow stays in the current state until the conditions are met for a transition. These self-transitions are not depicted.
The implemented state machine is rather simple and intended as a framework for future extensions. It operates as shown in Figure 12:
After reset, the state machine starts in the state Initialize where all subcomponents are brought into operational state. Once everything has been set up, the state machine transitions to the state Idle.

Software structure

Idle waits for user interaction. The command CONFIGURE, sent by the GUI, takes the system out of the waiting state and triggers the transition to the state Configure.
In Configure, all parameters are exchanged between the two CPUs. After that, all relevant data is present in the R-CPU to actually proceed to preparing the power electronics by transitioning to the state Precharge.
Precharge takes care of a precharged DC link and controlling the relevant contactors. After the precharging preconditions are met, the system transits to the state Active.
The state Active finally releases the control code from reset and allows switching patterns to be forwarded to the hardware. If requested to stop by the command IDLE, the state machine transits to the state Idle again to wait for the next phase of operation.
Nearly all states transition to the state Error upon erroneous behavior of any supervised subcomponent. Here, the system is being brought into a safe state by disabling switching signals and taking care of the proper position of all contactors.
At any time, the state and error information are being transmitted to the C-CPU and thus to the GUI.

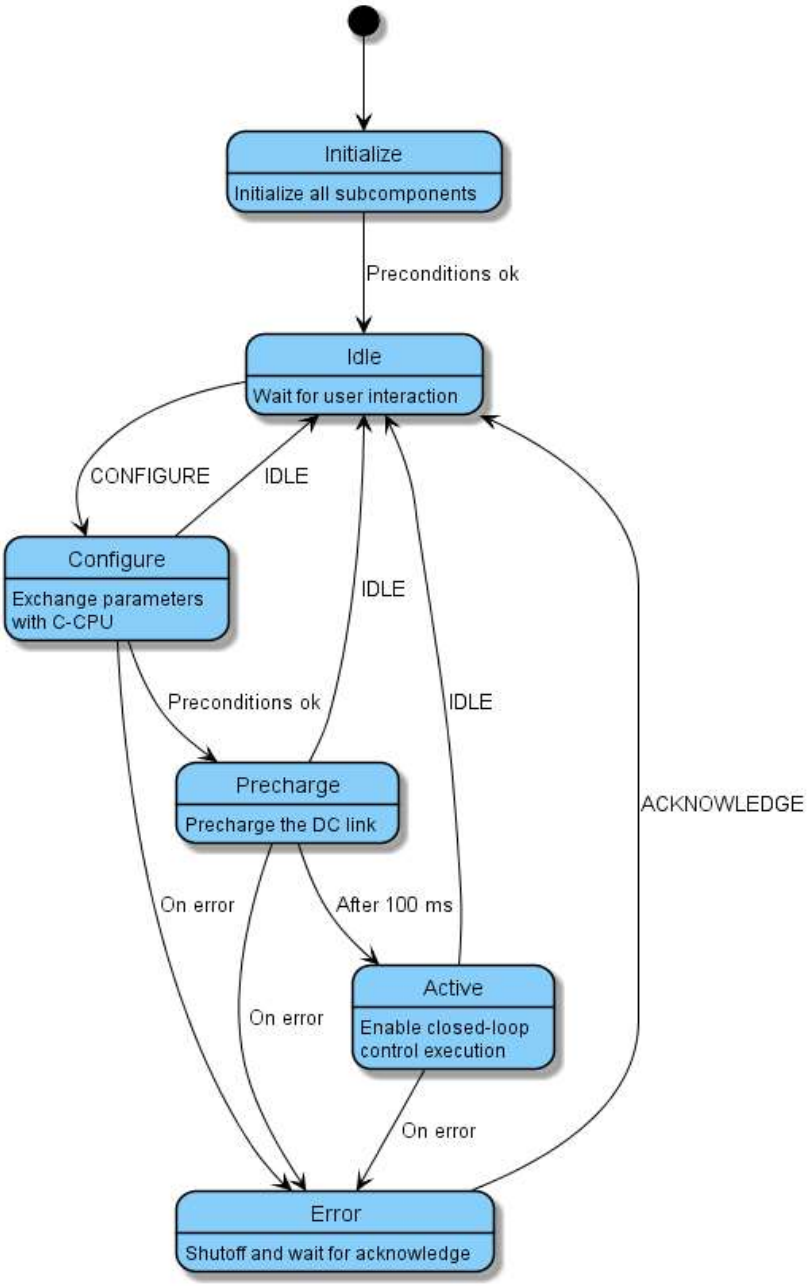


Figure 12 Main state machine

3.2.1.8 UART Communication

Another part of the *Background Domain* are all the parts involved in the communication to the C-CPU. Since the communication channel is used for multiple purposes, a layered and packetized approach has been implemented. There is a low-level layer taking care of the platform specifics of sending and receiving packets as well as checking the packet integrity by checksum verification. A high-level layer provides the actual packet handling.
In this system setup, two independent systems are talking to each other, which might be in a non-synchronized state. This occurs, if the systems are being started at different times (e.g. by restarting the R-CPU after a program download). The high-level layer takes care of this by implementing a state machine of its own which detects and corrects synchronization losses. This high-level layer finally distributed commands and data received and controls data sent back.

3.2.1.9 XScope

The final component present in the *Background Domain* is the *XScope* functionality which enables a live view of internal variables of the software or the model-based components. Two parts work together to allow a non-intrusive data transfer out of the *Real-time Domain.* Within the main program flow of the real-time code, the first part is usually realized as a simple function call and reads a set of channels and transfers the data to a buffer structure with fixed timing (i.e. once every n samples). The second part is implemented as a task and processes a large chunk of sampled channels at once in the *Background Domain* and transfers it to the accompanying *XScope* graphical user interface by whatever data channel is present on the system. The channels to display are either hard-coded or – if appropriately programmed – can be selected by the user at run-time.
Here, the data channel in use is provided by the *Hardware Abstraction* and uses a WinUSB connection over the USB port of the R-CPU.

3.2.2 Hardware Abstraction (DAVE)

The *Hardware Abstraction* is typically realized by low-level code or libraries or more complex frameworks provided by the silicon vendor. Here, DAVE is being used to provide access to the underlying microcontroller hardware. The main parts provided by DAVE are the space vector modulator (SVM) for switching signal and timing event generation, timers to generate events at lower rates, digital inputs and outputs, the analog subsystem to sample analog quantities and the WinUSB code layer as well as the low-level hardware functions.
The important aspect of the Hardware Abstraction is the configuration of the hardware at run-time (opposed to the configuration at compile-time) and the interfacing between the software and hardware components during control operation. Since part of the data for that comes from the model-based code, interfacing functions have been added to connect DAVE-based code to the automatically generated code from model-based development. The relevant files are listed in Section 0.

3.2.3 Signals and Parameters

The signals of the R-CPU software are listed in Table 16 for reference. The index of each entry is used to configure the *XScope* channels, see Section 3.2.1.9

Table 16 Signals of the R-CPU software

Index	Signal	Type	Description
100	Signals.Status.Command	uint32	Command for the main state machine
101	Signals.Status.UartCommand	uint32	Command for the inter-processor communication
102	Signals.Status.UARTPacketTimeout	bool	Boolean indicator for a packet timeout

103	Signals.Status.UARTErrorCode	uint32	Error code: 0: no error. See UARTHighLevelTask.c
104	Signals.Status.UARTPacketLengthErrors	uint32	Counter for packets with wrong or exceeding length
105	Signals.Status.UARTPacketENDErrors	uint32	Counter for enqueueing errors
106	Signals.Status.CurrentState	uint32	Current state of the state machine, see Table 11
107	Signals.Status.Prech_stat	uint32	Boolean state of the precharge contactor
108	Signals.Status.Brake_stat	uint32	Boolean state of the activation of the brake chopper
109	Signals.Status.Output_stat	uint32	Not used
110	Signals.Status.IRQTimeTotal	float	Total IRQ calculation time as float in µs
111	Signals.Status.IRQTimeBetween	float	Time between two consecutive control IRQ calls
112	Signals.Status.BrakePWM	float	PWM dutycycle for the brake chopper PWM
113	Signals.Status.OverVoltage	bool	Boolean state of the overvoltage detection
114	Signals.Status.RealtimeError	bool	Emergency switch-off active (protection)
115	Signals.Status.EnablePWM	bool	Boolean PWM enable from state machine to IRQ
116	Signals.Status.EnableINF	Bool	Not used
117	Signals.Status.SELV_Sync	Bool	Boolean state of UART synchronization
118	Signals.Status.PARAM_Sync	Bool	Boolean state of parameter synchronization
0	Signals.Measurement.Iph_U	float	Measurement of the phase current U
1	Signals.Measurement.Iph_V	float	Measurement of the phase current V
2	Signals.Measurement.Iph_W	float	Measurement of the phase current W
3	Signals.Measurement.IDC	float	Measurement of DC link current
4	Signals.Measurement.VDC	float	Measurement of DC voltage
5	Signals.Measurement.VACR	float	Measurement of AC voltage
6	Signals.Measurement.VOUT_U	float	Measurement of the phase voltage U
7	Signals.Measurement.VOUT_V	float	Measurement of the phase voltage V
8	Signals.Measurement.VOUT_W	float	Measurement of the phase voltage W
9	Signals.Measurement.offsetI	float	Average offset voltage of the current measurement
10	Signals.Measurement.NTC1	float	Measurement of temperature of NTC1

11	Signals.Measurement.NTC2	float	Measurement of temperature of NTC2
12	Signals.Measurement.NTC3	float	Measurement of temperature of NTC3
13	Signals.Measurement.NTC_cool	float	Measurement of temperature of NTC_cool
14	Signals.Measurement.V15	float	Measurement of the 15 V supply
15	Signals.Measurement.V5	float	Measurement of the 5 V supply
16	Signals.Measurement.NTC_ambient	float	Measurement of temperature of NTC_ambient
17	Signals.Measurement.NTC_dirty	float	Measurement of temperature of NTC_dirty
18	Signals.Measurement.NTC_IGBT	float	Measurement of temperature of NTC_IGBT
19	Signals.Measurement.AINSpare	float	Measurement of temperature of AINSpare
20	Signals.Measurement.DACSpare	float	Measurement of temperature of DACSpare
21	Signals.Measurement.Fil_Iph_U	float	Median-filtered current of phase U
22	Signals.Measurement.Fil_Iph_V	float	Median-filtered current of phase V
23	Signals.Measurement.Fil_Iph_W	float	Median-filtered current of phase U
24	Signals.Measurement.Fil_IDC	float	Median-filtered DC current
25	Signals.Measurement.Fil_VDC	float	Median-filtered DC voltage
26	Signals.Measurement.Fil_VACR	float	Median-filtered AC voltage
200	Signals.Control.Enable	bool	Command for the main state machine
201	Signals.Control.ControlFrequency	float	Command for the inter-processor communication
N/A	Signals.Control.SampleTimeInNanoSeconds	float	Boolean indicator for a packet timeout
202	Signals.Control.Setpoint	float	Error code: 0: no error. See UARTHighLevelTask.c
203	Signals.Control.CurrentValue	float	Counter for packets with wrong or exceeding length
204	Signals.Control.AmplitudeClamp	float	Counter for enqueueing errors
205	Signals.Control.Amplitude	float	Current state of the state machine, see Table 11
206	Signals.Control.Angle	float	Boolean state of the precharge contactor
207	Signals.Control.svmAmplitude	uint32	Boolean state of the activation of the brake chopper
208	Signals.Control.svmAngle	uint32	Not used
N/A	Signals.Control.PARAM_Sync	bool	Total IRQ calculation time as float in µs

300	Signals.Protection.all	uint32	Collection of all error bits, see GUI
301	Signals.Protection_Latch.all	uint32	Latched version of error bits, gets reset by GUI
N/A	Signals.LiveView.ChannelSet	float	Unused
N/A	Signals.LiveView.ChannelIndex[0]	float	Index XScope channel 1, see Section 4.2.8
N/A	Signals.LiveView.ChannelIndex[1]	float	Index XScope channel 2, see Section 4.2.8
N/A	Signals.LiveView.ChannelIndex[2]	float	Index XScope channel 3, see Section 4.2.8
N/A	Signals.LiveView.ChannelIndex[3]	float	Index XScope channel 4, see Section 4.2.8
N/A	Signals.LiveView.ChannelIndex[4]	float	Index XScope channel 5, see Section 4.2.8
N/A	Signals.LiveView.ChannelIndex[5]	float	Index XScope channel 6, see Section 4.2.8
N/A	Signals.LiveView.ChannelIndex[6]	float	Index XScope channel 7, see Section 4.2.8
N/A	Signals.LiveView.ChannelIndex[7]	float	Index XScope channel 8, see Section 4.2.8
N/A	Signals.LiveView.ChannelIndex[8]	float	Index XScope channel 9, see Section 4.2.8

The parameters of the R-CPU software are listed inTable 17. The first column contains the name of the parameter as it appears in the *Parameter Database*. Additionally, Table 18 lists entries in the *Parameter Database* which give access to signals instead of parameters.

ParamDB name	Parameter	Type	Description
N/A	Parameters.Control.InitializationSuccessful	uint32	Not used
Output.PhaseCurrent.SlewRateFast	Parameters.Control.SlewRateFast	float	Hysteresis threshold, fast slope activated if current larger
Output.PhaseCurrent.SlewRateSlow	Parameters.Control.SlewRateSlow	float	Hysteresis threshold, slow slope activated if current smaller
Input.VLink.Unbalance.Trip	Parameters.Protection.VLinkUnbalance	float	Trip level for the absolute difference of the median-filtered DC link and VAC input voltages
Output.PhaseVoltage.Unbalance.Trip	Parameters.Protection.VOutUnbalance	float	Unused, can be used in PLECS model
Output.PhaseCurrent.Unbalance.Trip	Parameters.Protection.IOutUnbalance	float	Unused, can be used in PLECS model
Input.VLink.BrakeChopperTrigger	Parameters.Protection.VLinkNom	float	Unused, can be used in PLECS model
Input.VLink.TripMax	Parameters.Protection.VLinkMax	float	Trip level for the median-filtered DC link overvoltage
Input.VLink.TripMin	Parameters.Protection.VLinkMin	float	Trip level for the median-filtered DC link undervoltage
N/A	Parameters.Protection.IGrid	float	Unused, can be used in PLECS model
Input.VGrid.TripMax	Parameters.Protection.VGrid	float	Trip level for the median-filtered VAC
Input.DCCurrent.Trip	Parameters.Protection.IDC	float	Trip level for the median-filtered IDC
Output.PhaseCurrent.Trip	Parameters.Protection.IOut	float	Trip level for the absolute median-filtered phase current
Output.PhaseVoltage.Trip	Parameters.Protection.VOut	float	Trip level for the median-filtered phase voltages
N/A	Parameters.Protection.V5OV	float	Unused
N/A	Parameters.Protection.V5UV	float	Unused
N/A	Parameters.Protection.V15OV	float	Trip level for overvoltage protection of the 15 V supply
N/A	Parameters.Protection.V15UV	float	Trip level for undervoltage protection of the 15 V supply

N/A	Parameters.Protection.Offset_TLI4971_max	float	Error threshold for phase current offset in state Configure
N/A	Parameters.Protection.NTC1_Trip	float	Unused
N/A	Parameters.Protection.NTC2_Trip	float	Overtemperature protection threshold
N/A	Parameters.Protection.NTC3_Trip	float	Overtemperature protection threshold
N/A	Parameters.Protection.NTC_cool_Trip	float	Unused
N/A	Parameters.Protection.NTC_ambient_Trip	float	Overtemperature protection threshold
N/A	Parameters.Protection.NTC_dirty_Trip	float	Overtemperature protection threshold
N/A	Parameters.Protection.NTC_IGBT_Trip	float	Overtemperature protection threshold
N/A	Parameters.Protection.NTC_IGBT_Warning	float	Overtemperature warning threshold
N/A	Parameters.Protection.Process_Load_Trip	float	Trip level for IRQ calculation time (0 – 100, 100 = 100%)
N/A	Parameters.Protection.IRQMax	float	Trip level in µs for IRQ calculation time, automatically calculated
N/A	Parameters.Protection.IRQSend	float	Unused
N/A	Parameters.Protection.IGridUnbalance	float	Unused
N/A	Parameters.Protection.TIGridUnbalance	float	Unused
Output.Calibration.IPhaseU.Gain	Parameters.Calibration.gain_U	float	Gain of two-point calibration(1)
Output.Calibration.IPhaseU.Offset	Parameters.Calibration.offset_U	float	Offset of two-point calibration(1)
Output.Calibration.IPhaseV.Gain	Parameters.Calibration.gain_V	float	Gain of two-point calibration(1)
Output.Calibration.IPhaseV.Offset	Parameters.Calibration.offset_V	float	Offset of two-point calibration(1)
Output.Calibration.IPhaseW.Gain	Parameters.Calibration.gain_W	float	Gain of two-point calibration(1)
Output.Calibration.IPhaseW.Offset	Parameters.Calibration.offset_W	float	Offset of two-point calibration(1)
Output.Calibration.IDC.Gain	Parameters.Calibration.gain_IDC	float	Gain of two-point calibration(1)
Output.Calibration.IDC.Offset	Parameters.Calibration.offset_IDC	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_V_ACR	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_V_ACR	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_VOUT_U	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_VOUT_U	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_VOUT_V	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_VOUT_V	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_VOUT_W	float	Gain of two-point calibration(1)

N/A	Parameters.Calibration.offset_VOUT_W	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_VDC_LINK	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_VDC_LINK	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_NTC_1	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_NTC_1	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_NTC_2	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_NTC_2	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_NTC_3	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_NTC_3	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_NTC_COOL	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_NTC_COOL	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_NTC_AMB	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_NTC_AMB	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_GD_SUP_FB	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_GD_SUP_FB	float	Offset of two-point calibration(1)
Output.Calibration.NTC.Gain	Parameters.Calibration.gain_NTC_IGBT	float	Gain of two-point calibration(1)
Output.Calibration.NTC.Offset	Parameters.Calibration.offset_NTC_IGBT	float	Offset of two-point calibration(1)
N/A	Parameters.Calibration.gain_NTC_SPARE	float	Gain of two-point calibration(1)
N/A	Parameters.Calibration.offset_NTC_SPARE	float	Offset of two-point calibration(1)
Operation.Debug.Undersampling	Parameters.Debug.Undersampling	float	Execute XScope sampling once every Undersampling steps

(1) Two-point calibration works like this: calibrated_value = gain * raw_value + offset

Table 17 Parameters of the R-CPU

ParamDB name	Signals	Type	Description
Operation.LiveView.ChannelSet	Signals.LiveView.ChannelSet	float	Chooses a predefined channel set – not used
Operation.LiveView.Channel1	Signals.LiveView.ChannelIndex[0]	float	Signal index of XScope channel 1, see 4.2.8
Operation.LiveView.Channel2	Signals.LiveView.ChannelIndex[1]	float	Signal index of XScope channel 2, see 4.2.8
Operation.LiveView.Channel3	Signals.LiveView.ChannelIndex[2]	float	Signal index of XScope channel 3, see 4.2.8
Operation.LiveView.Channel4	Signals.LiveView.ChannelIndex[3]	float	Signal index of XScope channel 4, see 4.2.8
Operation.LiveView.Channel5	Signals.LiveView.ChannelIndex[4]	float	Signal index of XScope channel 5, see 4.2.8
Operation.LiveView.Channel6	Signals.LiveView.ChannelIndex[5]	float	Signal index of XScope channel 6, see 4.2.8
Operation.LiveView.Channel7	Signals.LiveView.ChannelIndex[6]	float	Signal index of XScope channel 7, see 4.2.8
Operation.LiveView.Channel8	Signals.LiveView.ChannelIndex[7]	float	Signal index of XScope channel 8, see 4.2.8
Operation.LiveView.Channel9	Signals.LiveView.ChannelIndex[8]	float	Signal index of XScope channel 9, see 4.2.8

Table 18 Signals accessible via the Parameter Database

3.2.4 Folder and file reference

The directory tree in Figure 13 shows the folder structure of the R-CPU software. On the top level, these folders contain the main contributions to the code:

- Files related to the Hardware Abstraction (DAVE) are located in the subfolder DAVE and are automatically maintained and generated by DAVE
- The Debug subfolder holds files related to building the software. Most of the files are automatically generated temporary files that are not required for permanent storage. Some files, however, are required by the build-system and therefore are included in the software package. Those files should be added to a version control system while the automatically created ones might be added to ignore lists.
- The Libraries subfolder holds all required libraries as defined by the DAVE build tool chain.
- The PLECS subfolder is designed to receive the auto-generated files by PLECS within the AutoCode subfolder. It is mandatory to place the PLECS-generated files there because of a directive in the linker definition file, which places all code within the AutoCode subfolder in a specific section in RAM for faster execution.
- The source subfolder contains the actual code of the project. According to the general structure of the code, there are subfolders for the Background Domain, for the Realtime Domain and for platform-specific code.
- The subfolder Startup contains specific code to bring the CPU to operating state. This code usually is auto-generated by DAVE but moved to this location for clarity.

```
├──Dave
│      ...
├──Debug
│   └──PLECS
│       └──AutoCode
├──Libraries
│      ...
├──PLECS
│   └──AutoCode
├──source
│   ├──Background
│   │   ├──Communication
│   │   ├──Parameters
│   │   ├──StateMachine
│   │   └──XScope
│   ├──Platform
│   └──RealTime
└──Startup
```

Figure 13 Folder structure of the R-CPU source code

Some of the most important files of the project are listed in Table 19:

File	Description
.cproject	DAVE / Eclipse file containing C/C++ Development Tooling information, maintained by DAVE
.project	General DAVE / Eclipse project configuration
linker_script.ld	Compiler specific linker file which defines where code and data is to be placed in the memory map of the CPU. This file contains project-specific changes

	to the DAVE template that are required to execute the PLECS generated code from RAM for faster execution
source/main.c	Main entry point for program flow
source/Version.h	Defines the software version
source/Signals.h	Declares the signals of the project
source/Global.h	Defines global settings of the project
source/Background/Parmeteters/Parameters.h	Declares the parameters of the project

Table 19 Selected files of the R-CPU project

3.3 Inter-Processor Communication

This section describes the concept and the actual communication scheme of the inter-processor communication between the R-CPU and the C-CPU.
The communication infrastructure is based on the following assumptions:

- The transmission channel might be disturbed during operation of the device.
- Only rather slow communication is required, no live data.
- The data exchange is point-to-point only.

