PLL Simulation in Matlab

DSP lab Spring 2013

9/4/2013

Jacobs University Bremen

Joshan Chaudhary, Grishma Raj Pandeya

**Prelab Questions:**

1. **What are the two functions of a PLL in a communications system?**

The 2 important functions are used for synchronization tasks:

-carrier recovery, which involves synchronizing the local oscillator (LO) to the incoming signal

-symbol timing recovery, or properly aligning the sample times at the matched filter output.

1. **What are the key components (operations) used to implement a PLL?**

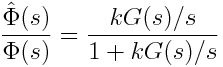
**Multiplier**: takes product of local reference from VCO and input reference

**LPF**: removes doubled frequency component from output of multiplier

**VCO:** low frequency component from multiplier passes through the filter and produces a slow varying voltage which is fed to the VCO. The VCO then controls the speed of the oscillator depending on whether y(t) lags/leads x(t). Feeding this phase difference to the voltage controlled oscillator (VCO) causes the oscillator to speed up when *y*(*t*) is lagging *x*(*t*) or to slow down when *y*(*t*) is leading *x*(*t*).

1. **When we write the second-order frequency response of the PLL, this expression relates the input and output \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of the references.**

phase



1. **What does the corner frequency of the PLL loop filter control?**

The corner frequency of the PLL loop filter controls how fast the loop adapts to phase changes.

1. **How should the loop filter corner frequency compare to the input reference frequency?**

The corner frequency ω0 should be much lower ( <0.1) than the input reference frequency or clock being tracked.

1. **What does it mean for a PLL to track?**

The PLL is said to track when the phase of input reference signal and the output reference signal is said to reach proper alignment, which brings the control voltage to the VCO to be close to 0.

1. **What does it mean for the PLL to be overdamped or underdamped?**

A PLL is said to be **overdamped** if it requires a long time to adapt to phase changes while it is **under-damped** when it has the characteristics of fast response but a tendency to overshoot a target leading to oscillation.

1. **Why is an accumulator useful for a PLL implementation?**

The accumulator is used because of instead of having an incrementing time variable t and a separate phase phi, we have a single real number, known as the accumulator, that keeps track of the current phase. At each time step, the contents of the accumulator tell us the current position in the sin () lookup table, where the lookup table holds just one sinusoidal cycle.

1. **Why do we need to sample sin() in our lookup table more finely than at the normal sample rate of the system?**

sin() has to be sampled more finely as the PLL often needs samples which lie in between the samples spaced by the sample period and the phase can change continuously to track the input.

1. **How do we keep our accumulator in the range [0,1]?**

The accumulator is kept in range by wrapping around once the value of the accumulator exceeds 1. This is done using accum = accum – floor(accum).

1. **What is the point of storing and restoring the state of the PLL for each block?**

The state of the PLL is stored and restored after every block in order to store the current estimate of the phase and amplitude. This allows better estimates for the following samples.

1. **How do we handle signals with arbitrary amplitude?**

The block of samples has to be scaled so that it has approximately unit amplitude. This is done by averaging the magnitude of the samples. After each block the amplitude estimate is stored and used to scale the next block of samples so that the amplitude of the scaled signal is approximately 1.

**Lab Write Up:**

1. **A paper design of your PLL, showing the important parameters that are needed for implementation.**

A paper design of out PLL implementation is given below:

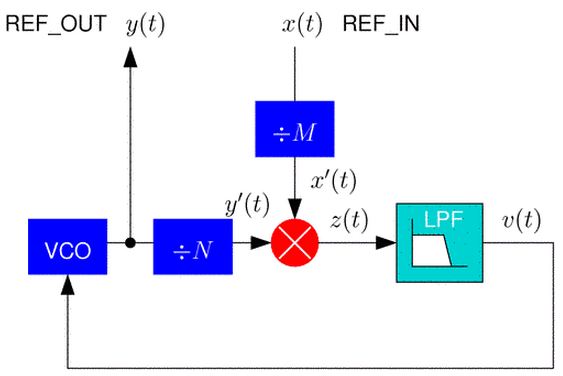


Fig: PLL design

We can see the basic PLL design is comprises of multipliers, a low pass filter and a Voltage Control Oscillator (VCO). The input reference signal x(t) is multiplied with a local reference signal y(t). The result is then passed through a low pass filter where the doubled frequency component is removed. Now, we get the difference between the phase of the input and the reference signal and send it to the VCO. Feeding this phase difference to the voltage controlled oscillator (VCO) causes the oscillator to speed up when y(t) is lagging x(t) or to slow down when y(t) is leading x(t). When the phases reach proper alignment, the control voltage to the VCO will be close to 0, which means the loop is tracking.

1. **A printout of your final MATLAB implementation of the PLL.**

Given below is the Matlab implementation of the PLL.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

pll\_init()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function [state] = pll\_init(f,D,k,w0)

% [state] = pll\_init(h, Ns);

%

% Creates a new pll filter.

