Tutorial n°3

GRID-TIED 3-PHASE SOLAR INVERTER

Written by: imperix SA, Rte. de l'Industrie 17, 1950 Sion, Switzerland
Nicolas Cherix <nicolas.cherix@imperix.ch>

Addressed topics:
- Use of the code libraries
- Implementation of a basic state machine
- Association with Typhoon HIL emulators

1 INTRODUCTION

This tutorial presents a possible control implementation for a three-phase grid-tied inverter using the BoomBox control platform. The considered system is depicted in Figure 1. Its main electrical parameters are shown in Table 1:

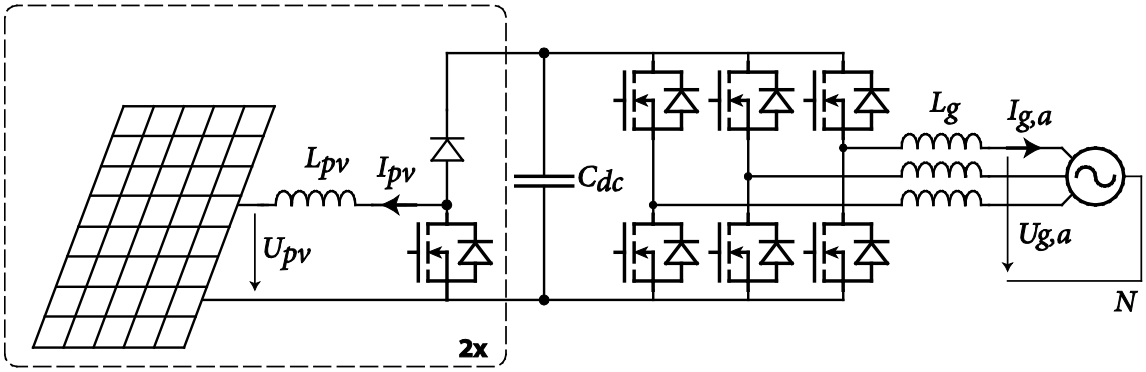

Figure 1 : Simplified electrical scheme of the considered system.

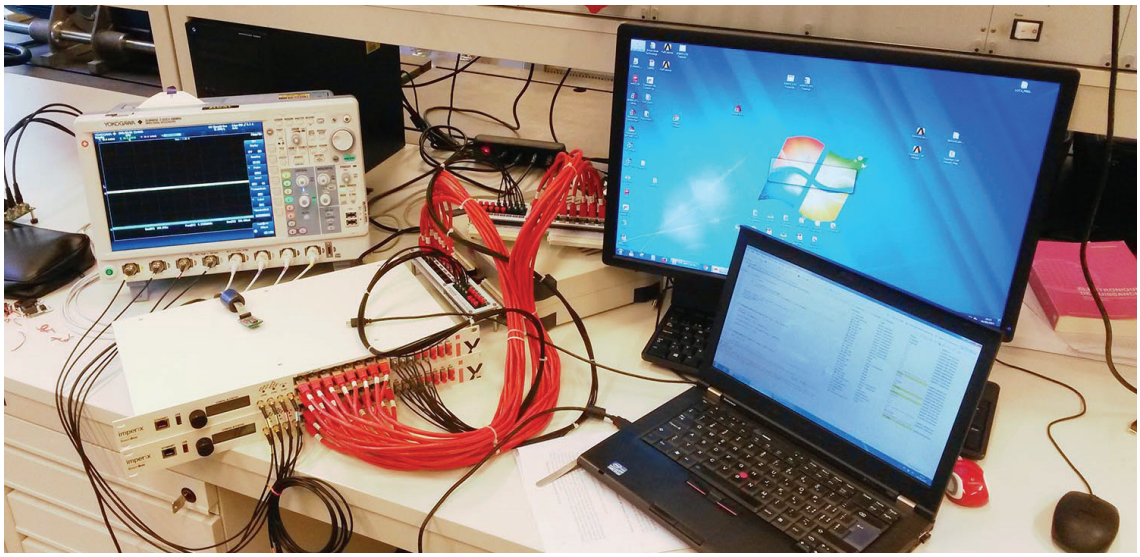

Figure 2 : Picture of a test setup using a Typhoon HIL400, emulating the power electronics converters. Only one BoomBox is utilized in the presented example.