The usual approach in such environments is a packetized data exchange operated by a layered software stack (see ISO/OSI reference model).

3.3.1 Packet Structure and Code Layers

The packetized data exchange uses a simple packet structure which is depicted inFigure 14.

0	1	2	3	4		n-3	n-2	n-1
START		CMD	LEN	PAYLOAD	…	PAYLOAD	CRC	END

Figure 14 Packet structure

A packet is a sequence of bytes with a fixed structure. The actual packet content is preceded by a two-byte START delimiter (0xefbe). The first actual message byte is the command byte CMD. It is followed by a length byte LEN, which defines the actual length of the following payload in bytes. The PAYLOAD is transmitted next and consists of a variable number of bytes as specified by the LEN field in the packet. The maximum payload size is 255 bytes. The payload is followed by a checksum CRC of one byte with the polynomial 0x31, a start value of 0xff and a final XOR of 0x00. The final byte is the END delimiter (0x0d).
In this project, a universal asynchronous receiver transmitter (UART) interface is being used. The UART operates at 115200 baud with 8 bits, no parity and one stop bit.
Raw packages are handled by the low-level code. When transmitting, a direct memory access (DMA) channel transfers the data without CPU interference. For data reception, a mode called direct mode has been chosen which makes use of the hardware-provided first-in-first-out (FIFO) buffer. The depth of the FIFO of 32 bytes allows autonomous reception of data at maximum baud rate without interrupting the CPU for more than 2 ms before data loss. The reception is handled by a task with a wake-up time of 1 ms, thus guaranteeing data consistency.
Once a packet is decoded and its checksum has been verified, the low-level code hands the packet content over to the high-level code by means of a queue, which effectively decouples the low-level and high-level processing of packets. The high-level code is realized as a task which blocks on the queue for received packets. A timeout is used on the blocking call to provide error handling and a reaction to loss of synchronization. The high-level functionality of the inter-process-communication is implemented as a finite state machine to provide a structured data exchange which requires that both sides of the communication channel stay synchronized.

3.3.2 State Machine

The Inter-processor communication requires for the C-CPU and the R-CPU to be synchronized to work properly. One example use case demonstrates that: usually during development, the C-CPU continues to operate whereas the R-CPU is being reset after a software update. In this case, the C-CPU has to cease normal operation and prepare for the R-CPU to reconnect and conduct its initialization steps. This is being handled by a finite state machine on the R-CPU which is depicted in Figure 15.
After Program start, the state machine initializes all subcomponents and readies the low-level parts. Upon completion, it transits to state Synchronization.
In state Synchronization, there is a handshake that informs the C-CPU of the beginning of operations. The R-CPU sends a 'hello' packet and transmits its software version for the C-CPU to register. The state machine transitions to State SendOwnParameter next.

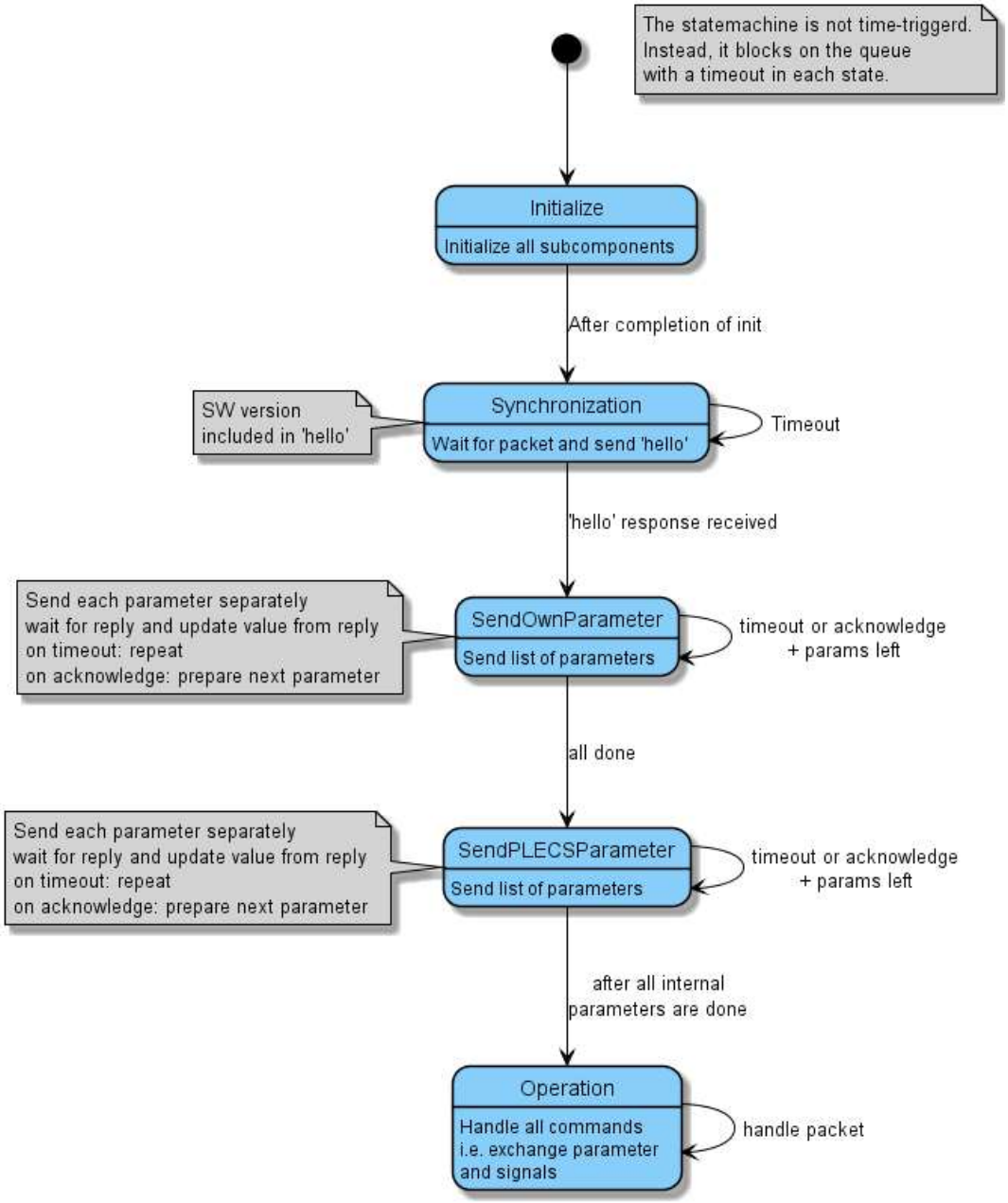


Figure 15 State machine of the inter-processor communication

During SendOwnParameter, the state machine walks through its list of internal parameters and exchanges them with the C-CPU, thus receiving pre-configured values out of the parameter database. After completion, execution is being continued in state SentPLECSParameter.
SendPLECSParameter performs the same operation for each parameter but now with the parameters defined by the model-based software layer. If all parameters are synchronized completely, the state machine transits to state Operation.
State Operation finally allows data exchange between the R-CPU and the C-CPU during operation of the device. Since all parameters are synchronized, the remaining tasks of the communication layer is to exchange operational data such as measurement and status information, to provide setpoint values to the control and to start or stop the device.
To simplify software integration with the pre-existing structure of the C-CPU, the commands and data structures for measurement and status information have been reused. Not all commands are implemented on the R-CPU. Table 20 lists and explains the commands and the data to transfer.

3.3.3 Command Reference

The following table lists the command used by the R-CPU software.

Command	Byte	Description	Data
CONNECT	0x02	Message used for synchronization handshake	Concatenation of a detection pattern and the version string
READ_STAT	0x05	Sent regularly by the C-CPU to update the status of the R-CPU in the GUI. Reply uses the same command	Current and latched error bits, amplitude of the SVM in percent, setpoint and current value and the state of the system
READ_ANALOG	0x06	Sent regularly by the C-CPU to update the measurements of the R-CPU in the GUI. Reply uses the same command	Measurements of phase currents, DC current, DC voltage, AC voltage, phase voltages, temperatures and supply voltages
CONTROL_INV	0x07	Sent by the C-CPU if the user requested a start or stop	None
SET_SPEED	0x08	Sent by the C-CPU if the user entered a setpoint	Float value of the setpoint
OK	0x0b	Used to acknowledge an operation	None
RESET	0x0c	Sent by the C-CPU if the user requested the acknowledge of an error	None
READ_PARAM	0x0e	Sent by the R-CPU initially to request a parameter value. Sent by the C-CPU in reply containing the value of an existing parameter	Parameter ID, value and name
NEW_PARAM	0x0f	Sent by the C-CPU in reply to a READ_PARAM command from the R-CPU if the requested parameter is a new one	Parameter ID, value and name
SYNC_PARAM	0x10	Sent by the C-CPU if a re-sync of parameters has been requested	None
SET_PARAM	0x11	Sent by the C-CPU if a parameter has been updated by the user	ID and value of the parameter

Table 20 Inter-processor communication commands

3.4 C-CPU Software

The main purpose of the C-CPU software is to manage communication between user and R-CPU and to organize and store parameters required by the software of the R-CPU. In terms of interfaces, this requires one or more communication channels between a graphical user interface (GUI) and the C-CPU, a communication channel between the CPUs (UART here, see Section 3.3) and a database to store parameters permanently. All operations are the result of user interaction and therefore considered non-critical in terms of timing. The structure of the C-CPU software therefore slightly differs from the R-CPU software, the most prominent difference is the lack of the *Real-time Domain*. Large parts of the code, mainly for interaction between C-CPU and the graphical user interface via USB, have been taken from the previous code base. The parameter handling and the communication structure between the CPUs are the main add-ons to the code base.

3.4.1 Structure

Since no *Real-time Domain* is required, all functionality is being done in the *Background Domain*. The main structural elements of program flow are tasks as provided by the real-time operating system (RTOS). Here, the built-in RTOS of the development environment DAVE is being used. Note that the terms task and thread are used as synonyms here.
Since no complex functions are required in the C-CPU, a simple setup-and-loop sequence is implemented for the functions. The communication interfaces and the background housekeeping are processed concurrently by tasks. Table 21 describes the tasks present in the software.

Name	Description
UARTInterfaceTask	This task handles high-level functionality of the communication interface. It is implemented as a state machine and handles the phases synchronization, parameter exchange, error handling and the actual operation. State transitions are triggered by reception of a data packet or by Boolean flags stored as Signals. Data packet reception is realized by a queue. Data packet transmission is done directly. See Section 3.3.2 for details.
UARTLowLevelTask	This task takes care of the low-level data reception. Packet parsing and checksum calculation as well as verification is done here. The chosen approach utilizes an IRQ-less implementation where the physical reception rate and the built-in FIFO depth of the hardware unit is carefully balanced to prevent data overflows even without IRQs. Once a data packet is correctly received it is handed over to the UARTInterfaceTask using a queue. See Section 3.3.1 for details.
USBThread	The task USBThread is used to receive data from the WinUSB driver. Once a packet has been received, it is being forwarded to the USBCommandHandler task for further processing via queue.
USBCommandHandler	The USBCommandHandler task processes incoming packets from the user interface and either processes them directly or forwards them to the R-CPU via the UART packetized interface.
Monitor	The monitor task sets up and monitors all auxiliary functions which are controlled by the C-CPU such as supply supervision and fan control.

Table 21 Tasks of the C-CPU software

3.4.2 Parameter Database

The Parameter Database is one of the central features of the C-CPU software.
Features of the Parameter Database are:

- Hierarchical structure of parameters, reflected in parameter naming
- Random access to all parameters during runtime based on name, ID or by tree-walking
- Binary representation for all parameters suitable for storage to and retrieval from non-volatile memories
- Guaranteed data integrity verification by checksums during von-volatile storage
- Simple editing of parameters by non-software people
- Run-time get and set operations for arbitrary parameters
- Mandatory ranges for parameter values
- Additional information such as units, a description and other flags as properties for each parameter
- Run-time insertion and deletion of parameters

One design goal of integrating the parameter database into the C-CPU has been to provide a software for the C-CPU that is compiled once and does not need to be changed if the set of parameters changes, i.e. the R-CPU software might add or remove parameters but no change and thus no recompilation is required for the C-CPU.
The basis for the features is a database of all parameters at runtime in the RAM of the C-CPU. This database is being initialized by either the set of parameters linked into the program of the C-CPU or a valid data structure in a non-volatile memory of the C-CPU (here: EEPROM emulation in flash). These parameters are being transferred to the GUI for the user to inspect and manipulate and are being synchronized with the R-CPU upon request. The linked-in parameter set is intended as a default set and as initial setup of the non-volatile data structure. The binary representation of the parameter structure is being built by an external tool out of a human-editable file.
Throughout the parameter database, the key to each parameter is its name (or rather a hash thereof) which basically represents a path to the parameter. Since all parameters are organized hierarchically, the path represents the sub-tree in which the parameter is located. The parameter name is structured by an optional sequence of sub-tree names and a mandatory final parameter name, all separated by a colon ('.'). For example, a parameter might be called Output.Calibration.IPhaseV.Gain. The parameters are mainly accessed inside the software by their names. Additionally, each parameter may have a unique ID as property which allows a mapping of parameters to a numerical representation.
Once loaded into memory, the C-CPU gives access to the parameter database to the graphical user interface and to the R-CPU. While the former only allows inspection and manipulation of parameters, the latter requires the addition or deletion of parameters at runtime. Therefore, the R-CPU parameters are prefixed by "Coder.". The C-CPU stores all new parameters encountered during parameter synchronization (see Section 3.3.2) within this sub-tree. If after successful synchronization any parameter left in this sub-tree has not been requested by the R-CPU, the parameter is considered obsolete and can be deleted. That way, the parameter database always is a direct representation of the parameters required by the current version of the R-CPU.
The parameter database functionality is included into the software project as a ready-to-use library with C binding. Since the library itself has been developed in C++, a C++ standard library has to be added to the linker flags.

3.4.3 Folder and file reference

The folder structure shows similarities to the one of the R-CPU. Due to differences in the domain setup (i.e. the missing Real-time Domain) and the addition of the Parameter Database, however, there are additional folders and the main source folder is structured differently.Figure 13 shows the top-level folders.
The sub-tree externals contains all software modules added to the C-CPU software, here only the Parameter Database libraries and the C wrapper files.
The sub-folder Parameters contains the files and tools to build the initial set of parameters linked into the executable to setup the database. The main file is Parameters.csv which is human readable and editable and contains all presently defined parameters. The ones coming from the R-CPU and the PLECS model are being added during run-time (see Section 3.4.2).
The sub-tree source contains tasks as the main structural elements, bundles global elements in a sub-folder and contains additional background code as well as the version information file.

4 System and functional description

4.1 Usage

This section describes the installation and usage of the software package and all tools. Please follow the installation and setup instructions carefully as some steps rely on earlier ones.

4.1.1 Prerequisites and Installation

The following tools and packages have to be installed for full usability of the software package:

DAVE	Development environment, compiler and linker
PLECS and license	Simulation and model-based software development
GPDTarget	PLECS target support package for code generation
C-CPU and R-CPU applications	DAVE embedded software projects for both processors on the GPD
XMC™ Link debug probe and J-Link software	Hardware probe and tools for debugging and flashing

The process for installation and setup is described in the subsequent sections. Optionally, the Visual Studio C# development environment may be installed but is only required to change and compile the GUI.

4.1.2 Installing the GPD software package

The software package for the GPD system contains the following folders and components:

C-CPU	DAVE source code project for the C-CPU
FOC	PLECS model containing the sensorless field-oriented control and a testbench
GPDTarget	PLECS target support package for the GPD target
GridSimulation	A simple simulation of the GPD system to study reactions to different grid situations
GUI	Visual Studio C# source code project for the graphical user interface for the GPD system
R-CPU	DAVE source code project for the R-CPU, framework code to embed and execute the model-based control code contained in the folder FOC
XScope	Software oscilloscope application for live-viewing of R-CPU data

To install the components, copy all folders to your computer. Keep a copy of the original files.

NOTE: Write down the location for later reference.

4.1.3 Installing PLECS

To install PLECS, follow the procedure described in the documentation available from the official web site: https://www.plexim.com/. In short, download and install the PLECS package, add your license file and start the application to test proper setup.

NOTE: A license for PLECS Blockset or PLECS Standalone *and* a license for PLECS Coder is required.

4.1.4 Installing the J-Link drivers for flashing

To be able to flash application binaries to the GPD system, a debug probe and the associated software drivers and tools are required. The probe supported by this software package has to be compatible with the J-Link tools from Segger (https://www.segger.com).
The toolchain has been tested with the XMC Link package from Infineon. Search for “Infineon XMC Link” for ordering information and documentation.

The installation process is documented online and involves downloading and installing the J-Link software package and connecting the debug probe to the processors.

NOTE: Write down the installation path of the J-Link tools during the installation. The path is required in a later installation step.

4.1.5 Installing DAVE

The DAVE software package consists of the compiler and linker for the XMC series processors being used in the GPD system as well as an integrated development environment. The tool can be used as stand-alone editing and compilation tool for software development or just as a command line compiler and linker, triggered by PLECS. The required steps of preparation involve the installation of DAVE itself and the import and compilation of the application projects. Even if PLECS is used to trigger the compilation of the R-CPU application later, the software needs to be built at least once from within DAVE.

4.1.5.1 Installation

The installation involves downloading the DAVE software package as offered by Infineon. After downloading, the contents need to be extracted into an appropriate location with proper permissions for the current user, e.g. `D:\Program Files\DAVE`.

NOTE: Write down the location for later reference.

After extraction, run the main executable to start DAVE as per the installation instructions and proceed with the next step:

4.1.5.2 Compiling the R-CPU software package

The R-CPU software has to be compiled once to be usable by PLECS. To accomplish this, create an empty workspace folder for the R-CPU code. Open the project R-CPU via the menu “File” → “Open Projects from File System”, choose the folder of the R-CPU and import the project. Now, switch the project to Debug and compile it, e.g. by hitting CTRL-B.
Optionally, the functionality of the J-Link debug probe can be tested by flashing the binary into the target by starting a debug session (e.g. by clicking the bug symbol). Please refer to Section 4.1.4 for the physical connection to the target.

NOTE: It might be necessary to check and adapt the debug configuration regarding the J-Link variables, pointing to the J-Link tools.

4.1.5.3 Compiling the C-CPU software package

Similar to the R-CPU software, create an empty workspace for the C-CPU.

NOTE: Mixing of workspaces is not recommended.

Import the project like above via the menu “File” → “Open projects from File System” and compile the project. Flash the software to the C-CPU as described earlier.

4.1.6 Installing the WinUSB drivers

WinUSB drivers are required to connect a Windows PC to the GPD system. The embedded processors implement a USB device which enumerates as a WinUSB device. Once installed, the drivers offer the required interface for the GPD GUI to connect to the C-CPU and the *XScope* software to connect to the R-CPU.
To install the driver, locate it either in your DAVE installation (e.g. in the subfolder `D_LibraryStore_4.4\resources\4.0.10\app\USBD_WINUSB\0\Templates\inf\`) or by

downloading the most recent version from Infineon. Unzip the contents and locate the .inf file. Install the driver by right-clicking the .inf file and choosing Install.
To test the connection, connect a properly flashed C-CPU to the computer and verify the connection in the Windows Device Manager.

4.1.7 Setting up the XScope software oscilloscope

No installation is required for the *XScope* software oscilloscope. To test it, simply start the already copied program. You may reject the Windows firewall popup question, as TCP/IP functionality is not required by this instance of *XScope*.
To setup the software, click on "New project" in the main view and set a project name. Afterwards, choose WinUSB and enter the GUID 40EEA1EB-EF67-4D6A-AB0B-8BB588598704 into the appropriate edit box.

DANGER: The USB connection to the R-CPU has to be galvanically isolated externally! The R-CPU is connected to the DC-link voltage. Use a USB isolator suited for your application and target voltage!

Once connected to the USB port of the R-CPU, click scan and choose a port. In the project page, enter the sample rate and a timeout. The FOC control example uses a control frequency of 2 kHz. An undersampling factor of 2 is currently selected as default, leading to an effective sample rate of 1 kHz. Choose 100 ms as timeout. The undersampling factor is a parameter and thus accessible in the *Parameter Database* by the name `Operation.Debug.Undersampling`.