%

% Inputs:

% h Filter taps

% Ns Number of samples processed per block

% Outputs:

% state Initial state

%% 1. Save parameters

% Store the filter taps

% Store the number of samples processed per block

%% 2. Create state variables

state.f = f;

state.D = D;

state.k = k;

state.w0 = 2\*pi/100;

state.T = 1;

state.x\_in\_old = 0;

state.y\_out\_old = 0;

state.z\_old = 0;

state.v\_old = 0;

state.accum = 0;

tau1 = k/(w0\*w0);

tau2 = 2\*D/w0-1/k;

state.a1=-(state.T-2\*tau1)/(state.T+2\*tau1);

state.b0=(state.T+2\*tau2)/(state.T+2\*tau1);

state.b1=(state.T-2\*tau2)/(state.T+2\*tau1);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Pll()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

function[state\_out,y] = pll(state\_in,x)

% pll.m

%

% Script to set the PLL filter.

% Global parameters

s = state\_in;

% Generate my input sin(2\*pi\*fi\*n).

% input frequency

s = state\_in;

N = length(x);

k = s.k;

a1 = s.a1;

b1 = s.b1;

b0 = s.b0;

f = s.f;

accum = s.accum;

z(1) = x(1) \* s.y\_out\_old;

%Low pass filter

v(1) = a1 \* s.v\_old + b0 \* z(1) + b1 \* s.z\_old;

%Compute accumulator

accum = accum + f - k/2/pi\*v(1);

accum = accum - floor(accum);

%Get y\_out sinusodal

y(1) = sin (2\*pi\*accum);

for n = 2:N,

%Multiply input x with y

z(n) = x(n) \* y(n-1);

%Low pass filter

v(n) = a1 \* v(n - 1) + b0 \* z(n) + b1 \* z(n - 1);

%Compute accumulator

accum = accum + f - k/2/pi\*v(n);

accum = accum - floor(accum);

%Get y\_out sinusodal

y(n) = sin (2\*pi\*accum);

end

%Save old values

s.accum = accum;

s.x\_in\_old = x(n);

s.y\_out\_old = y(n);

s.v\_old = v(n);

s.z\_old = z(n);

state\_out = s;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

test\_pll()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

clc

clear all

% test\_pll.m

%

% Script to test the PLL.

% Global parameters

Nb = 10; % Number of buffers

Ns = 100; % Samples in each buffer

%n = 0:1:Nb\*Ns-1;

%x = sin(2\*pi\*0.9\*n);

load('ref\_800hz.mat');

%load('ref\_stepf.mat');

x = ref\_in;

n = length(x);

%x = 10000\*sin(2\*pi\*0.11\*[1:1000]);

x1 = reshape(x, 100, 10);

state = pll\_init(0.1, 1.0, 1.0,2\*pi/100);

state.amp\_est = 1;

for i=1:Nb,

[state,y\_out] = pll(state,x1(:,i));

state.amp\_est

output(:,i) = y\_out';

end

y1 = output(:);

m=1:n;

%plot(m,y1,'r',m,x)

plot(m(400:500),y1(400:500),'r',m(400:500),ref\_in(400:500),'b')