Name	Nom. value	Specification	Employed sensor	Channel #
U_g	230 V_{RMS} (l-n)	Grid voltage (line-to-neutral)	IX-DIN800V	5, 6, 7
I_g	30 A_{RMS}	Current injected into the grid (phase)	IX-DIN50A	2, 3, 4
U_{DC}	650 V_{DC}	DC bus voltage	IX-DIN800V	10
$U_{PV1,2}$	350-650 V	Voltage(s) on the photovoltaic panels	IX-DIN800V	8, 9
$I_{PV1,2}$	20A	Current(s) exctacted from the PV panels	IX-DIN50A	0, 1
f_S	4 kHz	Switching frequency	N.A.	N.A.
Lg	4 mH	Filtering inductor	N.A.	N.A.
Lpv	4 mH	Filtering inductor	N.A.	N.A.

Table 1 : Electrical parameters of the studied system.

The proposed control strategy controls the grid current in a rotating reference frame (DQ-type) that is synchronized with the grid voltage. This synchronization is achieved by a software phase-locked loop (PLL). This conventional approach is well described in [1]. The equivalent scheme is shown in Figure 2:

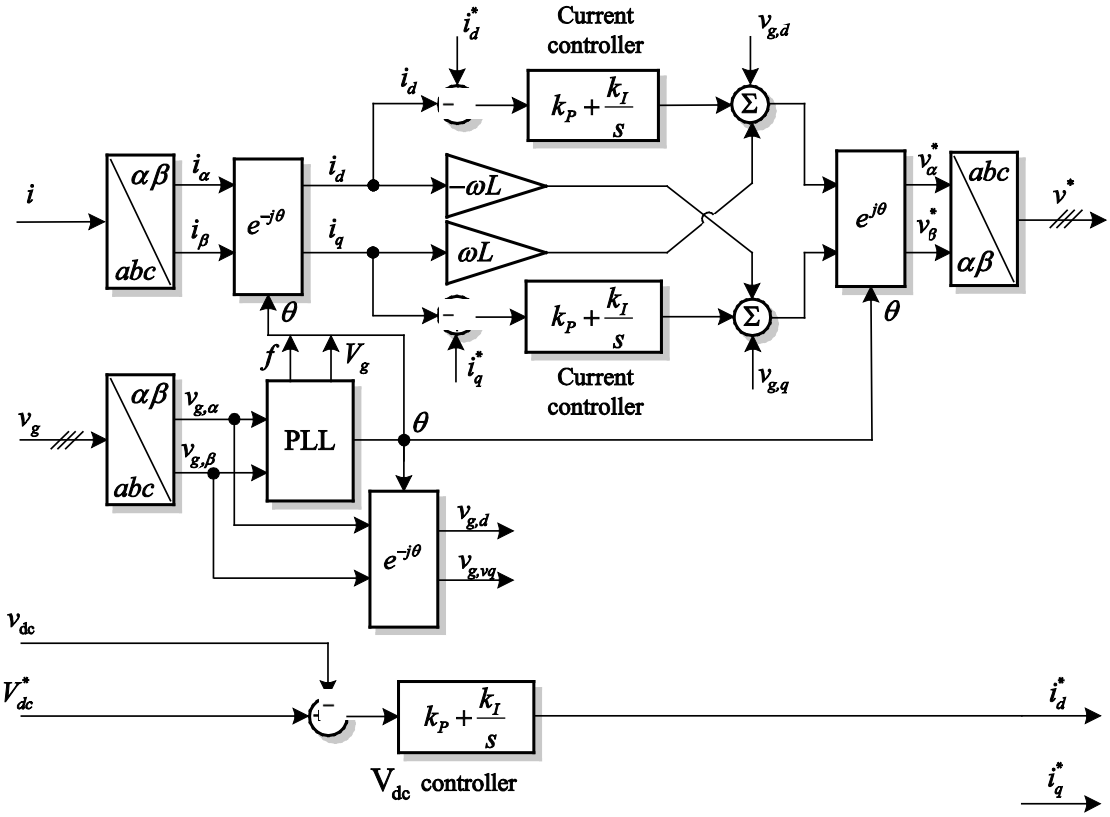

Figure 3 : Implemented control strategy for the three-phase inverter. Source [1].