NOTE: The sample rate has to be configured in accordance to the discretization step size and the undersampling settings in the code. If misconfigured, the scope will work but timing information will be wrong.

Now save your project. If already connected vial USB, click on connect on the right-hand side and switch to Live View using the corresponding icon at the top.

NOTE: Custom buttons are not supported.

4.1.8 Setting up the GUI

No installation is required for the graphical user interface that allows user interaction with the software. To test the software, simply start the executable in the subfolder "\U109_GPD_Inverter\bin\Debug". If already connected to the C-CPU via USB, the USB port should be available and a click on Connect should give access to the C-CPU.

4.1.9 Setting up the GPDTarget and PLECS model

Once copied, the GPDTarget and the PLECS model need to be set up.

4.1.9.1 Setting up the GPDTarget

The GPDTarget needs to be set up in PLECS before the model itself can be opened. To do so, start PLECS and go to "File" → "PLECS Preferences…". Switch to the Tab "Coder" and click "Change". Now choose the parent folder of the target location of your GPDTarget. A click on "Rescan" should show the GPDTarget in the "Installed targets" list.

4.1.9.2 Setting up the PLECS model

Now, the PLECS model FOC can be opened. Before starting simulation or code generation, the setup may be adapted to the machine it is installed on.
The model is being parameterized by an m-script which is called during initialization of the model right before execution of a simulation run. PLECS uses paths to determine where to look for script files in general.

The one being used here is located in the subfolder Parameters. There are several ways to make the required path known to PLECS:

- Enter the path to the initialization script in the Initialization tab found via the menu "Simulation" → "Simulation parameters…" and in the tab "Initialization"
- By providing the main folder of the PLECS model in the virtual drive S:

The latter method is supported by the batch file `SubstS.bat` which is located in the main folder of the PLECS model. The Windows tool subst.exe is used to map the main folder to the virtual drive `S:`. That way, all model-specific paths can be mapped to the `S:` drive and the initialization script found in "Simulation" → "Simulation parameters …" is preconfigured correctly. This is the recommended method since it greatly simplifies using multiple models with the same PLECS preferences.

NOTE: If substituting a path by the virtual drive `S:` is not possible on the machine, the invocation of the initialization script needs to be adapted in the simulation parameters found in the menu under "Simulation" → "Simulation parameters…"!

Now, start a simulation run from the menu or by hitting CTRL-'T' to check if all works properly.

4.1.9.3 Setting up the code generation and flashing

Once the model is open and simulates properly, the code generator needs to be set up.
Open the Coder Options by pressing CTRL+ALT+'B' or via the menu "Coder" → "Coder Options". For the model *FOC*, the discretization step size is already setup properly but needs to be set up if other models are being used. The Output directory needs to be adapted to your setup. Click on "…" next to the Output directory edit box and choose the subfolder "`PLECS/AutoCode`" of the R-CPU folder.
On the tab "Target", the target "GPD22kW" needs to be selected. In the lower part of the dialog, there are three tabs concerning the GPDTarget. Click on "Compilation" and enter the DAVE installation folder noted earlier into the edit box directly or by clicking "…". Now, click on "Flashing" and enter the path to the GDB client of the DAVE installation as shown to the left of the edit box. Add the path to the J-Link installation folder noted earlier. Back in the tab "Build options" you can choose whether you want to build and flash the application to the R-CPU.
To check the setup, click the "Build" button in the lower right part of the dialog. Rebuilding might take some time. If no error occurs, the system is properly set up.

NOTE: Building from within PLECS requires compiling the R-CPU code at least once in DAVE.

4.2 Graphical Programming and Workflow

This section describes the graphical programming workflow with PLECS and the GPD target library. Graphical programming with PLECS allows the development of control code within a simulation and the export of exactly the same control code to the final target. That way, identical behavior of the simulation and the real-world control is possible. The involved workflow differs only slightly from a simulation-only workflow. The key to this is the GPD target support package and its library. Once integrated as described in Section 4.1.9, target specific blocks can be included into a simulation just like all other blocks of the PLECS library. If setup properly, the library provides glueless integration of control elements while allowing code export to the GPD hardware platform and the R-CPU.

4.2.1 Model configuration

To connect a target block in the control subsystem to the real-world measurement or output, the target block needs to be configured. Double-clicking a block opens its mask which contains explanations and options to properly set up the block.

NOTE: Special care must be taken for the SVM block. Its carrier frequency has to be an integer multiple of the discretization step size chosen for the control subsystem.

To achieve real-world behavior and to generate code, the control subsystem itself needs to be configured as well. By default, PLECS generates an internal model representation which is most efficient for simulation but might be different from the code to be executed on the target due to possible differences in discretization and execution order. To get the same results and to be able to actually generate code, the control block has to be configured to code execution, which automatically enables atomic execution. To configure the control block, right-click on it, choose "Subsystem" → "Execution settings…" and check the box "Enable code generation". Enter the desired discretization step size or an appropriate variable name and click "OK". The box now is being displayed with a bold frame, indicating atomic execution. Now, the simulation behaves exactly like the target system.

WARNING: Pay close attention to configure the carrier frequency of the SVM block to be in accordance to the discretization step size. The easiest way to do so is to use variables which have a fixed relation to each other in the initialization tab of the simulation settings.

NOTE: In general: all target blocks need to be configured properly, see Section 4.2.5.

To complete the development, debugging signals usually need to be inspected during simulation and later during run-time. The target library contains a block to achieve this for simulation as well as on the target. The debugging block can be fed by arbitrary signals within the atomic block. The signals are available outside the block in simulation and can be accessed by *XScope* on the target platform (see Section 4.2.8).

NOTE: Debugging signals need to be copied in each step of the simulation and thus increase computation time of the real-time code.

4.2.2 Coder configuration

Once completely tested in simulation, the control code can be exported. The Coder can be invoked from the menu "Coder" → "Coder Options". Before building code for the first time, there are several settings to be made.

Figure 16 shows the configuration dialog of PLECS. It shows on the left-hand side the subsystem the code is to be generated for. The right-hand side shows the general settings. Make sure the setting of "Discretization step size" fits to the needs of the control and the system. Choose "float" as "Floating point format", since double leads to excessive computation time. Make sure to avoid non-limited integrations and absolute time. Finally, choose the subfolder `PLECS/AutoCode` of the R-CPU installation for "Output directory".

WARNING: The discretization step size determines the control IRQ rate of the real-time system! Make sure to choose a value which is compatible with the PWM frequency configured for the SVM block! See Section 4.2.5.

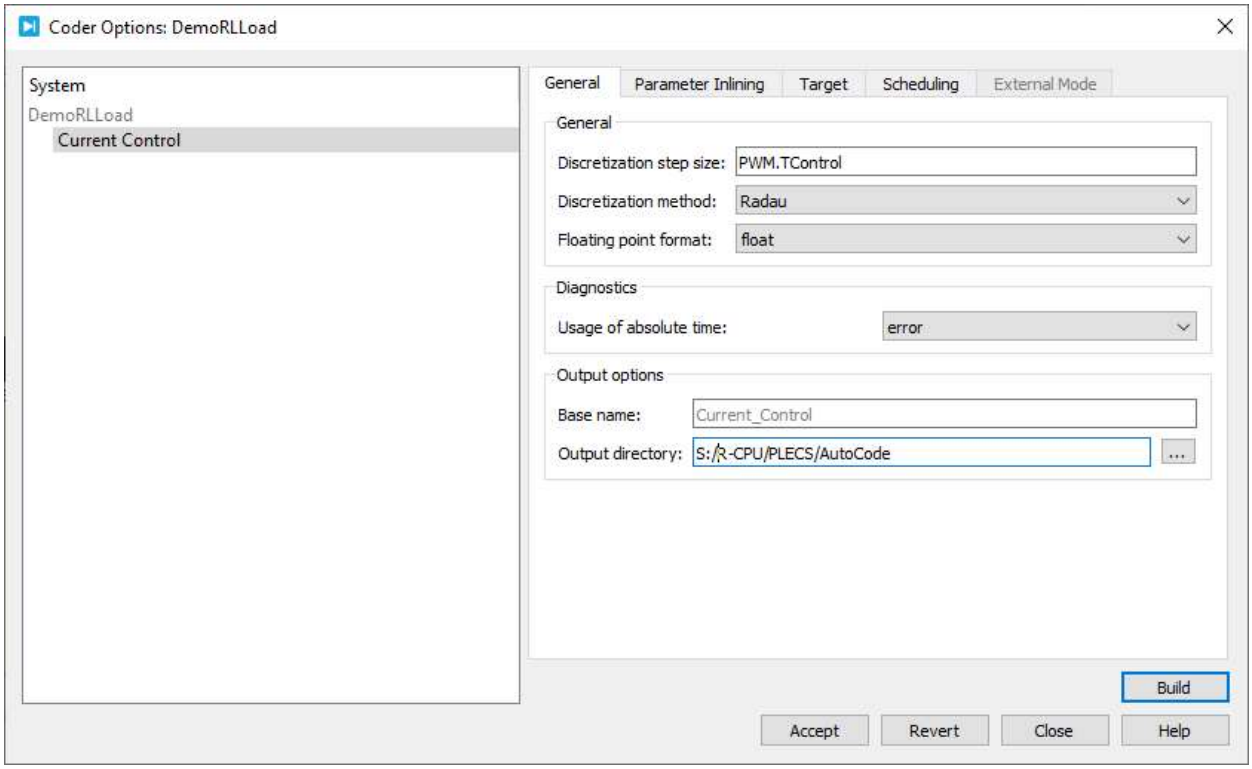


Figure 16 Coder configuration dialog of PLECS

Figure 17 depicts the target specific settings. If configured properly, the target GPD22kW can be selected and configured as required.

4.2.3 Parameter definition

The tab “Parameter Inlining” of the Coder settings can be used to give access to internal parameters of the control block by selecting “Inline parameter values” for “Default behavior” and drag-and-drop elements of the control block or its subsystems into the box “Exceptions”. All parameters present there are being added to the parameter list and can be accessed by the *Parameter Database*.
Since the name of parameters depend on the subsystems and elements of the hierarchy, it might be difficult to identify the required parameter in the list once a component has been dragged into the list. The dialog offers help by clicking the eye symbol next to the list. The same name is being used to identify the parameter in the code and in the *Parameter Database*.

NOTE: Parameters added via exception from inlining need to be copied in each step of the simulation and thus increase computation time of the real-time code.

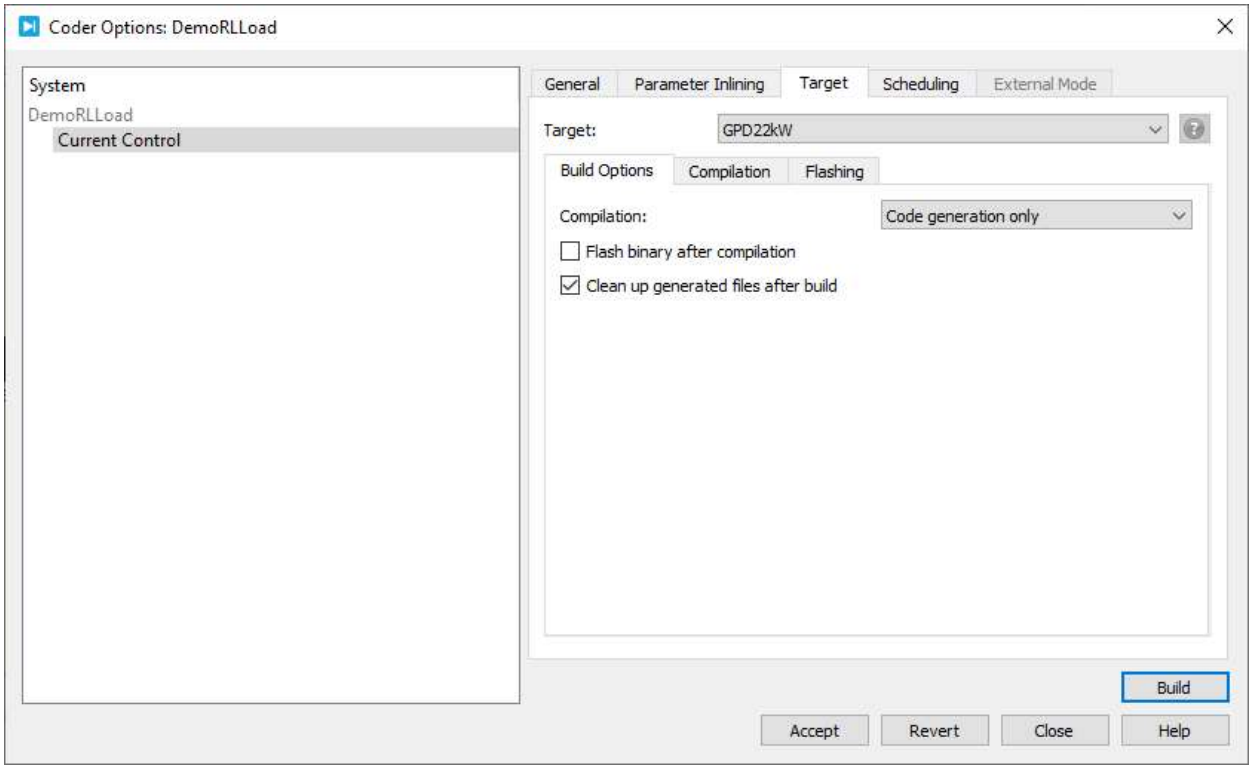


Figure 17 Target specific settings

4.2.4 Code generation

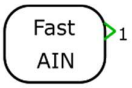
Once all settings are completed, the code can be generated, compiled and even flashed to the target as configured in the last step by clicking “Build”.

NOTE: When choosing to flash the code to the target, the code is being executed directly after flashing. For that to work, the system has to be powered and the J-Link connected.

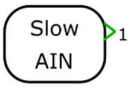
Due to the state machine of the software, the system does not power up immediately, however, and has to be started by the graphical user interface.

4.2.5 GPD target library reference

The following target library blocks are available for interaction of the control code with the software framework and the real-world interfaces of the platform.



The Fast AIN block gives access to the measurements updated each control code execution step. All measurements are subject to a calibration as parameterized by the *Parameter Database*, usually providing measurements of physical quantities in SI units. Additionally, a separate scale and offset can be added for convenience.
See the list of measurements in Section 3.2.1.4.



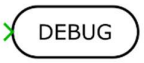
The Slow AIN block gives access to the measurements updated in a round-robin fashion and are not updated each control code execution step. All measurements are subject to a calibration as parameterized by the *Parameter Database*, usually providing measurements of physical quantities in SI units. Additionally, a separate scale and offset can be added for convenience.
See the list of measurements in Section 3.2.1.

The SIG OUT block can be used to write to a Signal of the software framework.
See the list of Signals in Section 3.2.3.

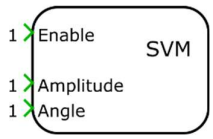
The SIG IN block can be used to read from a Signal of the software framework. See the list of Signals in Section 3.2.3.

The DOUT block is used to access digital outputs.

The PARAM block can be used to access a Parameter of the software framework. See the list of Parameters in Section 3.2.3.



The DEBUG block is used to provide access to an arbitrary amount of control signals using the *XScope* or within the simulation. To connect more than one signal, use a multiplexer.



The SVM block is the main output block of the software framework and directly interfaces to the APP provided by DAVE.
The output can be enabled and disabled by the Enable input.
The Amplitude and Angle signals control the switching pattern generation. The amplitude has to be provided per unit and the angle is represented by radians.
The block needs to be configured properly in terms of carrier frequency. This frequency and the discretization step size of the Coder settings determine the real-time behavior of the code. While the carrier frequency can be as high as desired, the discretization step size determines the IRQ frequency which is derived from the carrier frequency by event counting. See the involved DAVE APPs for valid ranges.

4.2.6 Example models

This section describes the models which come as part of the software package.

- The model V/f demonstrates the basic functionalities of the GPD target support package by basic voltage frequency control and an additional manual mode control.

4.2.7 V/f

This model demonstrates a simple V/f control. The model contains all required parameters for operation and is ready for simulation. If executed on the actual GPD hardware with an induction machine, it can be directly used to verify the functionality of the workflow and the hardware unit. Before operation, however, is has to be configured correctly. All parameters for configuration are present in the “Initialization” tab of the dialog “Simulation” → “Simulation Parameters…” as shown in Figure 18. Therefore, the parameter list has to be updated with application parameters.

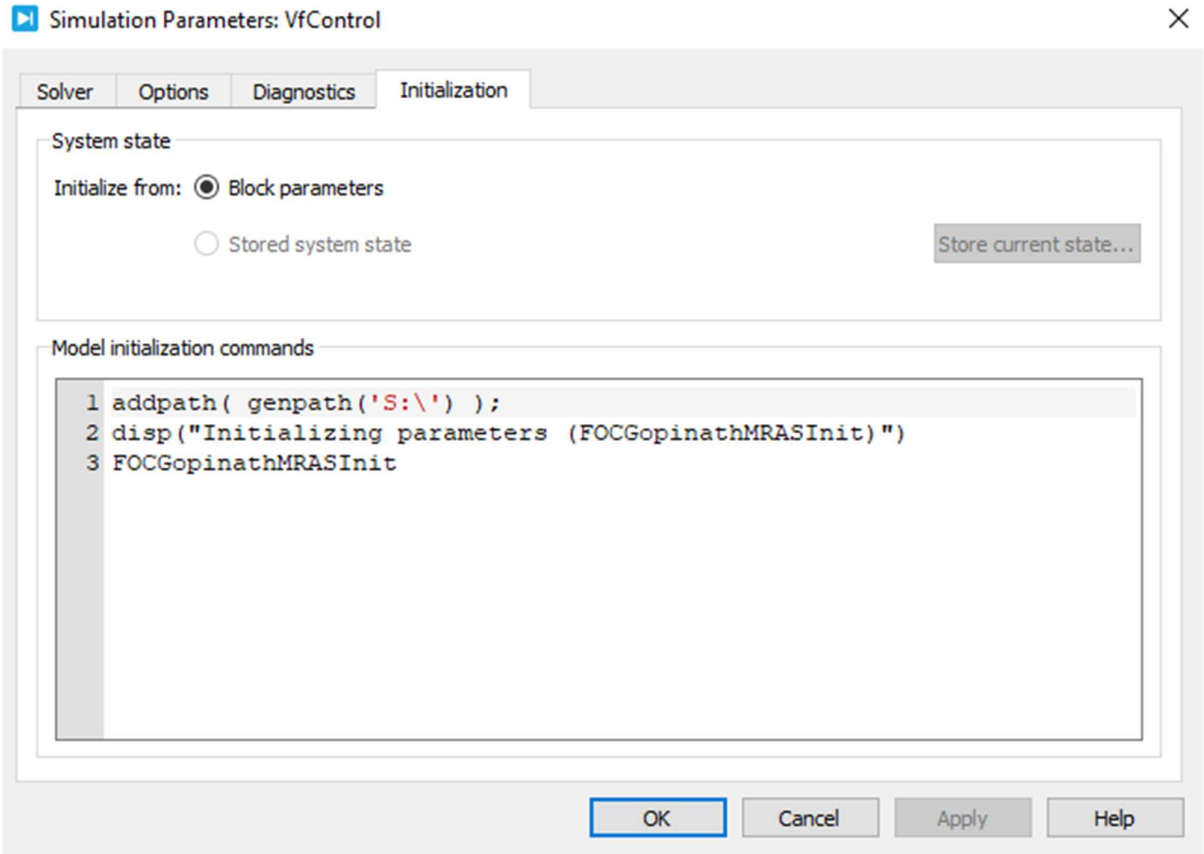


Figure 18 Simulation parameter definition for V/f model

The model itself is very similar to the one used throughout Section 4.2. It consists of a subsystem for the power electronic component similar to the one shown in Figure 19. Following the basic guidelines setup in this section, the control model is realized as a subsystem like the one shown in Figure 20. Additionally, it has an add-on subsystem which demonstrates the functionality of a brake-chopper.
The interfaces of the model to the GPD software are summarized in Table 22.