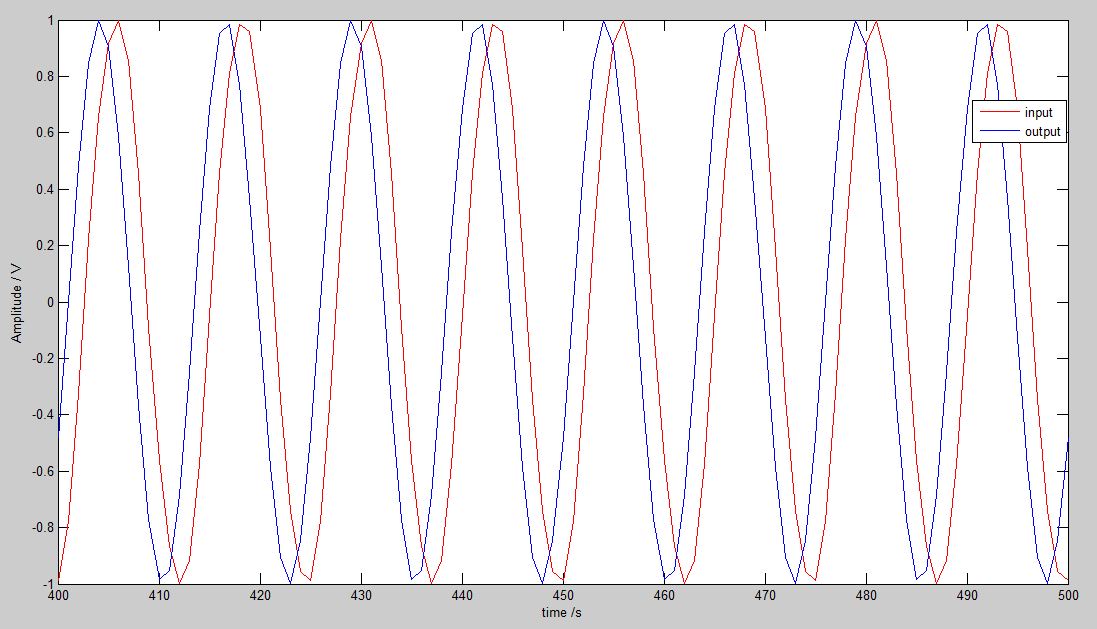


Fig: A plot showing and output of the PLL

1. **A plot showing the simulated performance of the final PLL, demonstrating that the PLL can adapt to abrupt changes in frequency/phase**

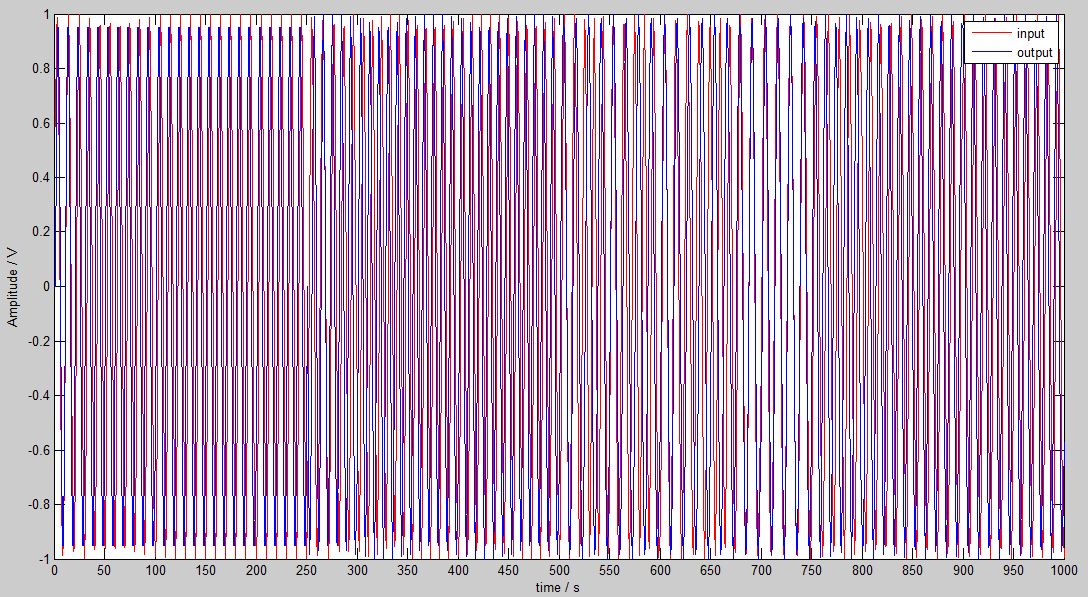
****

fig: a plot showing the simulated performance of the final PLL, demonstrating that the PLL can adapt to abrupt changes in frequency/phase.

The input frequency in the file ‘ref\_stepf.mat’ changes after 250 counts. So, let us take a look at how the PLL tracks at every interval of 250.

