Lab 6: Digital Phase Locked Loop (PLL): DSP Implementation

Objective

In this assignment, you will

- Code the PLL in C for the DSP board
- Experiment with frequency doubling, halving

Pre-lab

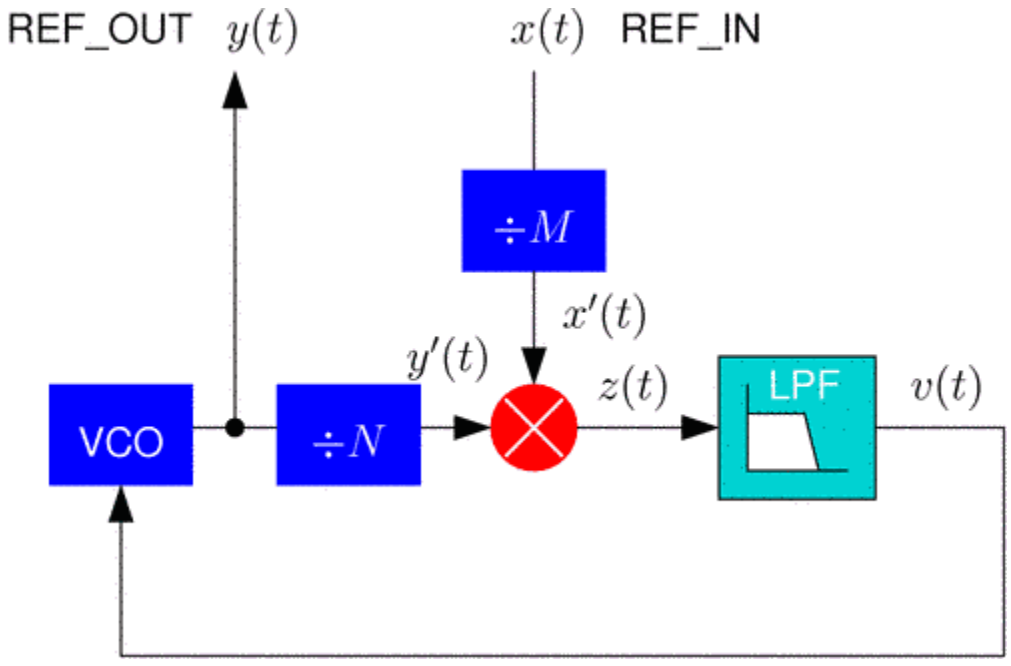
For the pre-lab, learn the background as usual and look at what is required in the "Pre-Lab Exercise" below.

Background

The purpose of this lab is to code your PLL in C. Also, in the upcoming labs for performing carrier recovery and symbol timing recovery, you will need to be able to generate harmonics and sub-harmonics of the frequency which you lock to.

Generating Harmonics, Sub Harmonics

In the diagram below, the frequency division operators allow you to generate rational harmonics and sub-harmonics of the incoming reference:

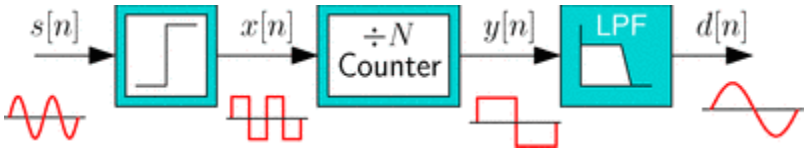


This can be easily understood by considering that the purpose of the PLL is to force the two signals coming into the phase comparator (the multiply operation) to have the same frequency. This means that

$$f_{y'} = f_{x'} \Rightarrow \frac{f_y}{N} = \frac{f_x}{M} \Rightarrow f_y = \frac{N}{M} f_x$$

where *f* indicates the frequency of the subscripted signal.

One way to accomplish these frequency divisions in C code is to operate on the sine wave directly by using the operations depicted below:



Here, the sine wave is converted to a digital signal using a sign() operation, which is then fed to a digital divide-by-N counter. Although the digital square wave *y[n]* is sufficient in many applications (such as symbol timing recovery), if a sine wave is again needed, this can be obtained by low-pass filtering.

For some applications where a sine wave (or other complex shape) is needed (like for carrier recovery), we can just use the lookup tables we had before in a clever way. Remember before that we had an operation like

$$accum \leftarrow accum + f - \frac{k}{2\pi} v[n]$$

and the output was generated as

```
s = sin_table[(int)((float)SIN_TABLE_SIZE*accum)];
```

If *s* is locked to an incoming signal, a doubled frequency signal *s2* could be generated by using a second lookup table that contains two sinusoidal cycles, or

```
s2 = sin_table2[(int)((float)SIN_TABLE_SIZE*accum)];
```

A signal with half the frequency could be obtained in a similar way by reversing the roles of the two lookup tables, or locking the signal to the table with two cycles and outputting the signal from the table with a single period.

Now you should realize that only a single lookup table is actually needed, by just doing different modulo and multiplication operations on the accumulator. Can you figure out how?

Pre-Lab Exercise

To help you get started with the lab please download the project pll_student_prj.zip. This will serve as a skeleton for your PLL C code. You should make sure you understand the following components of that code:

- Sine table initialization in sin_tables.c
- Members of the PLL state structure in pll.h
- Allocation of PLL state structure in pll_init() in pll.c
- Processing of a single buffer of samples in pll() in pll.c
- Initialization and processing in dsp_ap.c

While becoming familiar with the code, you should have noticed some things that are missing. You will need to fill in the missing details in dsp_ap.c and pll.c to get a working PLL.

In addition to the background material, we may ask you questions about functions in the `pll_student_prj.zip`, so please try to understand it before coming th to the lab.

Laboratory Exercise

In the lab, you should extend your code so that depending on the DIP switches, it generates double and half frequency versions of the output, both of which are synchronized to the incoming sine wave. We will use this later when we do carrier recovery.

Hint: You can read DIP switches 0-2 to form a binary number 0 through 7 using:

```
int_value = (USER_REG >> 4) & 0x7;
```

You could then check this value to decide whether to output the fundamental, half, or double frequency.

Check-Off

Demonstrate to the TA or instructor that the DSP-based PLL is working correctly. Your PLL should be able to handle changes in frequency of at least plus or minus 10% of the nominal reference frequency of 800 Hz (probably it will handle even more variation).

Demonstrate that you can also generate a synchronized signal with double or half the frequency as well.

Lab Write Up

Include the following items in your final write-up:

1. A printout of your final C implantation of the PLL. You can just print out the pll_init() and pll() functions (you do not have to print all of the surrounding `dap_ap.c` code!).
2. A description of how you tested your PLL. Indicate at what frequencies your PLL is no longer able to track.
3. A screen shot (like from the Scope program) showing that the PLL is tracking your input signal.
4. An explanation of any problems you ran into in the design and coding and how you solved them.