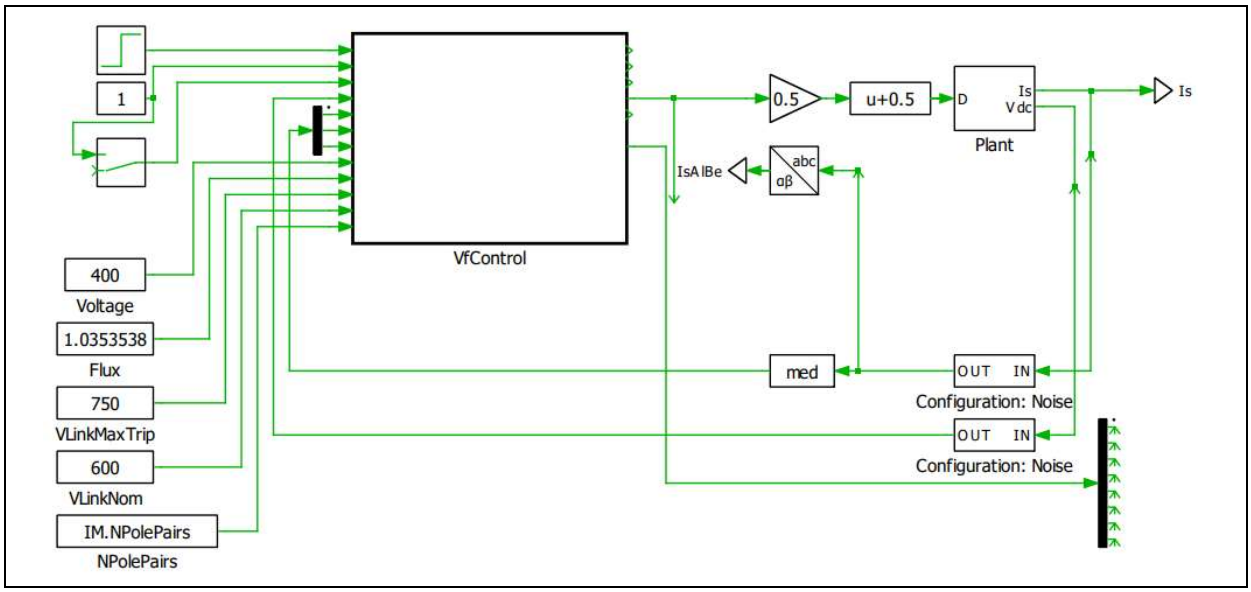


Figure 19 V/f control subsystem

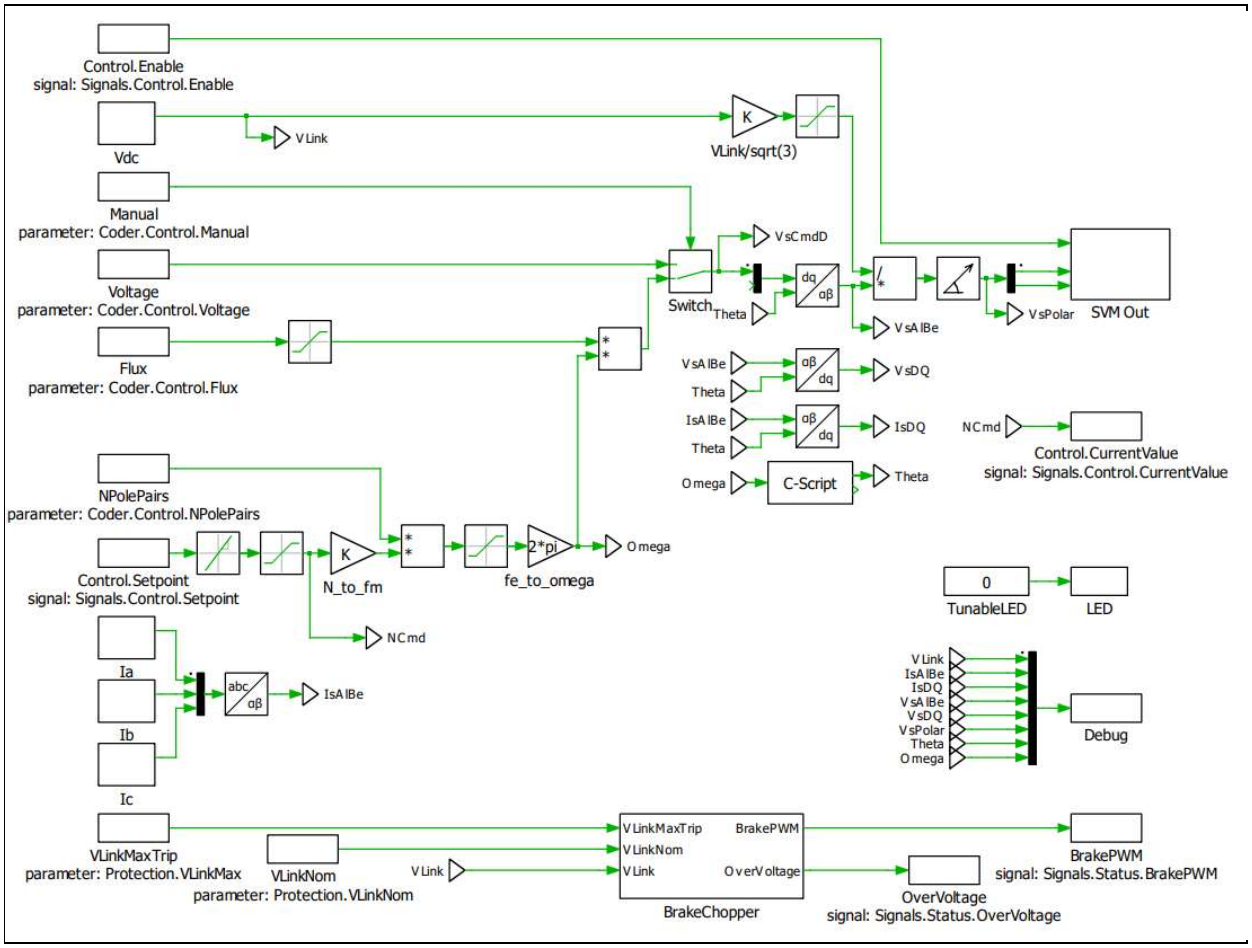


Figure 20 **V/f control**

Table 22 Click or tap here to enter text.

Name	Dir.	Type	Description
VDC	IN	Fast ADC	Measurement of the DC Link voltage
Ia	IN	Fast ADC	Measurement of the phase current A
Ib	IN	Fast ADC	Measurement of the phase current B
Ic	IN	Fast ADC	Measurement of the phase current C
IRMSCmd	IN	SIG IN	Setpoint for the phase currents in RMS
Enable	IN	SIG IN	General enable signal for the control
EnablePWM	IN	SIG IN	Enable signal for PWM generation
VLinkMaxTrip	IN	PARAM	Trip level for DC link voltage supervision
VLinkNom	IN	PARAM	Nominal DC link voltage
BrakePWM	OUT	SIG OUT	Brake chopper PWM duty cycle
OverVoltage	OUT	SIG OUT	Overvoltage indicator
Debug	OUT	DEBUG	Debug signals, added for debugging purposes
SVM Out	OUT	SVM	Output block for the space-vector PWM

The model is set up in accordance to the guidelines described earlier. The power stage can be configured to utilize a simple controlled voltage source or models of switching half-bridges. The control algorithm is implemented in the control subsystem and can be used for simulation and code generation.

The model is parameterized by a separate set of scrips which are loaded at simulation start or before code generation. See Section 4.2.1 for details on how to find the parameterization.

NOTE: Since the control scheme does not use additional sensors other than the phase currents, it is sensitive to parameter inaccuracies. The parameters therefore have to match the machine and system setup for expected performance of the control.

If unsure, simulate effects of a parameter mismatch by changing the machine parameters directly in the parameter masks of the respective component while keeping the parameters you would use for code generation unchanged. Do not forget to revert the changes.

4.2.8 Debugging with XScope

Testing and debugging is an important part of a development cycle. Close-loop controls are inherently difficult to debug due to the dynamic nature of control loops and because the loop obviously cannot be opened. A real-time view on external and internal quantities of the control loop therefore is essential.
The *XScope* toolchain offer insight into the software and all internal signals with up to 9 channels of floating-point data simultaneously. In addition to a graphical user interface, a code part that collects the data to be visualized is required to execute in each execution step of the control loop. For reduced computational burden or a reduction of data rate, it may execute once every N steps, sacrificing temporal resolution. A component in the *Background Domain* aggregates the data collected in the *Real-time Domain* and transmits it via an arbitrary communication interface to the graphical user interface. The R-CPU utilizes the WinUSB connection, often requiring a reduction in data rate.
The graphical user interface pre-processes and renders the incoming measurements and offers functions like an oscilloscope in that it supports scaling and placing single traces as well as controlling the time base of the view of traces. The view can be paused and unpaused, the traces can be recorded and screenshots of the current view can be exported. Each channel can be named, its visibility toggled and assigned a color. Cursors allow measurement of single samples, time and frequency.

NOTE: *XScope* does not process timing information from the debugging target. Therefore, it needs to be configured properly in order to support measurement of time or frequency. See Section 4.1.7 for a brief explanation of the configuration of *XScope*.

Figure 21 shows the scope view of the *XScope* debugging tool. The main part of the view shows the live data as traces similar to an oscilloscope, once connected to a running instance of the R-CPU.
Above the scope and above the red bar on the left-hand side are buttons to switch to the configuration view. The other buttons are the console (not in use here) and a help screen, explaining some of the configuration elements.
On the right-hand side below the red bar are buttons to save the current view as a screenshot, to configure the rendering properties, to pause and unpause the scope and to start and stop a record.
Next to the scope on the right-hand side are the control elements for each channel, the time base and the cursor. For each channel, there is a visibility checkbox, a scaling factor and a position (both of which refer to the divisions visible in the main view) and the channel color (clickable to choose a color). Most of the elements can be conveniently manipulated by the mouse wheel. Scrolling in the scope view is possible by using CTRL and the mouse wheel simultaneously.

The right-most part of the window is a control pane which holds buttons. The connect button establishes a connection to the R-CPU via WinUSB.

NOTE: When resetting the R-CPU, *XScope* needs to be disconnected and connected again.

Figure 21 Scope view of XScope

The R-CPU software framework allows the selection of the *XScope* channels via the *Parameter Database*. By default, the channels 1 to 9 are mapped to signals which show the most important quantities of the GPD target. Table 23 lists the default mapping.
If required, each channel can be mapped individually to either the default, a debug signal from the PLECS model or a signal from the software framework. The distinction is being made by an index per channel which

- is either -1 to activate the default mapping,
- is in range 0 to 999 to display a debug channel coming from the PLECS model (see `debugsignallist.h` in folder `PLECS/AutoCode` for signals and their index) or
- is larger than 1000 to show an internal signal (for indices and signals).

NOTE: The offset of 1000 required for the channel indices of the framework signals is a compile-time constant, preconfigured to match even large debug lists of PLECS models. If changed in code, the channel indices have to be adapted accordingly to see the same signals in *XScope*.

The parameters which control the mapping of the channels of *XScope* are called `Operation.LiveView.Channel1` to `Operation.LiveView.Channel9` and can be changed within the graphical user interface at runtime.
Example: Setting `Operation.LiveViewChannel3` to `1021` displays the median-filtered current of phase U on channel 3 in the *XScope* graphical user interface.

Table 23 Default mapping of the channels of XScope

Channel	Signal name / Function	Description
1	Current State	Current state of the state machine
2	IRQCounter	Counter of control loop execution events, overflowing at Control.ControlFrequency, thus leading to a sawtooth of 1 Hz

3	Control.Setpoint	Currently active setpoint for the model-based control
4	Measurement.Iph_U	Instantaneous measurement of the current of phase U
5	Measurement.Iph_V	Instantaneous measurement of the current of phase V
6	Measurement.Iph_W	Instantaneous measurement of the current of phase W
7	Measurement.VDC	Instantaneous measurement of the DC link voltage
8	Control.Amplitude	Command amplitude of the SVM generator
9	Control.Angle	Command angle of the SVM generator

Table 24 List of possible state IDs

State ID	State name	Usage
0	Initialize	Main state machine, see Section 3.2.1.5
1	None	Used internally
2	Precharge	Main state machine, see Section 3.2.1.5
3	Ready	Currently unused
4	Active	Main state machine, see Section 3.2.1.5
5	Pause	Currently unused
6	Stop	Currently unused
7	Error	Main state machine, see Section 3.2.1.5
8	Retry	Currently unused
9	Idle	Main state machine, see Section 3.2.1.5
10	Configure	Main state machine, see Section 3.2.1.5
11	Commissioning	Currently unused

4.3 GPD Operation

This section describes the operation of the GPD system by using the graphical user interface. The main functions are: operating the device (e.g. starting, stopping and providing setpoints), showing the state of the device (i.e. measurements, error states and the overall system state) and giving access to the parameters of the system.
The user interface is shown inFigure 22. It consists of a main window which is divided into two parts: the upper part shows different tabs whereas the lower part of the window contains a message area where error information and current messages are displayed. In addition to that, a dialog for diagnostic information can be opened from the *General* tab.

4.4 General tab

The *General* tab itself is divided into two areas as well: a schematic overview of the GPD device shows its components and displays state information and measurements. Above that schematic overview, there are elements which control the connection to the GPD target and the target itself. The elements of the top left corner are used to configure and control the connection to the C-CPU whereas the elements on the right-hand side are used to actually operate the GPD target and provide a setpoint.

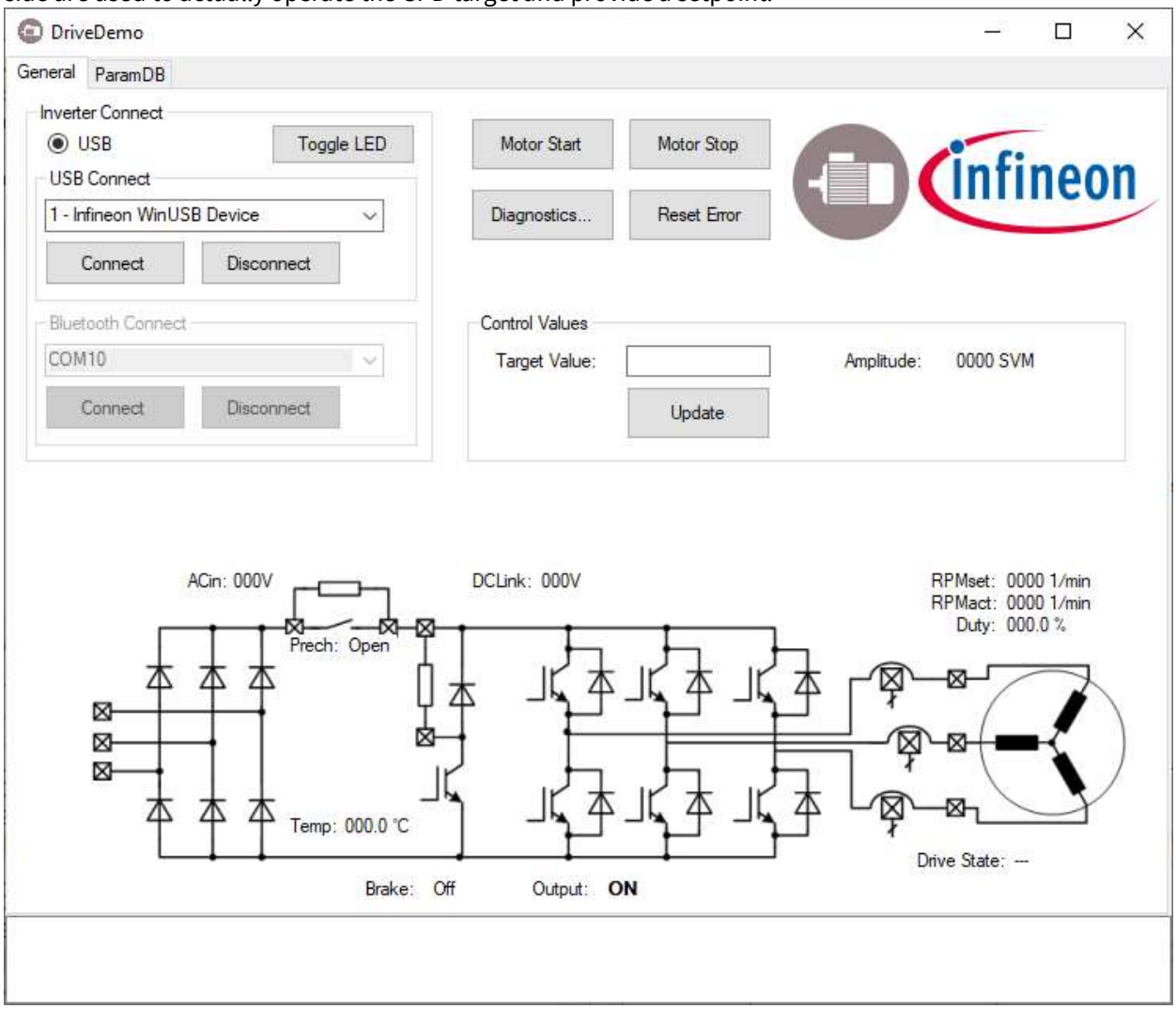


Figure 22 Main view of the graphical user interface

If the WinUSB drivers have been installed correctly, choosing the proper port and clicking “Connect” in the upper left part establishes the communication channel between the graphical user interface and the C-CPU of the GPD target. After connection, the message area of the main view displays information about the current software versions of the C-CPU and R-CPU and the parameter database being loaded. The system is now ready to interact. Clicking “Toggle LED” changes the state of one of the LEDs on the C-CPU board and can be used to check the connection.
The right-hand side of the view contains the main elements to operate the device. The buttons “Motor Start” and “Motor Stop” send commands which change the state of the R-CPU software to ACTIVE and back to IDLE respectively, thus starting and stopping the operation and the model-based control code on the R-CPU as described in Section 3.2.1.5.
To actually provide a setpoint to the control code, an arbitrary floating-point value can be entered into the edit box “Target Value”. After clicking “Update” the value is being transferred to the R-CPU to arrive in the signal `Signals.Control.Setpoint`. In return, the amplitude command for the SVM pulse generator is being transferred back to the GUI regularly and can be inspected next to the edit box.
In case of an abnormal situation during operation, the state machine of the R-CPU enters the error state and disables all switching signals. This error condition is being displayed appropriately in the user interface. Now either a closer inspection of the error is possible in the *Diagnostics* view or the error simply can be acknowledged by clicking “Reset Error”.
During operation, some of the measurements and status information can be inspected in the main view. The *Diagnostics* view offers additional information and is available at any time.

4.5 Diagnostics view

Upon clicking “Diagnostics” in the *General* tab, a dialog is opened which allows a closer inspection of the system Figure 23 shows the *Diagnostics* view.
In the upper part, a different structural representation of the GPD target is being shown: Information about the power supply, the status of the communication channels, temperature sensors, the fan and the hardware overcurrent protection is being displayed.
The lower part shows the status of all latched and instantaneous error flags as well as the values of phase current and DC link current measurements. Errors are being highlighted in red and the respective bits are marked as ‘1’ in the bit vectors (see respective source code for a definition of the bits). The information of the C-CPU is displayed on the left-hand side and described in Table 25. The flags of the R-CPU are shown on the right-hand side and described in Table 26.

