UNIVERSITY OF CALIFORNIA
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS 249
Fall 2012

A. Sangiovanni-Vincentelli
P. Nuzzo
J. Jensen

Lab 3: Design of a Digital Switched-Mode Power Supply
Due Tuesday, November 29, 2012

Phase 2: Phase Locked Loop and Power Control Design

Lab 3 focuses on system-level design of a digitally controlled switched-mode power supply (SMPS) and its prototyping on an industrial-strength platform, using LabVIEW and Hardware-In-The-Loop (HIL) validation. The LabVIEW code and a few useful references for this lab have been posted on Piazza. The lab is organized in two phases. The assignment for Phase 1 can be found on the class website, including an introduction to LabVIEW and exercises to gain familiarity with the sbRIO Processor and FPGA programming environments. We provide details on Phase 2 below. At the end of this lab, you are required to write a brief report documenting your design methodology and choices as well as your learning experience with LabVIEW.

Exercise 1: Design of a Software Phase Locked Loop

The goal of this exercise is to model, implement and test a Phase Locked Loop (PLL) sub-system for FPGA control applications of 3-phase power systems. Such a PLL must track the phase and frequency of a reference input signal to which it locks. In our case, the PLL receives as a reference input the grid voltage (a 60-Hz sinusoidal voltage signal) and generate 3 output sinusoidal signals, all locked at the same frequency as the reference signal, and such that any two of them have a phase difference of 120 degrees ($2\pi/3$ radians). The output phases of the PLL will be expressed in π radians in our setup. To deploy the PLL on the sbRIO FPGA, the PLL code should be part of the `FPGA Grid PLL.vi` block diagram of the `Power Systems.lvproj` project. However, to save time on multiple FPGA compilations, make sure you prototype your design first on your desktop, and then deploy it to the FPGA. In fact, within the `Host Grid PLL Simulator.vi` you can simulate the 60-Hz wall signal and your PLL on the desktop and use the same fixed-point calculations you would use on the FPGA. Note that the simulation is accurate with respect to timing and fixed-point quantization error, however it does not run in real-time.

While you can find more PLL implementation details in the reference papers [1-4] below (which are also posted on Piazza), we recall here the discrete-time PLL linear model in the z-domain, which is shown in Fig. 1.

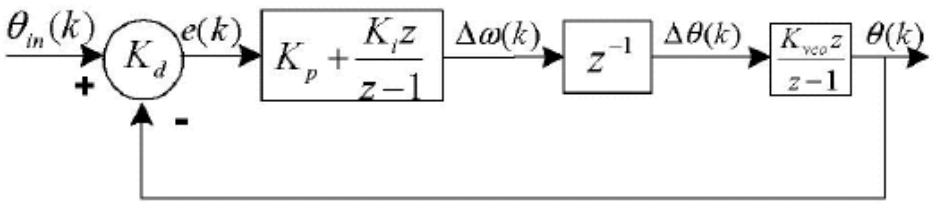

Fig. 1: Linear discrete software PLL model from [1].

The PLL can be seen as a linear system in the phase domain of the reference and output signals. Each sample of the error signal $e(k)$ between the reference phase $\theta_{in}(k)$ and the PLL output phase $\theta(k)$ is fed into a proportional-integral (PI) controller and used to calculate the change in angular frequency of the main voltage $\Delta\omega(k)$. Multiplying $\Delta\omega(k)$ by the sampling time, which is represented by the unit delay z^{-1}, determines the sampling time increment of the phase angle $\Delta\theta(k)$. By integrating the increment $\Delta\theta(k)$ over the range 0-2π, the estimated phase angle $\theta(k)$ is then obtained. The estimated angle is fed back into the phase detectors and used to adjust the error signal towards zero, resulting in the estimated angle and the grid phase angle being the same. The output of your PLL is an ideal sine wave of magnitude 1 whose phase and frequency are locked to the grid, and two additional ideal sine waves shifted 120 degrees and 240 degrees, respectively. The ideal signals are passed into a space vector modulator that modulates the ideal sine wave signals with a 4 kHz PWM signal, which is fed into the circuit below.

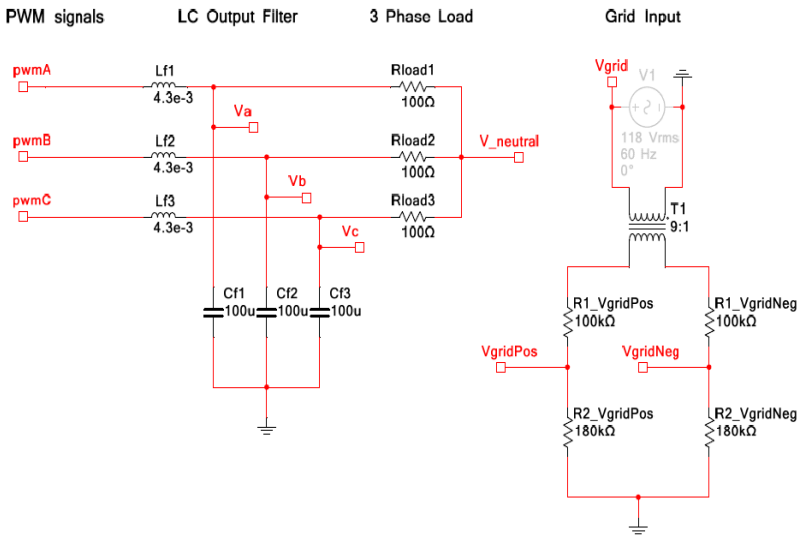

Fig. 2: Circuit schematic for the 3-phase output filter and load (left side) and the circuit used to convert the grid voltage levels to voltage levels within the range supported by the FPGA.