Additionally, both photovoltaic strings rely on a PI-type current control, the two current setpoints being defined by a superposed Maximum Point Point Tracking (MPPT) algorithm that uses a perturb-and-observe approach.

In this example, all functional blocs constituting the control implementations are directly based on the pre-written control libraries existing in the BoomBox development environment. This shows that the control implementation of such a power conversion structure can be easily achieved, thanks to the BoomBox development environment.

2 HARDWARE SET-UP

When associated to an emulation device such as a Hardware In-The-Loop (HIL) simulator, the set-up is trivial. The BoomBox simply needs to be connected to the emulator. For Typhoon HIL devices, the necessary cabling interface is even available among the BoomBox accessories.

As presented in the previous tutorials, the BoomBox's analog inputs depends on the measurement sensors, which are here directly emulated by the HIL device. A possible configuration is given in Table 2. This approach guarantees a maximum compatibility with the final application by configuring the emulator so that the output gains for each measurement also emulates the final – and real – sensor. The configuration of the utilized Typhoon HIL 400 is shown in Table 3:

Type	# Channel Bbox	Sensor (fictive)	Sensitivity	Limits	Bbox gain	Bbox limits
Current	0 to 4	IX-DIN50A	100 [mV/A]	Various	2.0	Various
Voltage	5 to 10	IX-DIN800V	2.46 [mV/V]	Various	4.0	Various

Table 2 : Configuration parameters of the BoomBox (analog inputs).

Type	# Channel HIL	Sensitivity to emulate	Output gain of the emulator
Current	1 to 5	100 [mV/A]	÷ 10
Voltage	6 to 11	2.46 [mV/V]	÷ 407

Table 3 : Configuration parameters of the Typhoon HIL 400 emulator (analog outputs).

Note: The Typhoon HIL emulator HIL400 and HIL600 have their output voltage limited to ±5V, instead of ±10V for the newer generations HIL402 and HIL602. This particularity may impose additional restrictions that must be taken into account.

3 SOFTWARE CONFIGURATION

3.a CONFIGURATION OF THE TIME BASES

As in tutorial n°1, this application makes use of two different times bases, which are associated with two distinct interrupts and service routines:

1) A **fast** time basis, defining the switching, sampling and control frequency, here 4 kHz.
2) A **slow** time basis, related to the execution of the MPPT algorithm, here 250 Hz.

Subsequently, these two time bases are typically linked with service routines such as **UserInterrupt1()** and **UserInterrupt2()**, respectively. This approach is detailed hereafter:

```
SetFreqGenPeriod(0, (int)(SWITCHING_PERIOD/FPGA_CLK_PERIOD));
RegisterExt1Interrupt(&UserInterrupt1, 0, 0.0, 0);          // No post-scaling here

SetFreqGenPeriod(1, (int)(0.0005/FPGA_CLK_PERIOD));
RegisterExt2Interrupt(&UserInterrupt2, 1, 0.0, 4);          // Period post-scaled by 2x4=8
```

This example shows an important detail: indeed, as the ratio **0.0005/FPGA_CLK_PERIOD** is greater than 65'535, it is unfortunately impossible to program a time basis that is slower than 457 Hz on a ClockGen (16 bits counters). This motivates the configuration of a frequency of 2 kHz on the ClockGen #1, with a 8x post-division of the output clock.

3.b CONFIGURATION OF THE PULSE-WIDTH MODULATORS

As in the tutorials n°1 and n°2, the pulse-width modulators (PWMs) can be directly mapped to the fast time-base (ClockGen #0, see above). The corresponding code is as follows:

```
ConfigPWMChannel(0, 0, TRIANGLE, (int)(600e-9/FPGA_CLK_PERIOD));
...
ConfigPWMChannel(4, 0, TRIANGLE, (int)(600e-9/FPGA_CLK_PERIOD));
```

In this example, triangular carriers and 600µs deadtimes are implemented on all channels. Other configurations are of course possible. More information on this topic can be found in tutorial n°1.