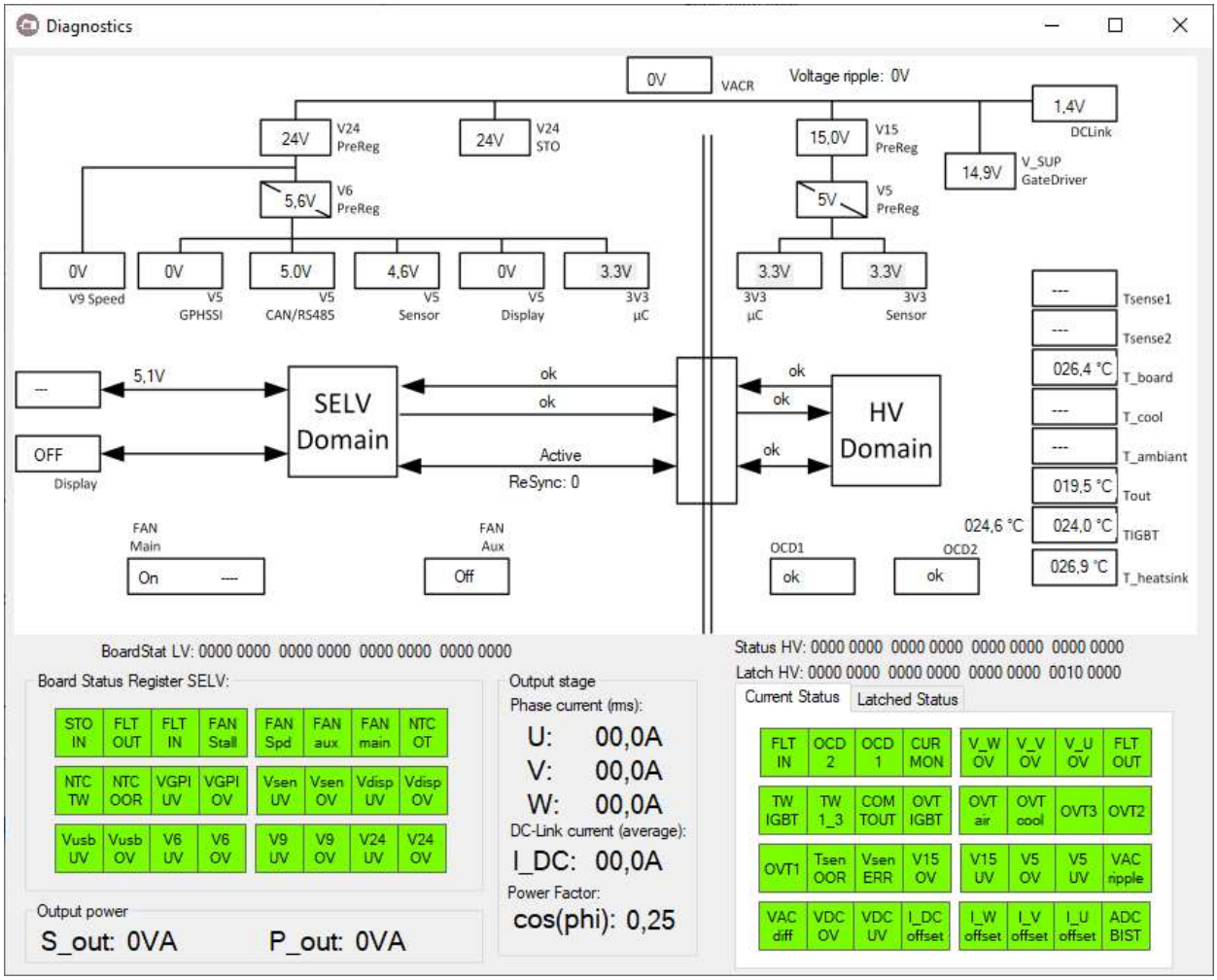


Figure 23 Diagnostics window

Table 25 Flags of the C-CPU software

Flag	Description
STO IN	Safe Torque Off input
FLT OUT	Fault out from SELV board (C-CPU)
FLT IN	Fault indication from HV board (R-CPU)
FAN Stall	Fan stall detected
FAN Spd	Fan speed signal missing
FAN aux	Aux fan status
FAN main	Main fan status
NTC OT	NTC overtemperature
NTC TW	NTC thermal warning
NTC OOR	NTC out of range
VGPI UV	V_GPHSSI undervoltage
VGPI OV	V_GPHSSI overvoltage
Vsen UV	V_sensor undervoltage
Vsen OV	V_sensor overvoltage
Vdisp UV	V_display undervoltage
Vdisp OV	V_display overvoltage

Vusb UV	V_USB undervoltage
Vusb OV	V_USB overvoltage
V6 UV	V_6 undervoltage
V6 OV	V_6 overvoltage
V9 UV	V_6 undervoltage
V9 OV	V_9 overvoltage
V24 UV	V_6 undervoltage
V24 OV	V_24 overvoltage

Table 26 Flags of the R-CPU software

Flag	Description
FLT IN	Fault indicator from SELV board (C-CPU)
OCD 2	Overcurrent detection 2
OCD 1	Overcurrent detection 1
CUR MON	Current monitoring of phase and DC Link current
V_W OV	Overvoltage detection in phase W
V_V OV	Overvoltage detection in phase V
V_U OV	Overvoltage detection in phase U
FLT OUT	Fault out from HV Board (R-CPU)
TW IGBT	Thermal warning IGBT module
TW 1_3	Thermal warning on NTC1, NTC2 or NTC3
COM TOUT	Communication timeout
OVT IGBT	Overtemperature IGBT module
OVT air	Overtemperature air sensor
OVT cool	Overtemperature cooler
OVT3	Overtemperature NTC3
OVT2	Overtemperature NTC2
OVT1	Overtemperature NTC1
Tsen OOR	IGBT module temperature sensor out of range
Vsen ERR	V_SENS error
V15 OV	V15 overvoltage
V15 UV	V15 undervoltage
V5 OV	V5 overvoltage
V5 UV	V5 undervoltage
VAC ripple	V_AC ripple overvoltage
VAC diff	Difference between V_AC and V_DC too high
VDC OV	V_DC-Link overvoltage
VDC UV	V_DC-Link undervoltage
I_DC offset	DC-Link overcurrent OR sensor offset too high
I_W offset	Phase W overcurrent OR sensor offset too high
I_V offset	Phase V overcurrent OR sensor offset too high
I_U offset	Phase U overcurrent OR sensor offset too high
ADC BIST	ADC build in self-test error

4.5.1 ParamDB tab

In addition to the *General* tab, the *ParamDB tab* allows inspection and control of all parameters of the system. Figure 24 shows the application with an activated *ParamDB tab*. In the upper area, the list of parameters is being displayed once the synchronization process between the CPUs succeeded. In the lower area, there are the elements to control updating the table and storing the current set of parameters to non-volatile memory.

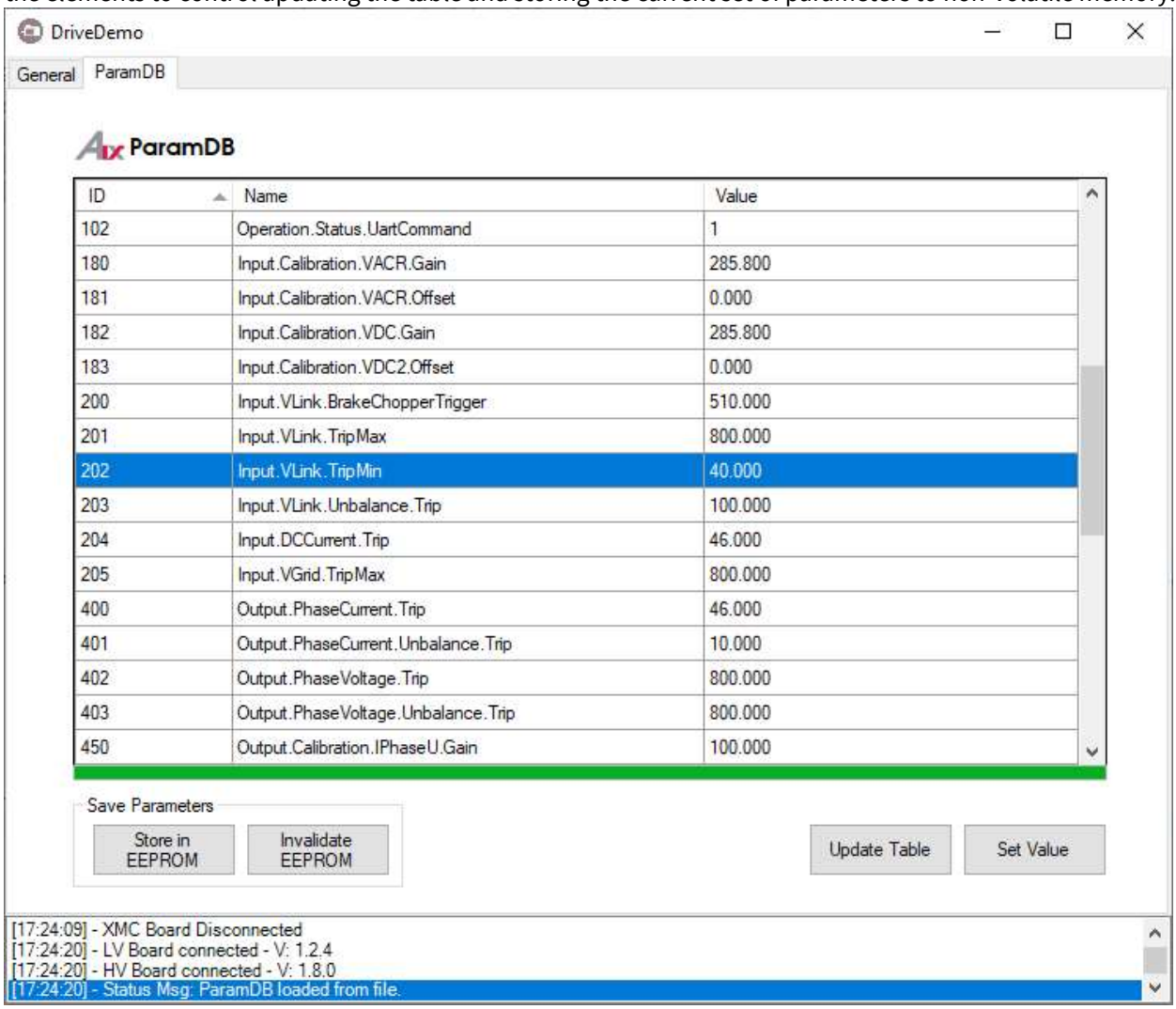


Figure 24 Parameter view of the user interface

After starting the application, the list of parameters needs to be updated by clicking “Update Table”. Once updated, parameters can be inspected and directly changed in the list by selecting a parameter and typing a new value. Floating-point values are entered using a either a colon or comma as decimal point. A click on “Set Value” or pressing Enter updates the parameter in the list and sends an update to the C-CPU end eventually to the R-CPU. The message area of the main window informs about the success of the operation. Figure 24 list parameters and signals accessible this way.

As described in Section 3.4.2, the *Parameter Database* loads from and stores to a non-volatile memory. When executing the software for the first time, however, there might be no binary to load from and the parameter set linked into the software binary is being used. The message area of the main window generally informs about loading and storing the parameters. It either states “loaded from file”, meaning that the database has been initialized from the linked-in parameters, or it mentions “loaded from EEPROM” which indicates that the

parameter binary in the non-volatile is valid and the parameters have been restored from the previous operation of the GPD target.
The parameter storage to non-volatile memory is not being done automatically. If storing the current parameter is desired, a click on “Store in EEPROM” saves the current set of parameters in the C-CPU to the non-volatile memory. The message area shows the result of the storage operation.
Should the binary be corrupted or the parameters should be initialized for some other reason (e.g. a software update of the C-CPU), the contents of the non-volatile memory can be invalidated by clicking “Invalidate EEPROM”.

5 Commissioning

The REF-22K-GPD-INV-EASY3B must be connected to a TN-S line system as shown in Figure 25. The connections to the inverter can be made only after the converter is unpacked and all packaging material has been removed.

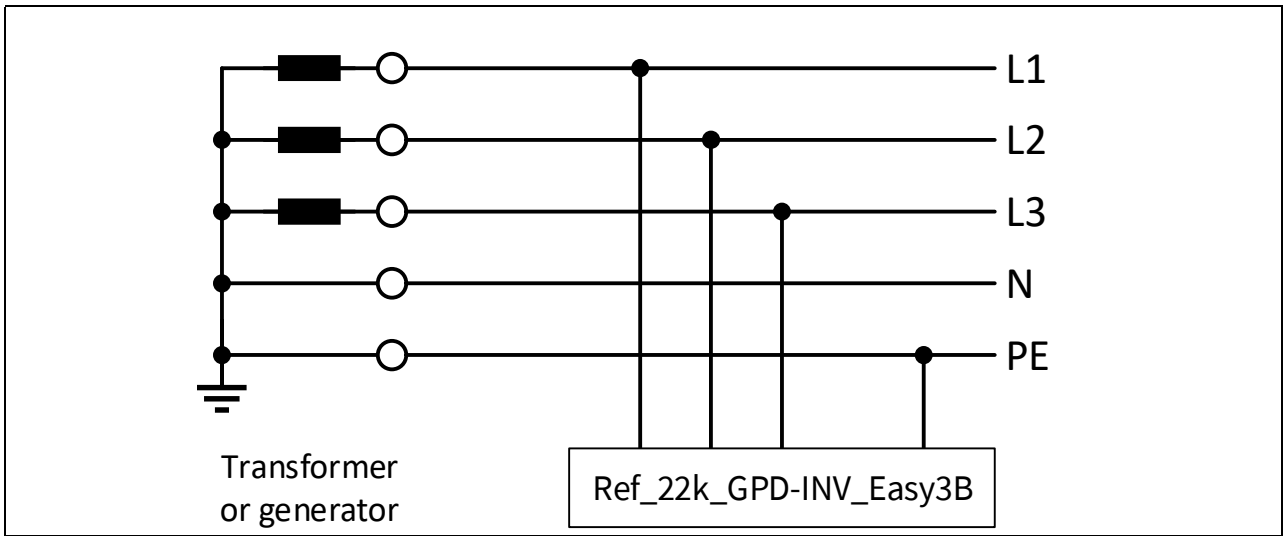


Figure 25 Connection of the inverter to TN-S line system

A TN-S system transfers the PE protective conductor to the installed plant or system using a cable. Generally, in a TN-S system, the neutral point is grounded. A TN system can transfer the neutral conductor N and the PE protective conductor either separately or combined. Also the connection to a TT system is permissible, but the connection to an IT is not permissible. The reason here is the missing or high-impedance earth connection.

The connection of the inverter is shown in Figure 26. The cross-section of the wires for the line connection, the brake chopper and the motor cable are in the range of 1.5 to 10 mm². Ensure that the inverter is in a no-voltage condition and the DC-link is discharged.

To connect the line feeder cables (L1, L2, L3 and PE), see Figure 26. To ensure correct fusing, it is recommended to make a selection according to IEC is 3NA3824 (80A). Also, connect the motor feeder cables (PE, U, V and W) to the inverter. The connection of a brake resistor is optional.

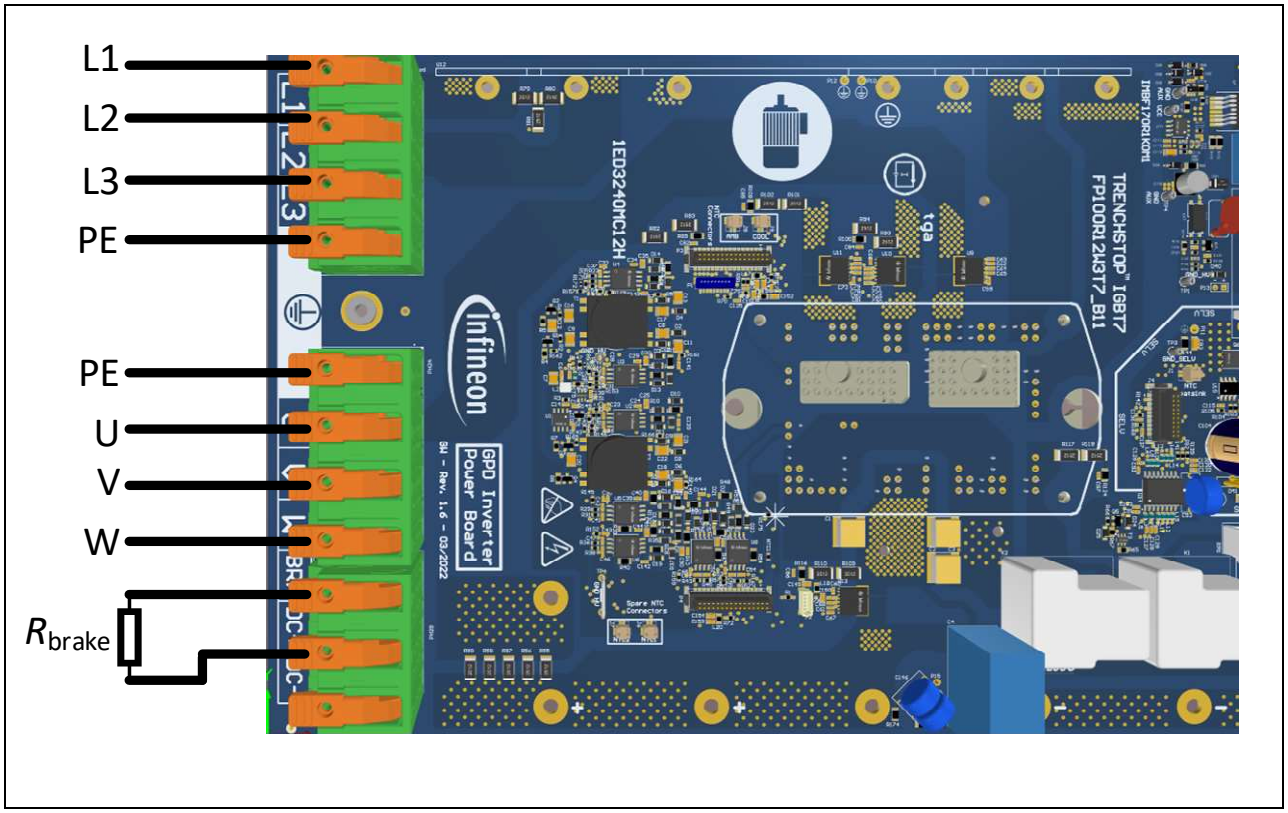


Figure 26 Connecting of the inverter

Also connect a USB cable to the inverter allowing for communication with the drive. The USB port is located at the front of the inverter; see Figure 26.

Before you start the inverter, you need to install the XMC driver and launch the GUI. The software can be downloaded via Infineon.com. Please register your inverter to get access to the software.

For installation, it is required to have admin rights on your computer. The software was tested with a laptop (HP EliteBook 840 G5) and Windows 10 Enterprise (Build: 10.0.17134). You have to first install the USB driver, so extract the file XMC_WinUSBDriver.zip.

To start with the installation, connect your computer with a USB cable to the inverter, and apply an appropriate voltage to the line feeder cable. The inverter will start operation in idle mode. Now you can install the USB driver: Go to “Control Panel\All Control Panel Items” then to “Device Manager.” Select the new USB device “Infineon WinUSB Device;” see Figure 27 part 1, and double click it.

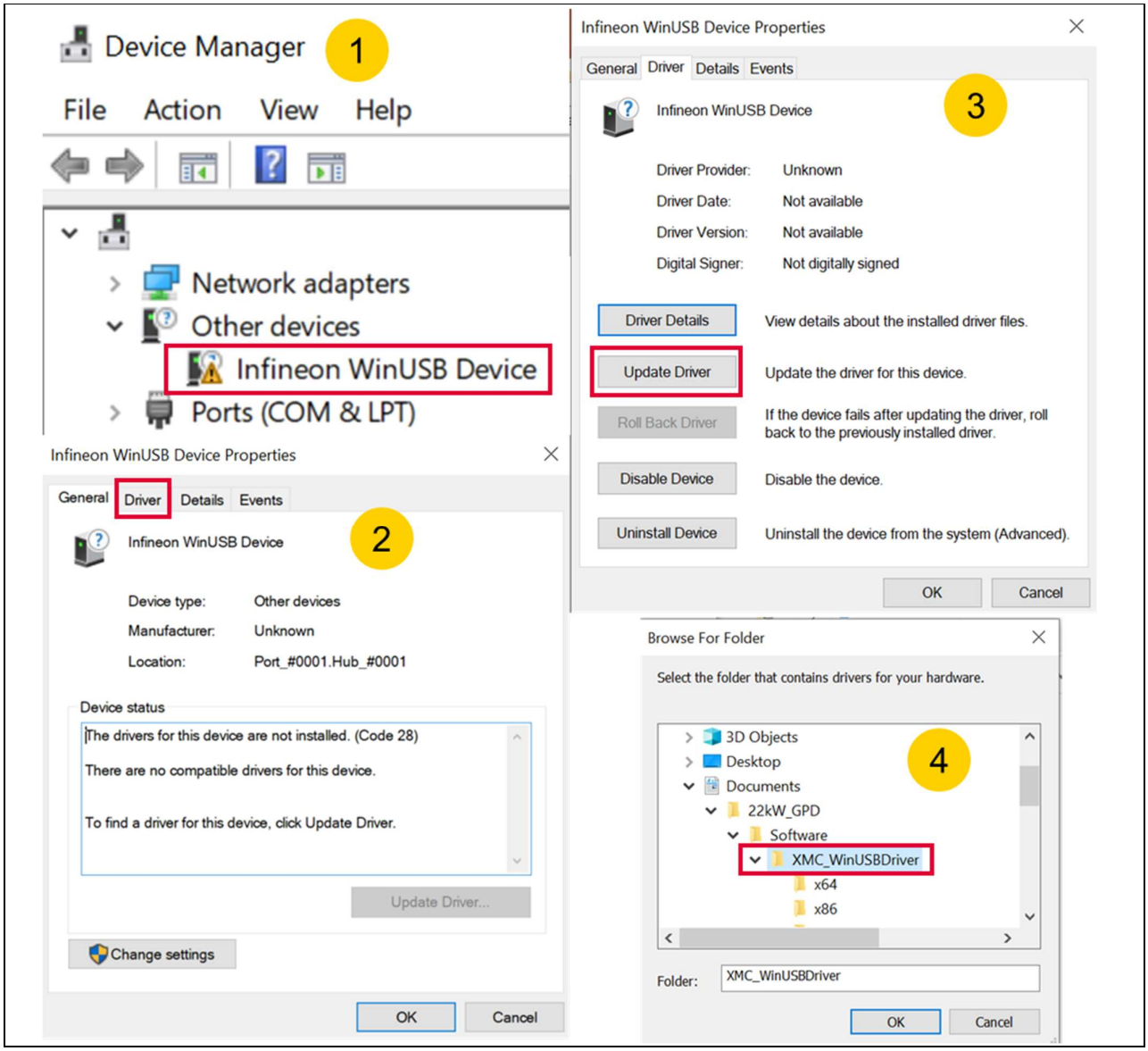


Figure 27 USB device installation

The window, as shown in Figure 27 part 2, will then appear. Click on the “Driver” tab, then “Update Driver” as shown in Figure 27 part 3. Now select the extracted folder “XMC_WinUSBDriver” as shown in Figure 27 part 4. Click on “OK” and close all windows you opened previously. The USB driver is now installed.

The next step is to extract the file GUI_20200819.zip; the software is available at Infineon.com. Open the folder “Debug” and double click on “GPD Inverter.exe”. A window as shown in Figure 28 (left side) appears. You can connect your drive system by selecting the USB device via the drop-down menu and click on “Connect.” Now your computer is communicating with the drive system.