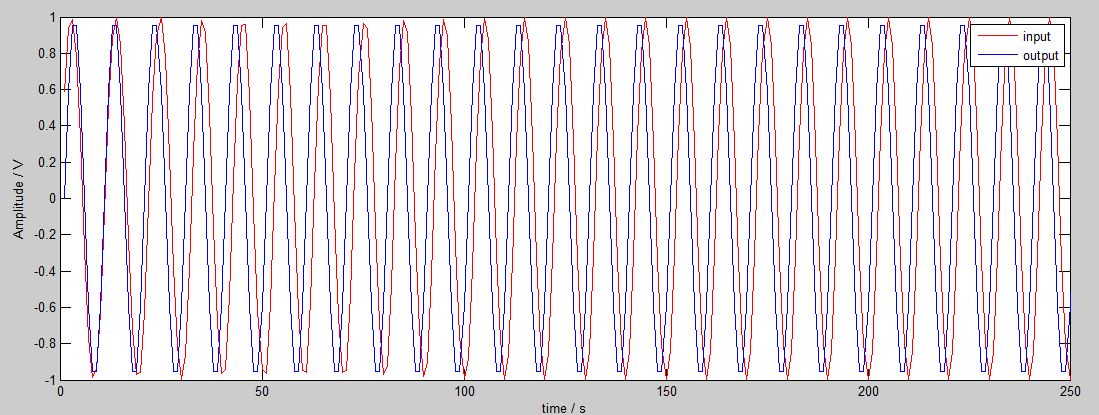


Fig: tracking 0-250

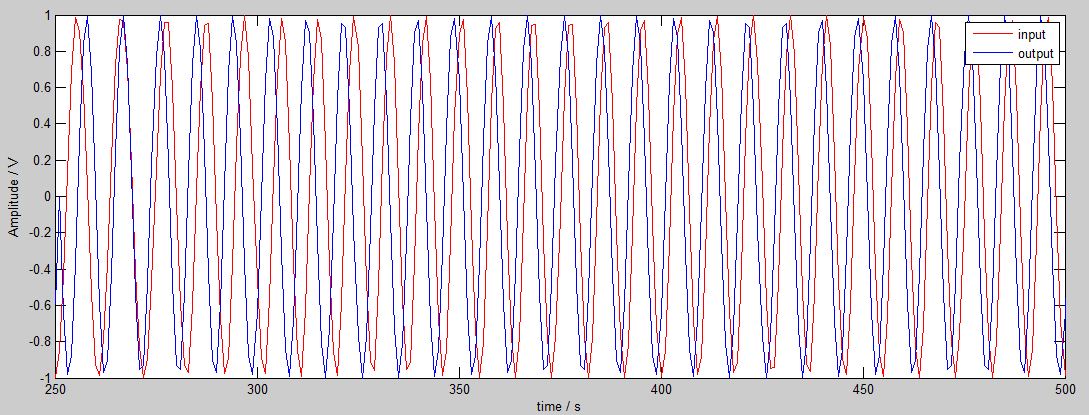


Fig: tracking 250-500

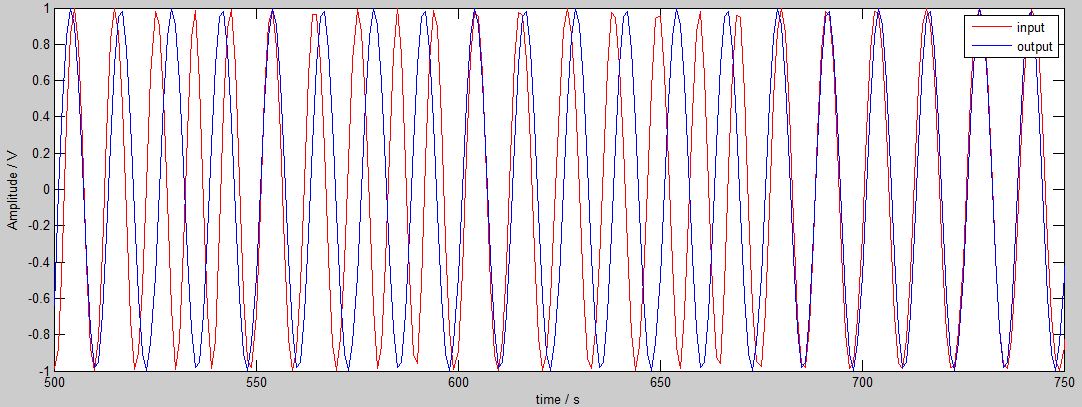


Fig: tracking 500-750

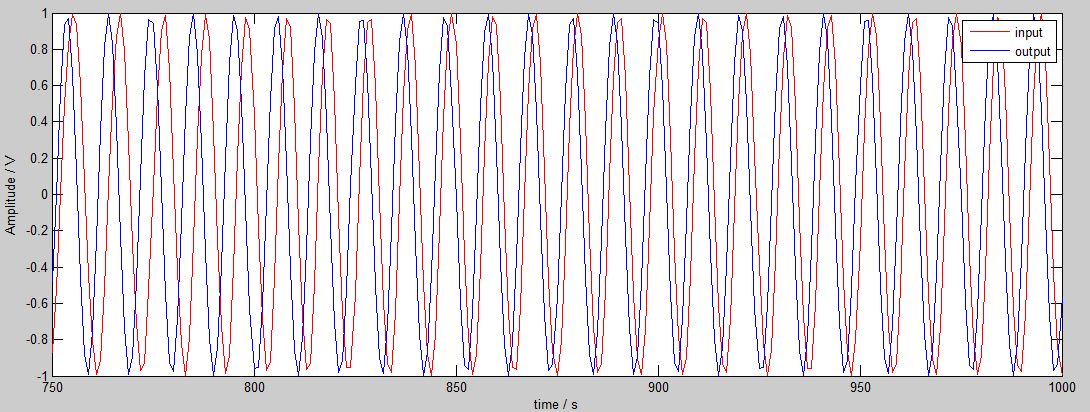


Fig: tracking 750-1000

We can observe that the PLL take some time to adjust and then after a while it finally tracks the signal. Also, we can see that on the lowest frequency interval 500-750 the tracking is somewhat off and taking much time to track.