Besides, in this example, the PWM channels are not immediately activated, since a finite state machine is responsible for managing the various operating modes. The corresponding activation and deactivation is directly managed by the **SetOpMode()** routine which can be easily understood from the code example.

3.c CONFIGURATION OF THE ANALOG-TO-DIGITAL CONVERSION

The AD conversion parameters can be configured as in tutorial n°1. This allows to directly exploit meaningful floating-point quantities at the user level. To this end, **SetADCAdjustements()** must be invoked during **UserInit()** similarly to the following code:

```
SetADCAdjustments(0, IX_DIN50A_GAIN/2.0, 0.0);
...
SetADCAdjustments(10, IX_DIN800V_GAIN/4.0, 0.0);
```

Finally, the sampling can be advantageously made in the middle of the switching period, i.e. at the exact instant when the current – including its ripple – is equal to its sliding-average value. To do so, the sampling is configured in the middle of the period defined by the ClockGen #0:

```
ConfigSampling(0, 0.5);
```

4 IMPLEMENTATION OF THE CONTROL APPLICATION

4.a DEFINITION OF THE USER STATE MACHINE

In this application, several operating modes are established. Thanks to the use of a HIL emulator here, these modes may – or may not – correspond to realistic operating regimes, or may only be applicable to debug conditions. In this example, five operating modes are implemented:

1) The **STANDBY** mode, corresponding to the idle state during which the entire power converter is completely inactive.
2) The mode **OPENLOOP**, during which no closed-loop control is achieved. On the contrary, the modulation indices of the inverter are generated directly such that they produce sinusoidal waveforms, irrespectively of any measurement. This mode may typically be useful in case of preliminary operation in a passive load and/or used for debug purposes.
3) The mode **CURRENTONLY**, which executes only the current control of the inverter. No control of the DC bus voltage is executed. This mode therefore requires that the HIL emulator simulates a fixed voltage on the DC bus (e.g. by disconnecting the PV strings and placing a constant voltage source on the bus).

4) The mode **BUSVOLTAGE**, running the grid current control and the bus voltage control (cascade implementation). Unlike **CURRENTONLY**, this mode is only applicable when the bus is capacitive and not directly tied to a constant voltage source.
5) The mode **TRACKING**, during which all closed-loop controls are executed. This mode therefore corresponds to all tasks achieved by **BUSVOLTAGE**, plus the control of the DC currents extracted from the PV strings.

Obviously, depending on the application, other operating modes could be defined, corresponding to other control strategies or different combinations of control mechanisms.

4.b Implementation of the closed-loop control

In this example, the implementation of the closed-loop control tasks follows the same principles as previously described in tutorials n°1 and n°2. It extensively relies on the functions and objects which are pre-written and available in the API folder.

4.c Definition of the main interrupt service routine

In this example, the following sequence of tasks is proposed for the main control interrupt. A first part is related to the control of the PV strings, while a second is dedicated to the inverter:

1) Retrieval of the measurements using **GetADC()**. The exact sampling instant is that configured earlier using **ConfigSampling()**.
2) Execution of the current control of both PV strings:

```
Epv1 = Upv1 - RunPIController(&Ipv1_reg, Ipv1_ref - Ipv1);
Epv2 = Upv2 - RunPIController(&Ipv2_reg, Ipv2_ref - Ipv2);
SetPWMDutyCycle(3, Epv1/Udc);
SetPWMDutyCycle(4, Epv2/Udc);
```

3) Computation of the instantaneous powers extracted by the PV strings. This will be used by the MPPT algorithms. An elementary low-pass filtering is also implemented:

```
#define k_iir_lpf 0.05
Ppv1 = k_iir_lpf* (Upv1*Ipv1) + (1.0-k_iir_lpf)* Ppv1;
Ppv2 = k_iir_lpf* (Upv2*Ipv2) + (1.0-k_iir_lpf)* Ppv2;
```