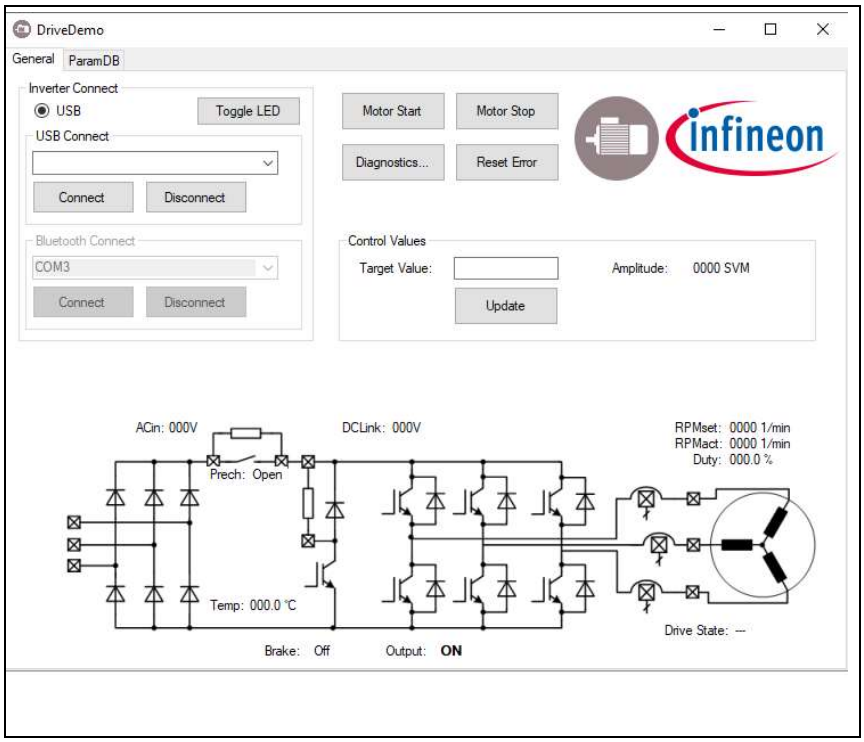


Figure 28 Main GUI and Config tab

By clicking on the “Config” tab, a window will be shown as in Figure 28. You can read the Config file from your inverter by clicking on “Read config.” Now you can change the settings. The “Control Method” can be selected either as “V/f” or “Manual.” The parameters for the “V/f” mode can be adjusted in the “Inverter Configuration” window. In the manual mode, you can select the “Speed” value and the “Amplitude” value manually; see the “General” Tab of the GUI. In the “Config” tab you can also change the “IGBT Overtemp shutdown” value. The “Load Pulse Pattern Config” window allows you to operate the inverter in a periodic or cyclical mode. This function works only in “Manual” mode, and changes the duty-cycle value for a specific time. This approach allows for an easy implementation of an overload pattern, as shown in Figure 3 using an inductive-resistive load. In addition, a short-circuit test is also possible via the GUI; further details will be explained in Section 6.5.

Via the “General” tab, you can start the motor. You can run the motor in “V/f” mode; see section “Control Method.” The speed of the motor, hence the output frequency of the inverter, can be adapted by the slider “Speed.” After selecting your speed, click on “Update.” You can change the “V/f” ration in the “Config” tab, as already mentioned. Via the “Manual” control mode, you can select the “Speed” and the “Amplitude” which represents the modulation index.

The current status of the inverter can be seen by clicking on “Diagnostics.” A window will appear as shown in Figure 29. The voltage levels of the different power supplies will be shown, as well as any flagged faults in the inverter.

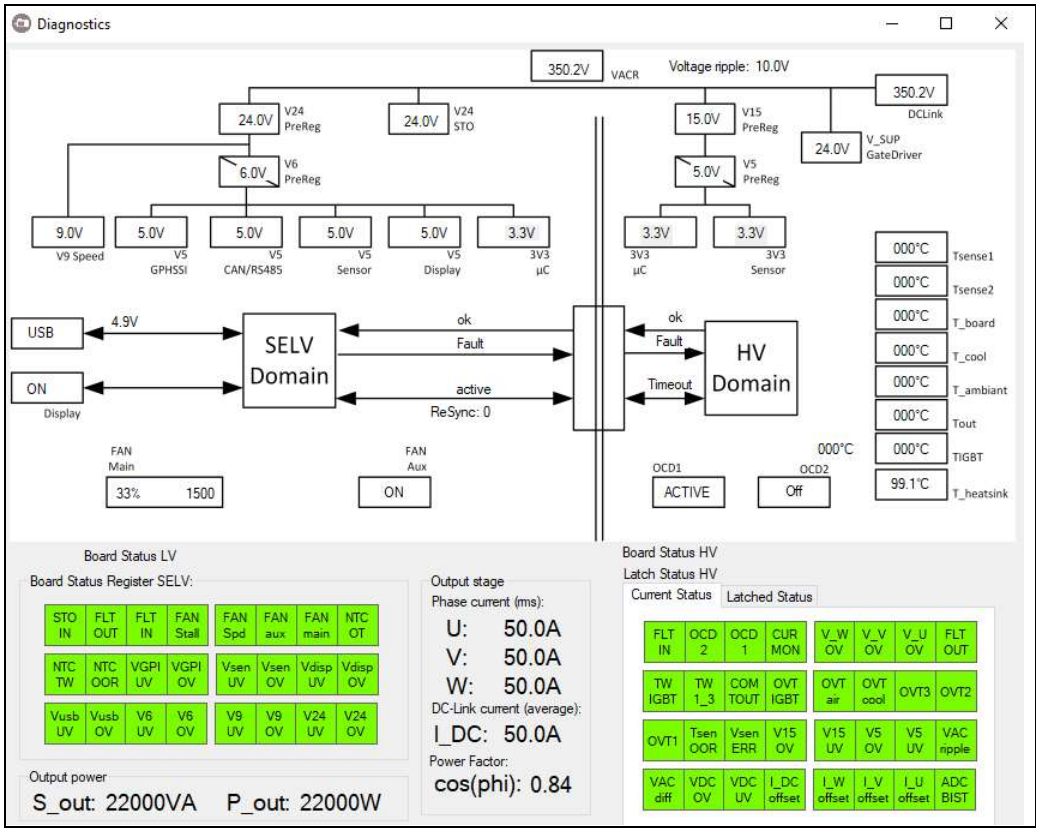


Figure 29 Diagnostics and oscilloscope of the GUI

5.1 Description of the functional blocks

5.1.1 Hardware partitioning

The inverter hardware is partitioned into five boards:

- Power board
- EMI filter
- High-voltage logic board
- Low-voltage interface board
- DC-link board

Thanks to the separation of these individual functions, a high level of functionality can be implemented with minimum board space. Each board will be described in the following:

Power board:

The power board is used as a central connection and wiring unit between all other sub-boards. It contains the main connectors (power inlet and outlet), the power module, the main DCDC-converter and the gate driver

including their power supplies. While the power board itself only contains a small DC-link capacitor, it connects to the capacitor bank which has ix (3 sets of 2 in series) electrolytic capacitors.

DC-link board:

The board has the electrolytic capacitors of the inverter. The separation of the capacitors on an additional board allows more design freedom and a more compact design of the inverter.

EMI-Filter:

The EMI input filter is used to suppress RF noise generated by the inverter operation. It is designed as three-phase CLC-topology. Due to the size and weight of the individual components, it is implemented on a separate sub-board that connects to the power board via screw connectors.

High-voltage logic board (FELV board):

All logic functions required to control the drive motor are implemented on the high-voltage (HV) logic board. It is based on a XMC4800 microcontroller as its central processing unit. The board contains the infrastructure to supply and supervise the microcontroller, diagnostic functions for the high-voltage part of the drive as well as all safety and protection functions needed to protect the inverter in case of overload or short-circuit conditions. The HV board features a direct connection the components on the power board. All control inputs for the motor control as well as the diagnostic status information are provided via a UART link to the low-voltage (SELV) interface board. Besides the UART link, the board features both a digital status input and output signal to indicate and receive information on critical error or fault conditions.

In order to minimize the number of required isolation barriers, the FELV board is referenced to the negative potential of the DC-link.

Low-voltage logic board (SELV board):

The low-voltage logic board is used to provide the user and external sensor interface functions. It features connectivity options for:

- EtherCat
- USB 2.0
- WLAN
- Bluetooth
- RS485
- High-Speed CAN

Besides the user interface function, the board also provides the capability to interface with external sensors, such as:

- High-resolution rotor speed sensor (two wire current interface)
- High-resolution rotor angle sensor (GP-HSSI Interface)
- Temperature sensor
- STO A & B Input

The data exchange between SELV and FELV board is established via a bidirectional UART link. All relevant control commands received via the above-listed interface options or the touch display are forwarded to the

FELV board via this link. For safety and redundancy reasons, the board provides two additional direct connections to the FELV board (1x input, 1x output) for error indication on the respective board. In case of a broken UART connection, critical error conditions on either board can be indicated via these redundant signals to cause the inverter to transition into a safe state.

In order to simplify the supply scheme for the multiple external user interfaces, the SELV board is supplied via a safety-isolated, low-voltage power supply. The connection to the high voltage logic board (UART, direct connection lines) is therefore established via digital isolator devices featuring reinforced isolation capability.

The STO A & B inputs are factory - bridged with short cables. The function is not designed according to functional safety specifications and can be used as an enable inputs to shut of the drive. The function should not been used to protection before injury. Other precautions must be taken into account!

5.1.2 Isolation coordination

The inverter design provides three major main voltage domains:

- A mains-connected domain
- A low-voltage domain with functional isolation
- A safety-isolation domain

In Figure 30 you will find a drawing of the isolation coordination scheme.

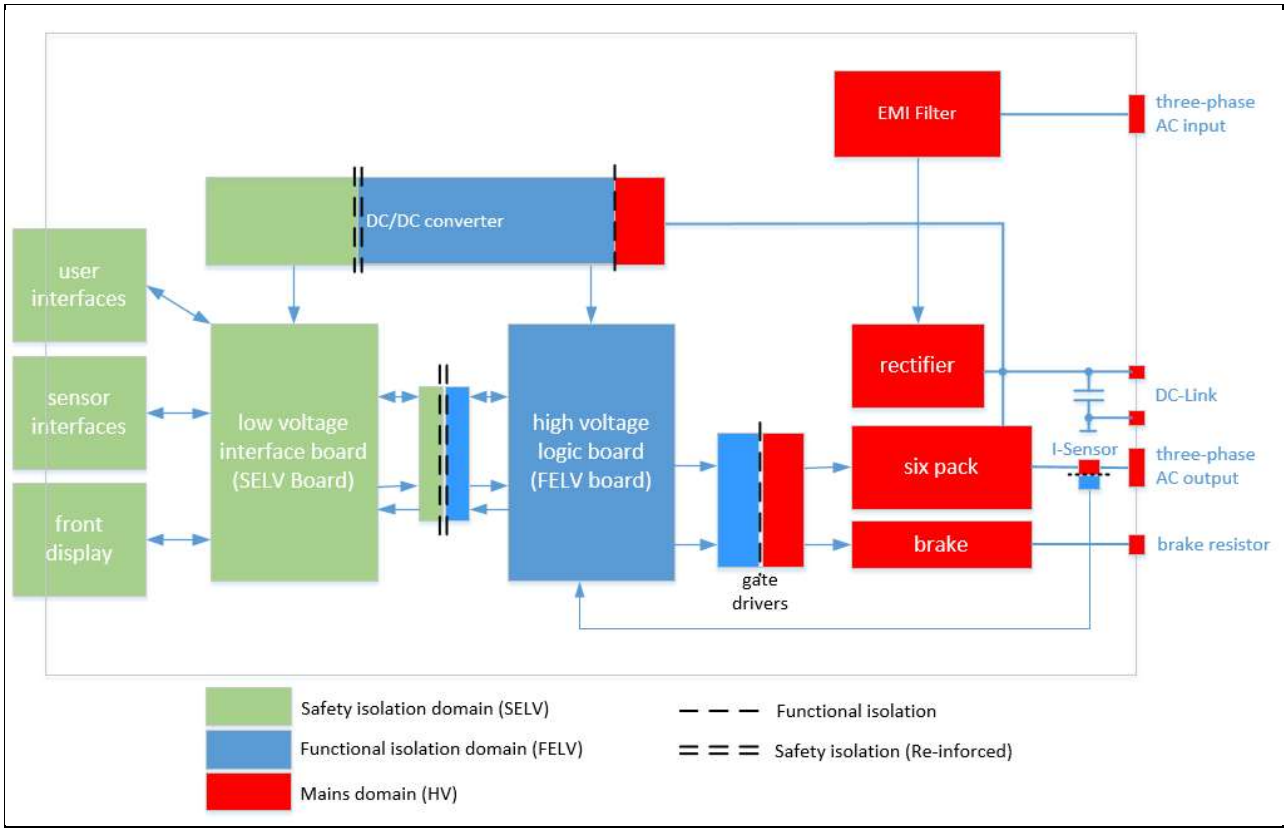


Figure 30 Basic isolation coordination scheme, the low- voltage interface board is referenced to PE and the high- voltage logic board is referenced to DC-

All power components (EMI filter, rectifier, IGBT inverter, brake switch) are directly connected to the mains supply domain. The main controller is based on a functional isolated scheme, which is connected to the DC-potential of the power domain. Thanks to this isolation scheme, all interfaces between the power domain and the main control domain only provide functional isolation. This applies in particular for the gate drivers, their power supply, the output current sensors and the voltage measurement circuits. All user interface connections are implemented on a safety-insulated domain. In order to establish a communication link between the safety-isolation domain and the functional isolation domain, a digital interface using a safety isolated data coupler is implemented. The supply for the entire board is provided by a centralized DC/DC converter which takes power from the mains supplied DC-link. The converter provides two output voltages, a functional isolated output for the functional isolation domain (FELV) and a reinforced output voltage for the user interface (SELV) domain.

5.1.3 Board interconnection schemes

The power board is used as a main platform for all sub-boards. The mains power connection is performed by power connectors on the bottom-left side of the inverter board. After passing the EMI filter board, which is connected to the left side of the power PCB, the power is routed to the power module. The rectification of the AC input power is performed in the power module; the DC-link capacitors are located on the right side of the inverter and are connected via the power PCB. Since the capacitor bank is implemented as a separate unit, the mechanical and electrical connections are made by special mounting screws. A CAD-model of the inverter is shown in Figure 31.

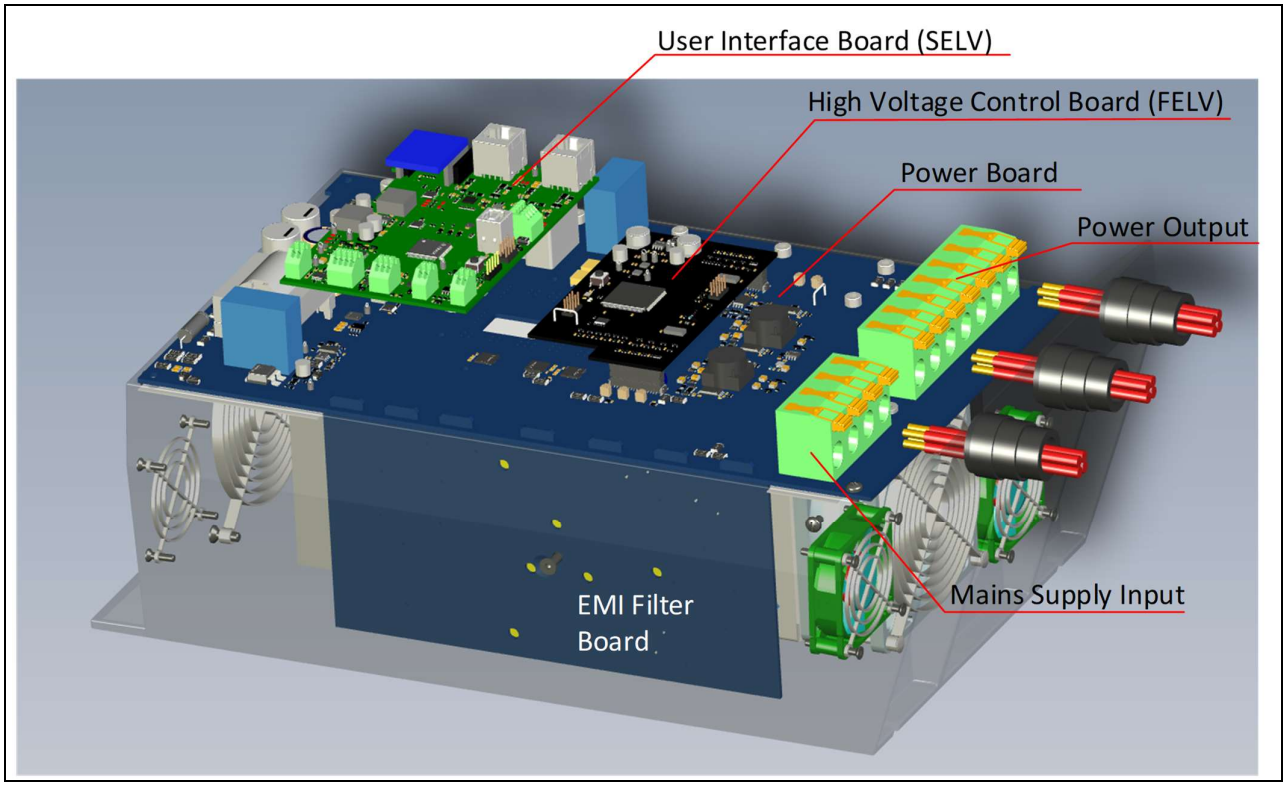


Figure 31 Inverter Construction Scheme (front-left view), Housing removed

The inverter power supply is performed by a DC/DC converter which generates galvanically isolated supply voltages for the power board, the high-voltage control board and the user interface board. Both user interface boards as well as the high-voltage control board are connected the power board via pin headers.

The user interface board connects to the power board on a safety-isolation island which provides the DC power for the board and the isolation stage for the digital communication link to the high-voltage control board. All gate-driver control signals as well as all voltage and current measurement signals are routed to the high-voltage control board via high pin-count connectors.

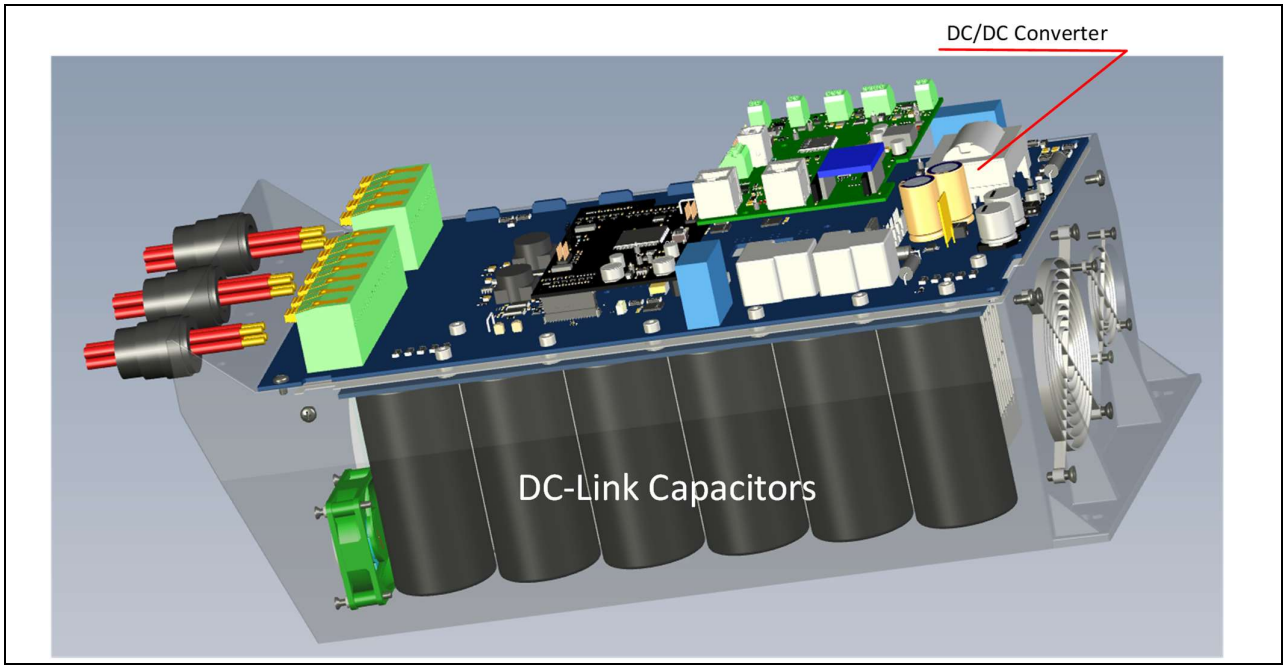


Figure 32 Inverter Construction Scheme (top-right view), Housing removed

5.1.4 Supply schemes

The three-phase AC input voltage is rectified to the DC-link supply voltage by the B6 rectifier bridge in the power module. In order to prevent high inrush currents during power-up when the three-phase supply is first connected, the DC-link capacitors are pre-charged via power resistors. After the capacitors are charged to match the line voltage, the power resistors are shorted out using power relays.

The DC-link voltage is converted to two low-voltage domains by a SiC-based flyback converter. A first output voltage (V15_HV) is used as the low-voltage supply for the functional isolated part of the inverter, supplying the gate drivers and the high-voltage logic board. The second voltage domain (V24_SELV) is used to power the low-voltage user interface board, local sensors (e.g. heat sink temperature sensor) and the cooling fans.