The right side of Fig. 2 shows the circuit used to convert the grid voltage levels from the wall to levels that are compatible with the sbRIO FPGA, hence by your PLL. The left side of Fig. 2 shows the 3-phase output filter and load that you implemented on the breadboard during Phase 1. Here are a few guidelines to proceed with your design:

- To simplify the design, you can assume that the system parameters K_d (phase detector gain) and K_{vco} (voltage-controlled oscillator gain) are both equal to 1. Moreover, you are free to choose the values of the proportional and integral gains K_p and K_i of the PI controller in the loop to achieve stable closed-loop dynamics. You may find the following references useful:
  [1] W. S. Phipps, et al., "A Novel Three-Phase Software Phase-Locked Loop," EPE-PEMC 2006.
  [2] S.-K.Chung, "Phase-locked loop for grid-connected three-phase power conversion systems," IEE Proc. Electr. Power Appl., vol. 147, no. 3, May 2000.
  [3] D. Hu, "FPGA-Based Digital Phase-Locked Loop Analysis and Implementation," Master Thesis, UIUC, 2011.
  [4] W. Li, et al., "Introduction to Phase-Lock Loop System Modeling," Tech. Rep., Texas Instruments.
  [5] Lacaita, et al., Integrated Frequency Synthesizers for Wireless Systems, Cambridge University Press, 2007 (Recommended as a general reference on PLL concepts, but it does not cover details and implementation of software PLLs).
- There are a few ways to implement the phase detector (PD) block, which is represented as a subtraction in the linear model of Fig. 1. One example is provided by the "digital" implementation shown in Fig. 3 below (taken from [5]). The output of the XNOR gate is only affected by the zero crossing of the input signal, which is why the inputs are represented as logic square waves. In (b) the PD characteristic is shown obtained by averaging (low-pass filtering) the OUT signal.

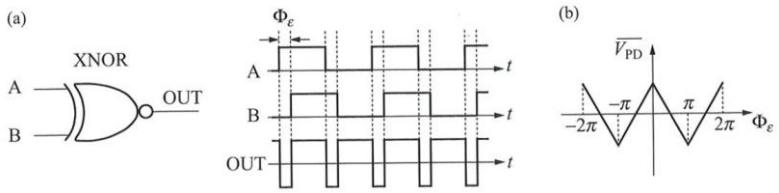

Fig. 3: XNOR PD: (a) schematic, (b) static characteristic.

- You are allowed to use any pre-built block from the LabVIEW libraries for your PLL. For an example of a single-channel PID implementation on an FPGA target, you can refer to the `Using Discrete PID - cRIO` project in the directory: labview\examples\control\pid\fpga.llb\CompactRIO
- The inputs and outputs in the FPGA PLL loop adopt a fixed-point configuration, which you are recommended to use in your design. Here are two further references you may find useful:

  http://zone.ni.com/reference/en-XX/help/371361H-01/lvconcepts/numeric_data/#Using_FXP

  http://zone.ni.com/devzone/cda/pub/p/id/363

(BONUS) While the output phase of your PLL is practically aligned to the grid phase (after locking), the three phases *Va*, *Vb* and *Vc* at the load will have a phase shift due to the presence of the LC filter. Such a phase shift is clearly filter and load-dependent and it is undesirable for several applications. However, tracking and compensating for such a phase shift is not deemed as very critical from a timing standpoint, since variations in the load (e.g. due to temperature variations) are typically characterized by slow dynamics. Design, implement and test a control algorithm that is capable to track the phase shift between the load and the grid voltage and compensate for it by adequately shifting the angle of your PLL reference signal. Which computational resources on the sbRIO would you use to deploy such a controller? Motivate your answer (*Hint:* you are basically required to build a programmable delay to pre-process the PLL reference signal).

Exercise 2: Design of the Power Control Algorithm

The last step of Phase 2 is to model, implement and test the actual power control algorithm for the 3-phase systems. The controller receives as a reference input the desired power to be delivered to the load (in mW). It compares the desired power with the actual measured power and adequately sets the pulse width modulation (PWM) index to null the difference between them. This time, you will deploy the controller on the sbRIO Processor, and your code should be part of the `RT 3-Phase Power.vi` block diagram of the `Power Systems.lvproj` project.

Please refer to the references posted on Piazza for further details on PWM. The PWM modulation index is a real number in the [0,2] range. The core architecture for your design will be a proportional-integral-derivative (PID) controller. As for Exercise 1, you are allowed to use any pre-built block from the LabVIEW libraries to implement the controller. You are free to choose the proportional gain, the integral and derivative times for the closed-loop dynamics to be stable. By default your control algorithm executes periodically within a Timed Loop structure at a frequency of 3 kHz. The period of the loop (333 us) is provided by the "Period" node in the power control loop.