4) Execution of the PLL(s):

```
abc2DQ0(&Ug_dq0, &Ug_abc, theta);
theta = RunDQPLL(&DQPLL, &Ug_dq0) ;
```

5) Computation of the active power injected into the grid and the feedforward value of the corresponding current **Id_feedfoward**:

```
Id_feedforward = 2.0/3.0*(Ppv1 + Ppv2) / Ug_dq0.real;
```

6) Switch between the possible operating modes. During the **CURRENTONLY** operating mode, this section contains:
   a) Coordinate transformations.

b) Execution of control algorithms. In **CURRENTONLY** mode, this mainly involves the execution of the grid current controllers. In this example, the maximum modulation depth is augmented by injecting a third harmonic on the zero-sequence converter EMF:

```
Eg_dq0.real =    Ug_dq0.real
                 + RunPIController(&Id_reg, Ig_dq0_ref.real-Ig_dq0.real)
                 - OMEGA*LGRID*Ig_dq0.imaginary;
Eg_dq0.imaginary = Ug_dq0.imaginary
                 + RunPIController(&Iq_reg, Ig_dq0_ref.imaginary-Ig_dq0.imaginary)
                 + OMEGA*LGRID*Ig_dq0.real;
Eg_dq0.offset = -1.0/6.0*Eg_dq0.real*cos(3*theta);
```

c) Inverse coordinate transformations.

d) Computation and update of the modulation indices :

```
SetPWMDutyCycle(0, 0.5 + Eg_abc.A/Udc);
SetPWMDutyCycle(1, 0.5 + Eg_abc.B/Udc);
SetPWMDutyCycle(2, 0.5 + Eg_abc.C/Udc);
```

7) When the operating mode is defined to **BUSVOLTAGE**, the bus voltage control is also executed just before the grid current control. The following additional code is therefore also executed (there is no **break** statement at the end of the case):

```
Ig_dq0_ref.real = Id_feedforward - RunPIController(&Udc_reg, Udc_ref - Udc);
```

8) Sending of the updated modulation indices to the FPGA-implemented modulators:

```
UpdatePWMData();
```

4.d DEFINITION OF THE SECONDARY INTERRUPT SERVICE ROUTINE

A possible sequence of tasks for the secondary control interrupt is as follows. It essentially contains the execution of the MPPT algorithm as well as the main state machine:

1) When the MPP tracking is active, the current setpoints for the two PV strings are determined by the MPPT algorithm. Otherwise, this choice is left free to the user (the setpoints remain undefined):

```
if (enable_MPPT==1){
        Ipv1_ref = RunMPPTracking(&string1_mppt, Ipv1, Ppv1);
        Ipv2_ref = RunMPPTracking(&string2_mppt, Ipv2, Ppv2);
}
```

2) Execution of the transition between the various operating modes. As shown in the code example, the corresponding function call also manages the activation / deactivation of the PWM channels:

```
SetOpMode(next_opmode);
```

It is worth noting here that no tests are made to evaluate possible transition conditions between the operating modes. In this example, these transitions are only meant to be triggered by the user through the command-line interface. The following section specifically shows how this can be handled.

4.e DEFINITION OF THE COMMANDS THAT ARE AVAILABLE TO THE USER

In addition to the blocking and release of the PWM signals (commands **enable/disable**), the command-line access of the BoomBox allows the user to freely define numerous actions. The definition of these actions is made in the **cli_commands.c** file.

For the present example, the two following commands are implemented:

a) **setmppt**, which is meant to activate or deactivate the MPPT algorithm (the command **setmppt 0** disables the MPPT, while the command **setmppt 1** activates it).
b) **mode**, followed by the name of the operating mode, which allows to force the change between the various operating modes.

The configuration and principle of operation of these commands is described in the tutorial n°1.

5 REFERENCES

[1] R. Teodorescu, M. Liserre and P. Rodríguez, and P. C. Loh, *"Grid converters for photovoltaic and wind power systems,"* Wiley, 2011, 407p.