5.1.5 Component selection

The power module is in an EASY 3B package, part number FP100R12W3T7_B11, based on the latest IGBT7 technology allowing a compact and economic solution for an industrial drive. The FP100R12W3T7_B11 module carries all power semiconductors which are needed to control the motor current [1]. Due to the reduced VCEsat voltage compared to IGBT4 and enhanced switching controllability, the IGBT7 technology is tailored for industrial drives. For short-circuit protection of the IGBTs and for motor current control, the current sensor TLI4971-A120T5 is used. The TLI4971-A120T5 provides a fast short-circuit detection time and accurate measurement results over a wide temperature range [2]. For the gate driver IC, the EICE™ driver 1ED3240MC12H was selected. It is a dual-channel isolated IGBT gate driver IC with two level slew rate control.

The Infineon CoolSiC™ MOSFET IMBF170R1K0M1 with a blocking capability of 1700 V is used for the auxiliary power supply. By using the SiC-MOSFET, the design of the flyback inverter can be simplified.

The reference inverter uses two microcontrollers; one for control and one for communication. For control, the XMC4800-F144F2048 is used, for communication, the XMC4300-F100K256 AA is designed in. Both microcontrollers communicate via a UART interface.

5.2 Setup two level slew rate gate driver

For setting up the right gate resistors a double-puls characterization has been done depence on the powerboard.

The load is a three-phase R-L load with a particularly high resistive component, which results in a particularly high power factor. However, one can assume that the load current will not change during turn-on and turn-off.

The gate resistor values were selected in line with the double-pulse characterization results, aimed at keeping the inverter's dvCE/dt below 5 V/ns for all operating conditions. This design rule takes into account the variation of temperatures and collector currents.

The test results indicate that the target of 5 V/ns is met at a gate resistance of e.g. RG,on = 3.7 Ω for turn-on at low currents of 10 A. Such double-pulse test results are usually too optimistic for commercial applications, as the applications cannot be optimized in the same way as laboratory tests. Therefore, a value of 18 Ω was selected for slow turn-on. Double-pulse tests had been performed also for evaluating dvCE/dt at turn-off resulted in the turn-off gate resistance RG, off at high currents. These two values would be the selected values for a conventional gate driver.

Selecting gate resistances for the two-level, slew-rate control gate driver also requires that the changeover point be defined. These gate resistance values are given in Table 27 for reaching the target of 5 V/ns for turn-on and turn-off. [8]

Table 27 Gate resistor values for fast and slow switching

$R_{G,on,slow}$	18 Ω
$R_{G,on,fast}$	3.7 Ω
$R_{G,off,slow}$	11.2 Ω
$R_{G,off,fast}$	3.1 Ω

The resistances for RG, on, slow and RG,off, slow are selected to yield a very slow switching speed, which indicates EMI-friendly behavior. The resistances for fast switching (RG,on,fast, RG,off,fast) are considerably lower. Clearly, fast switching is more efficient when it is activated in the appropriate operating range. However, severe oscillations could be triggered, if such low gate resistances are used at very low currents [8].

The measurements in this section compare the results of 3 different modes of operation:

- Mode 1: slow turn-on, fast turn-off
- Mode 2: fast turn-on, slow turn-off
- 2L-SRC-mode with
    - IC < 35 A: slow turn-on, fast turn-off
    - IC > 35 A: fast turn-on, slow turn-off

The correlated gate resistor value for each mode is given in Table 27. The 2L-SRC mode combines both mode 1 and mode 2 depending on the instantaneous phase current. Table 28 shows the recalculated physical resistor values of resistors R1 through R4.

Table 28 Effective gate resistor values for fast and slow switching

	Mode1	2L-SRC	Mode 2
$R_{G,on,slow}$	18 Ω	18 Ω	–
$R_{G,on,fast}$	–	3.7 Ω	3.7 Ω
$R_{G,off,slow}$	–	11.2 Ω	11.2 Ω
$R_{G,off,fast}$	3.1 Ω	3.1 Ω	–

An analysis of the switching speed is shown in Figure 33. The upper graph in Figure 33 depicts 1000 sampled waveforms of turn-on events, which represents a good variety of these occurrences. The sampling of the waveforms is synchronized to the load current frequency, so that switching waveforms are sampled randomly. The lower graph shows a histogram of the sampled waveforms structured in dv/dt clusters. Each bar represents the number of events with a width of 0.05 V/ns starting at 1 V/ns on the left side. As seen in Figure 33, the transients range from 0.5 V/ns to 4.5 V/ns, and thus do not exceed 5 V/ns. The IGBT is therefore driven within the expected range of 5 V/ns. [8]

The selected gate resistor values in Table 29 are corresponding with the descripted schematic in Section 1.5.

Table 29 Selected gate resistor values R1, R2, R3 and R4

R1	30 Ω
R2	18 Ω
R3	4,7 Ω
R4	51 Ω

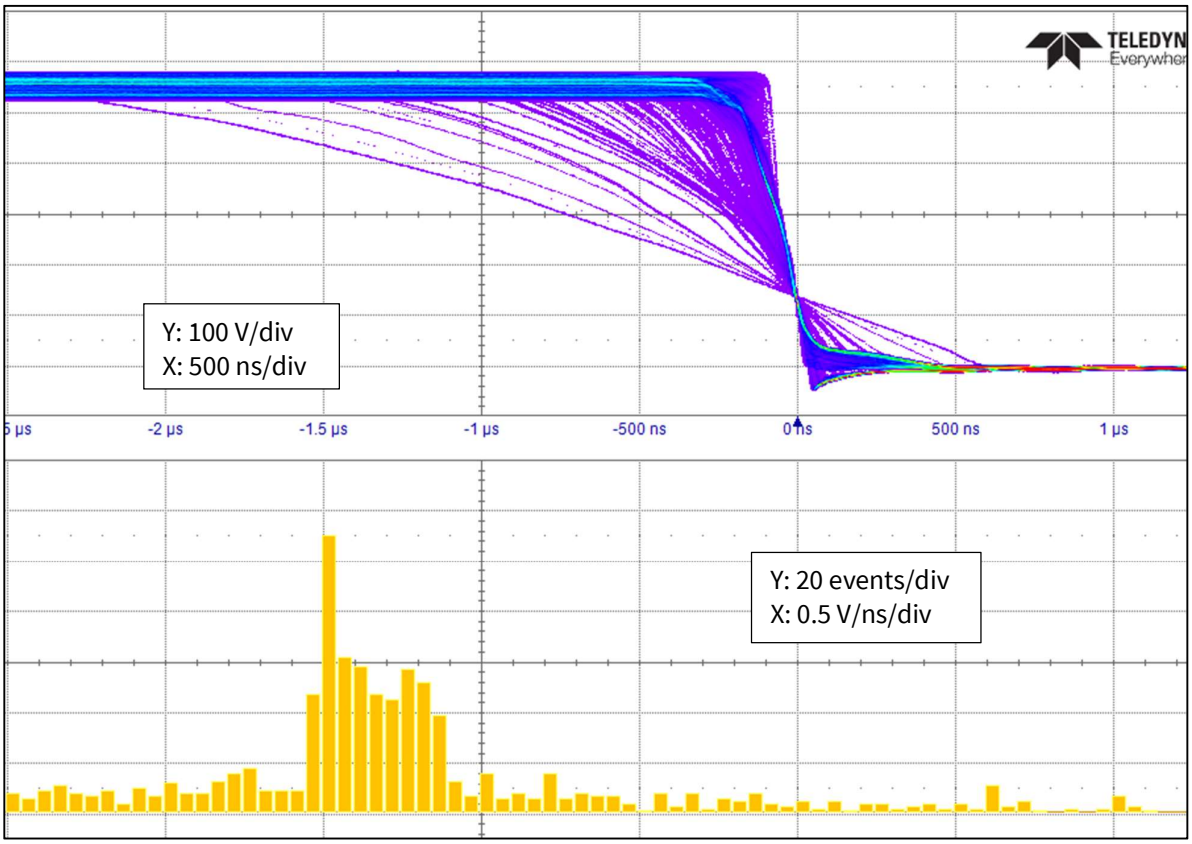


Figure 33 Figure 1 Example of a set of turn-on waveforms (top) and histographic analysis of dvce/dt (bottom) [8]

6 System performance

6.1 Test results inverter start-up

The inverter is supplied via an auxiliary supply which derives its energy from the DC-link capacitors, so the power supply starts when the DC-link voltage increases above a threshold voltage, approximately 100 V. In Figure 34 the inverter start-up is shown. The three-phase rectified AC voltage “Rectifier_AC_In” is shown in red, the output of phase V is shown (V_Output) in blue. In yellow the DC-link voltage, in the schematic called “Rectifier_AC” is shown and one output voltage of the auxiliary power supply “V15_HV” is shown in green.

The auxiliary power supply starts working if the voltage “Rectifier_AC” is above 100 V. The voltage “V15_HV” is moving around the set point due to the light load conditions during start-up. The reason for this is pulse skipping of the auxiliary supply under light load.

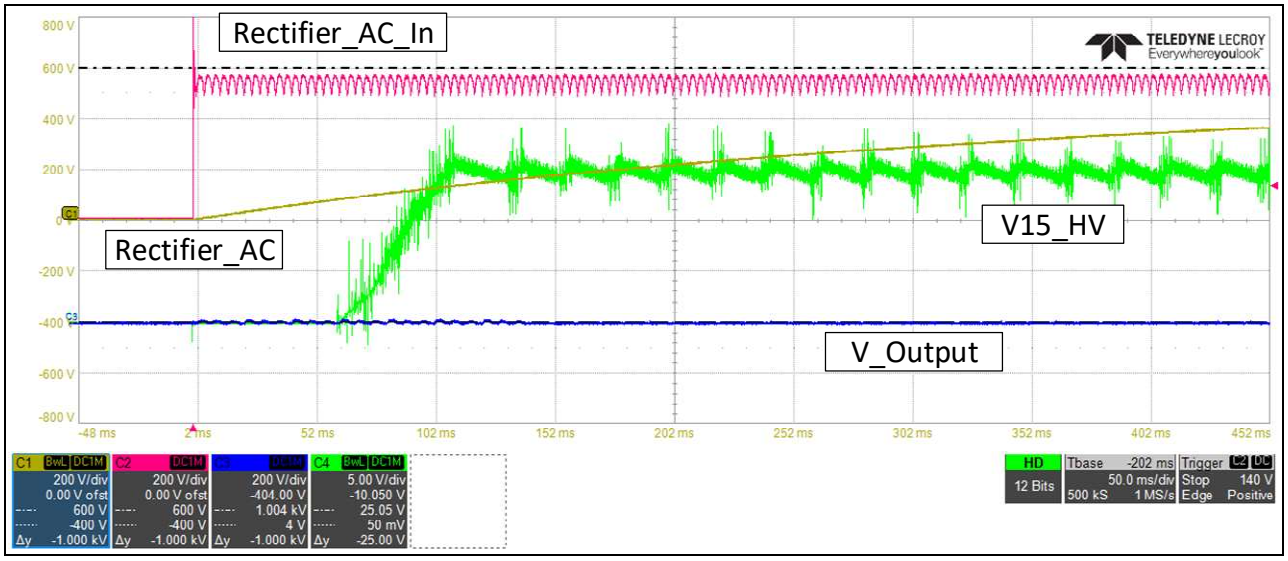


Figure 34 Inverter Start-up

If the inverter is connected to the grid, the in-rush current must be limited. Therefore, the DC-ink capacitors will be charged via the pre-charge relay and resistors. This relay will be closed after approximately 2 seconds, as shown in Figure 35. the pulse width modulation of the output phase starts approximately 200 ms after the relay closes.

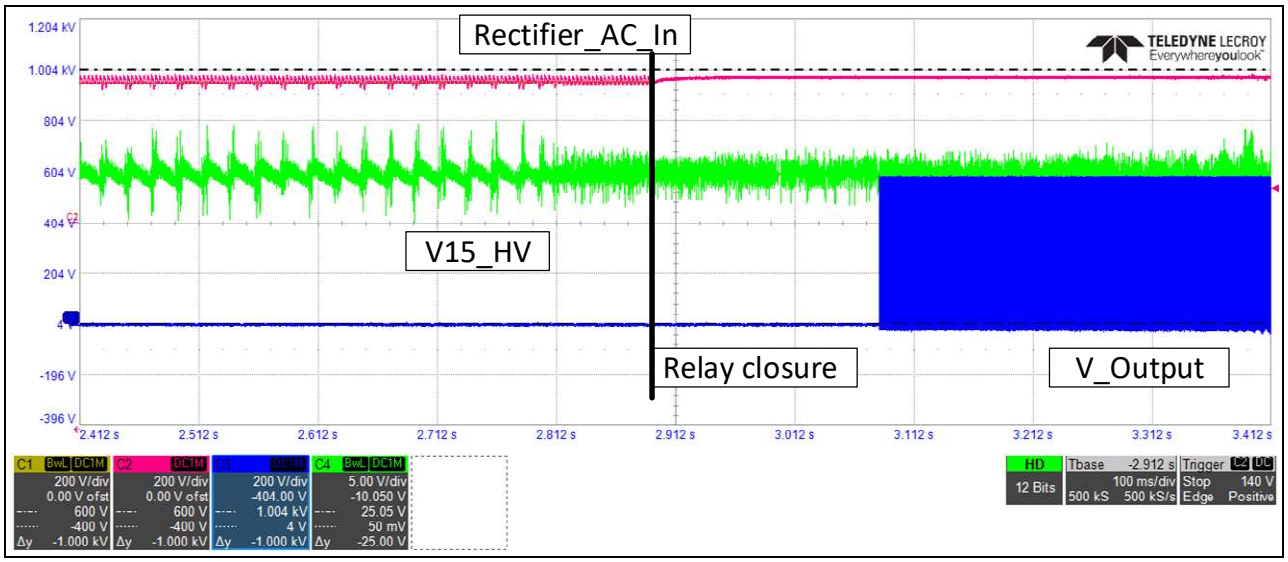


Figure 35 Relay closure

6.2 Operation under rated conditions

The following chapter shows the nominal operation conditions. The inverter is supplied by a 400 VAC grid and can be connected at the output to a symmetrical RL-passive load for test purposes rather than a motor load, as illustrated in Figure 36.

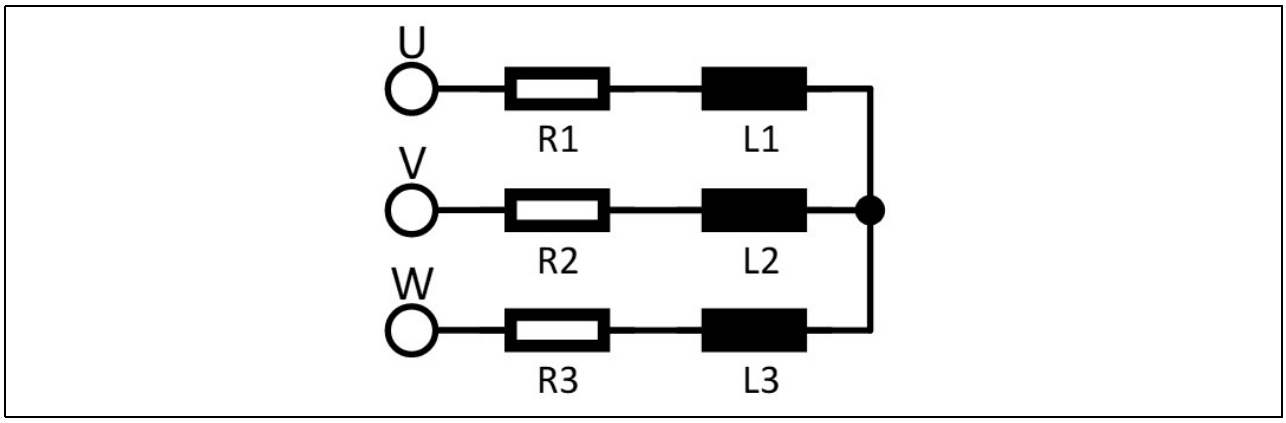


Figure 36 Passive load

The value of the inductor L1, L2 and L3 is 1 mH; the value of the resistor is 3.4 Ω. The sinusoidal output frequency is set to 50 Hz and output current to 45 $A_{RMS.}$ The switching frequency was set to 4 kHz. The measurement results are shown in Figure 37.

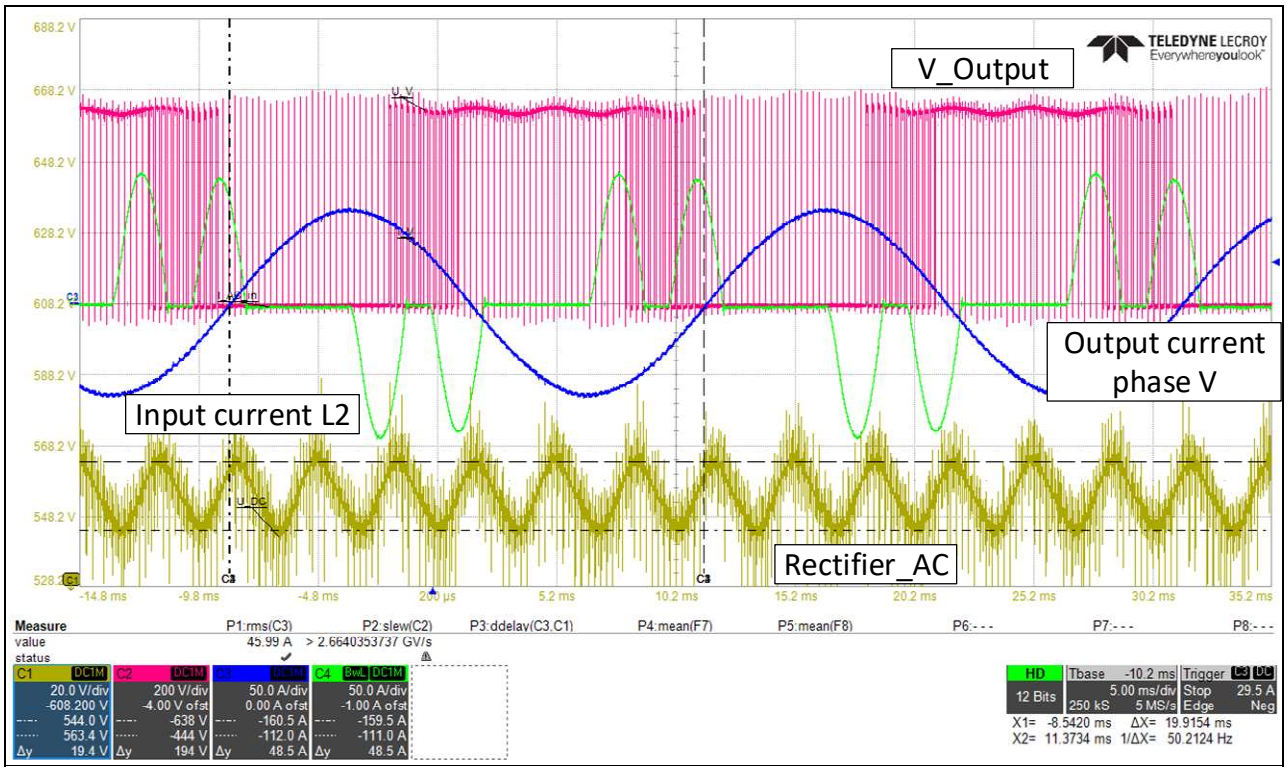


Figure 37 Inverter under nominal operation conditions

The output phase current (blue and the resistor voltage (red) are almost sinusoidal, whereas the input line current (green) shows the typical waveform of a capacitor charging current. The yellow signal shows the rectified DC-bus voltage of about 562 V.

6.3 Switching behavior of the IGBT – turn on/off

Figure 38 is the synopsis of all three modes of operation at 2 different current levels, 4.5 A r.m.s. (left column) and 50 A r.m.s. (right column). The number of events at a given switching speed is depicted for turn-on (top row) and turn-off (bottom row). The results of 2L-SRC and mode 1 are in principle similar, as 2L-SRC remains constantly in mode 1, as the change point of 35 A is never reached. While the faster switching mode 2 shows a clear peak at dvCE/dt of more than 6 V/ns, the slower switching mode 1 and the 2L-SRC mode have peak at lower than 2 V/ns. Both peaks at 4.5 A turn-on are active turn-on events.

The turn-off diagram at 4.5 A can be interpreted below 1 V/ns in such a way that the instantaneous phase current is high enough to fully charge up the output capacitance of the IGBT after its turn-off. The peaks at 2 V/ns and above 6 V/ns are caused by the active turn-on of the opposite switch in that half bridge. The phase current is not high enough to charge the output capacitance up to the DC-link voltage. Thus, the turn-on of the opposite switch pulls up the collector-emitter voltage. At high phase currents of 50 A r.m.s., the dvCE/dt covers a larger range, with high peaks at 1 to 1.5 V/ns for all three modes, and a second peak above 3 V/ns for 2L-SRC and mode 2. The same interpretation regarding the second peak is possible at the turn-off with 50 A r.m.s., but at a lower level. The phase current is usually at levels that are sufficient to charge the output capacitance and only a few events had been measured above 3 V/ns. [8]

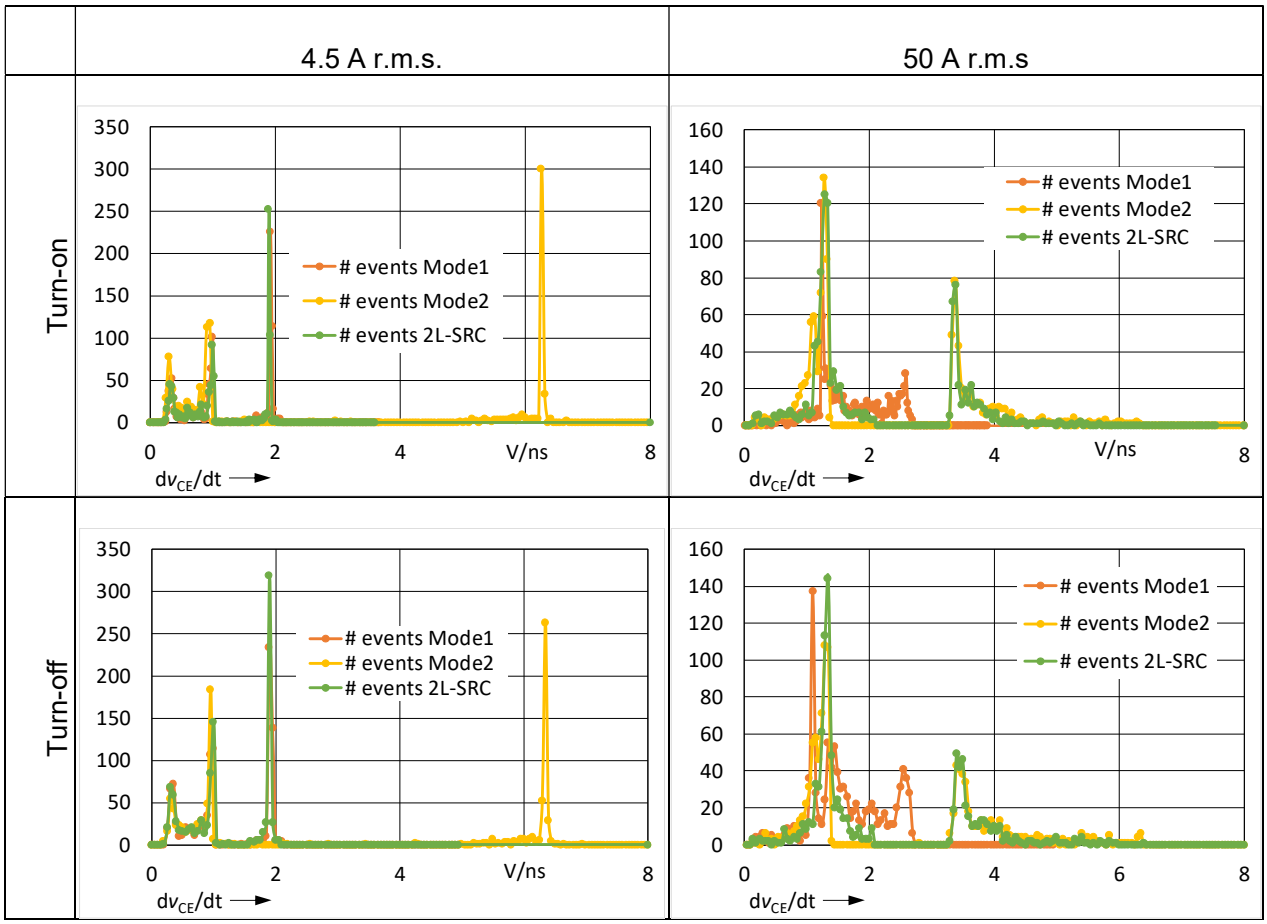


Figure 38 Switching events low side IGBT [8]

6.4 Thermal behavior of the inverter under nominal operation

Figure 38 and Figure 39 depicts the measured surface temperature of an IGBT and its freewheeling diode. The tests were performed with a thermocouple connected to the surface of the IGBT and diode in question. The fan speed was fixed at 80% of its maximum speed so that results would not be influenced by the fan's speed control. The solid lines represent the results during 2L-SRC operation mode, and the dashed lines represent the results of mode 1.

The graph illustrates clearly that both the IGBT and the diode have the lowest temperature at high phase currents. Therefore, the diode is more than 12°C colder during 2L-SRC mode than during mode 1, and the IGBT is more than 7°C colder than during mode 1. The dotted line represents the temperatures at operation with mode 2. The temperatures of the IGBT and its freewheeling diode are still higher in mode 2 compared to the operation in 2L-SRC mode.

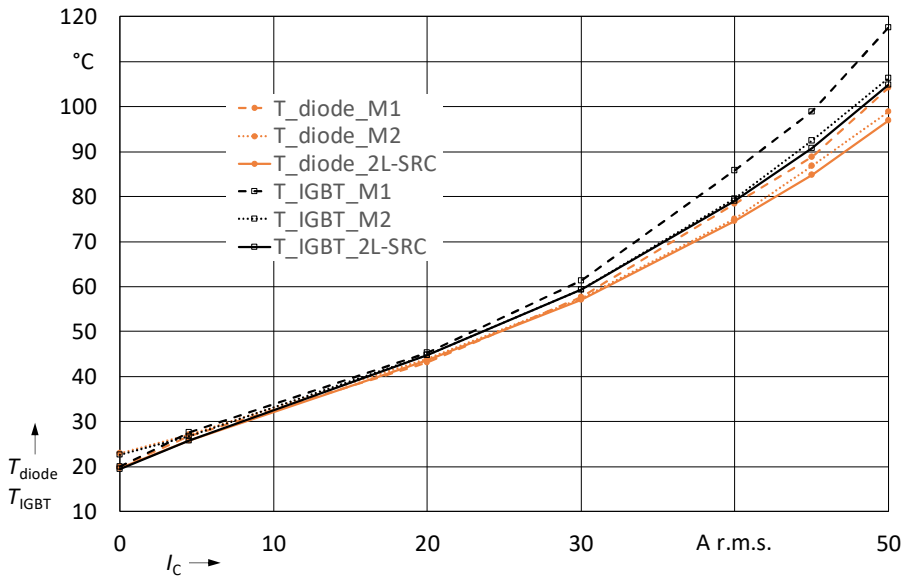


Figure 39 IGBT and diode temperature at high load [8]

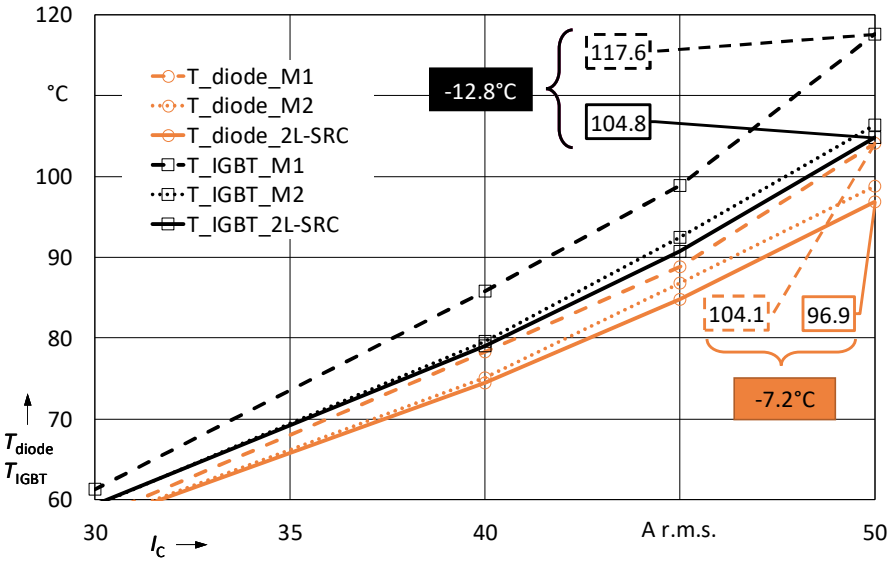


Figure 40 IGBT and diode temperature in detail at high load [8]

A typical drive overload pulse pattern is shown in Figure 41. In a period of 5 minutes, a 1-minute overload condition of 150% of the nominal current is required.

Table 30 Load cycle parameters

High-load interval	60 s
Low-load interval	240 s
Cycle period	300 s
Low-load phase current	25 A r.m.s.
High-load phase current	45 A r.m.s.

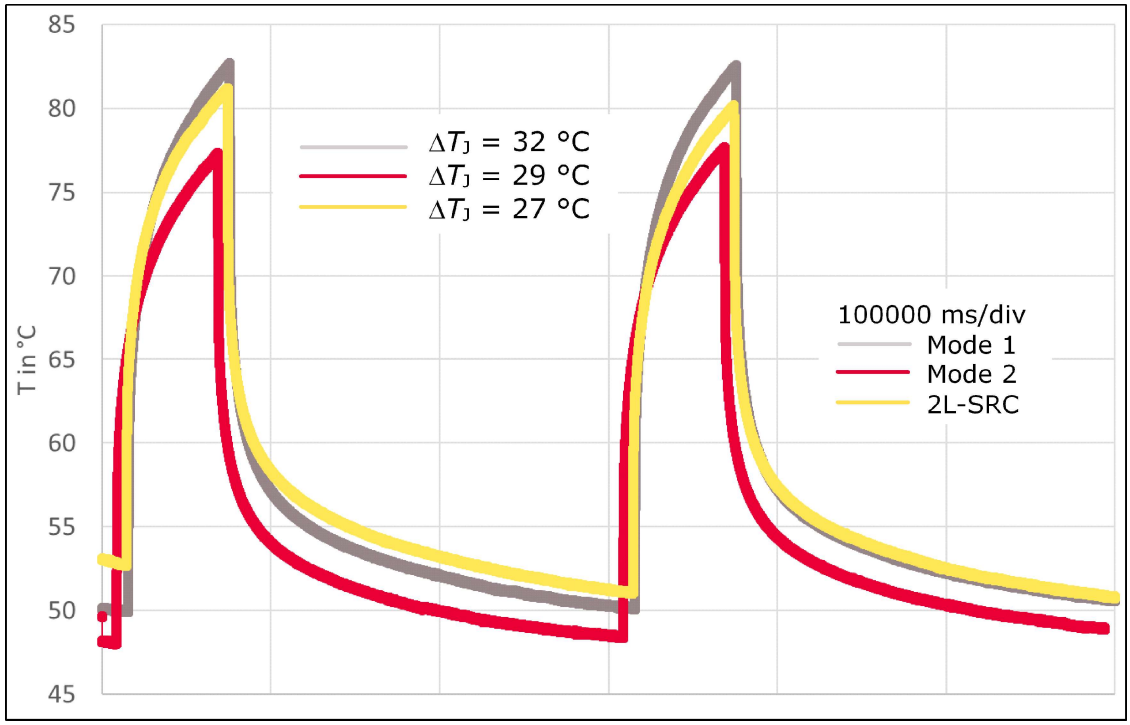


Figure 41 Temperature of the IGBT NTC and heatsink during heavy overload [8]

Figure 42 shows a temperature measurement of the IGBT and diode surface during the load cycling of the inverter. The cycling parameters are given in detail in Table 30. The load cycle lasts 300 seconds at a high-load to low-load ratio of 1:4, resulting in low-load intervals of 240 seconds and high-load intervals of 60 seconds.

It can be seen that the temperature difference for mode 1 is highest and reaches ΔTJ = 32 °C, because the slow turn-on switching at high current dissipates more energy compared to mode 2, which uses lower gate resistance for turn-on. Thus, mode 2 dissipates less energy during high load operation resulting in ΔTJ = 29 °C. Figure 41 depicts, again, that operating the inverter in 2L-SRC mode yields in the lowest temperature difference. 2L-SRC mode operates with higher gate resistance in low load, thus losses are slightly high during low load leading to a slightly higher temperature there. On the other side, 2L-SRC shows best performance during high load operation, therefore, the temperature difference is the lowest between low load and high load with ΔTJ = 27 °C. [8]

6.5 Short-circuit measurement

The short-circuit behavior of the drive was tested by using the setup, as shown in Figure 42. A short circuit was applied across the output terminals. The cable length between the terminal U and DC- is 1.2 m and has a cross section of 6 mm². The drive was powered up and ready for the short-circuit test.

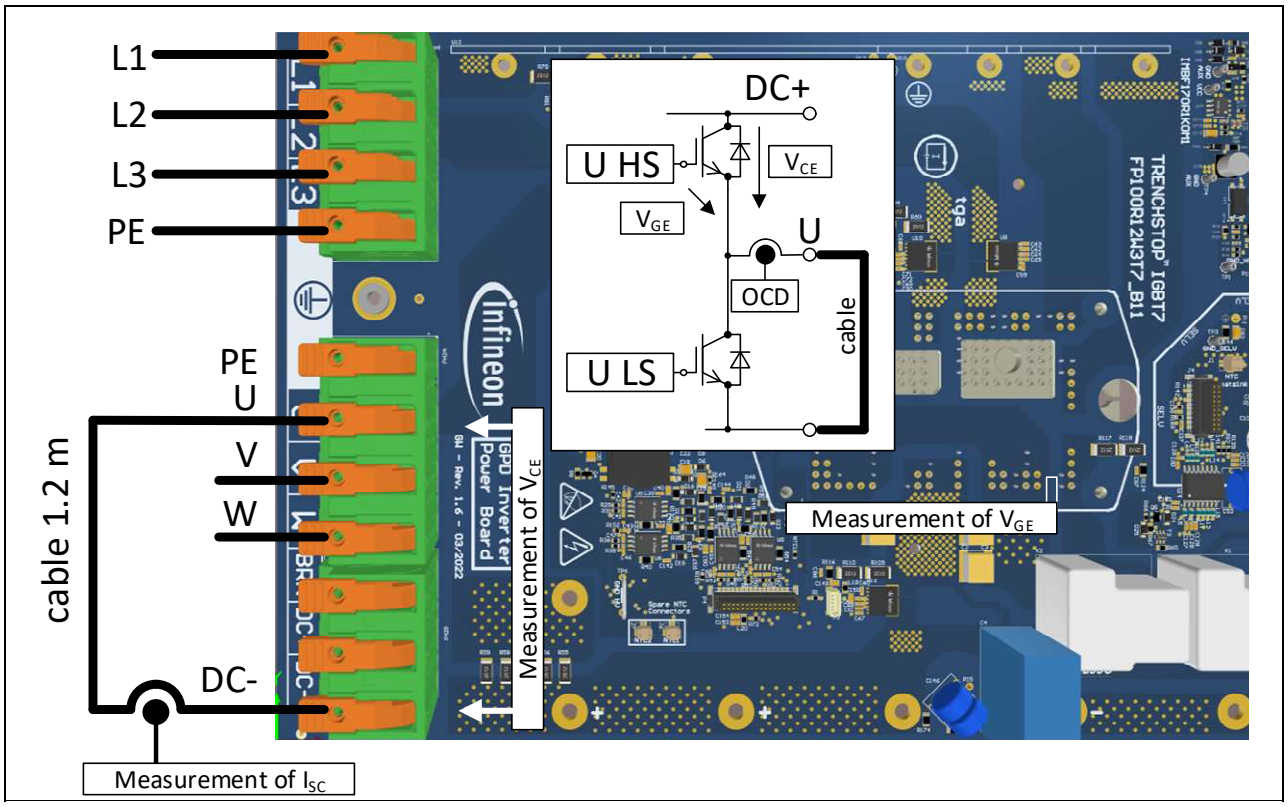


Figure 42 Short circuit setup

The short-circuit test can now be performed via the GUI. You will find the “Short Circuit Test Pulse” section at the “Config” tab. Here you can select the “Pulse length.” It is recommend to set the pulse length to 8 µs. Then you have to select the switch you want to test; in this case “U LS” (phase U, low side switch). To start the test click on “Fire.”

For verification if the short-circuit turn-off of the IGBT behaves correctly, the following voltages and currents were measured, see Figure 43. The gate emitter voltage (V_{GE}), the collector-emitter voltage (V_{CE}) and the current flowing through the short-circuit cable (I_{SC}) are shown in the figure.

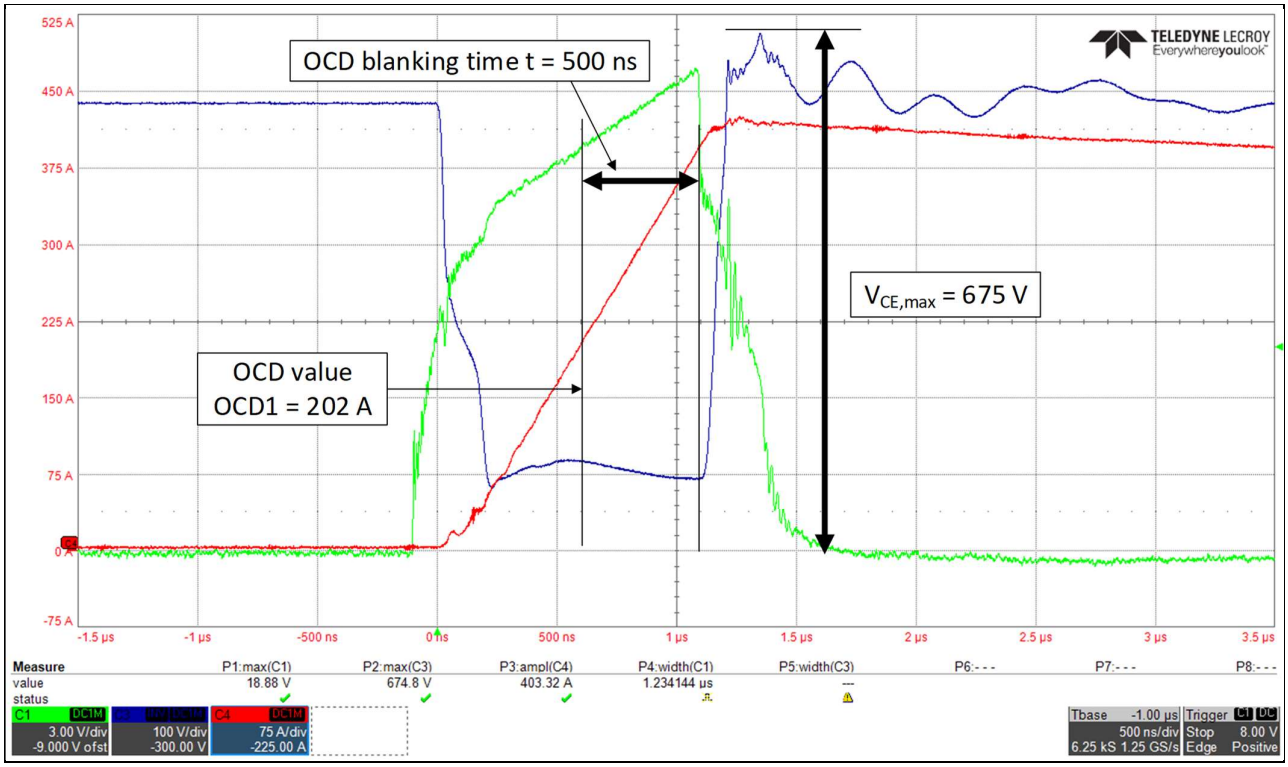


Figure 43 Short circuit measurement

The overcurrent detection (OCD) level of the current sensor is set to 168% of the nominal current measurement range of 120 A. Hence, the OCD current level is 202 A. The oscilloscope measurement in Figure 43 shows that the OCD signal of the current sensors triggers after 500 ns, then the IGBT is commanded to turn-off, and the current ISC declines. The overvoltage overshoot due to the di/dt is moderate with a maximum value of 675 V, hence an overvoltage of approximately 90 V. This is well within the IGBT 1200 V maximum rating.

7 References and appendices

7.1 Abbreviations and definitions

Table 31 Abbreviations

Abbreviation	Meaning
CE	Conformité Européenne
EMI	Electromagnetic interference
UL	Underwriters Laboratories
IGBT	insulated-gate bipolar transistor
SiC	silicon carbide
MOSFET	metal–oxide–semiconductor field-effect transistor
GUI	graphical user interface
PCB	Printed circuit board
SELV	safety extra-low voltage
FELV	functional extra-low voltage

7.2 References

[1] Infineon Technologies AG. AN2018-14 (2020): TRENCHSTOP™ 1200 V IGBT7 T7 Application Note. V1.2 www.infineon.com

[2] Infineon Technologies AG. AppNote TLI4971 Electrical Drive (2019): High Voltage General Purpose Drive (GPD) with Hall-Effect Current Sensor. V1.0 www.infineon.com

[3] Infineon Technologies AG. Datasheet (2020): FP100R12W3T7_B11 EasyPIM™ module. V0.2 www.infineon.com

[4] Infineon Technologies AG. Datasheet (2020): TLI4971 high precision coreless current sensor for industrial applications in 8x8mm SMD package. V1.1 www.infineon.com

[5] Infineon Technologies AG. Datasheet (2020): EiceDRIVER™ 1ED32xxMU12H Compact. V2.0 www.infineon.com

[6] Infineon Technologies AG. Datasheet (2020): IMBF170R1K0M1 CoolSiC™ 1700V SiC Trench MOSFET Silicon Carbide MOSFET. V2.1 www.infineon.com

[7] Infineon Technologies AG. Datasheet (2018): XMC4300 Microcontroller Series for Industrial Applications. V1.1 www.infineon.com

[8] Infineon Technologies AG. PCIM paper (2022): Two-level, slew-rate control reduces temperature stress of semicanductors in power modules. www.infineon.com

Revision history

Document version	Date of release	Description of changes
1.00	2021-01-29	First release
2.00	2022-11-17	Second release

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2022-11-17

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

UG-2020-28

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.