Application Report
SPRABT4A–November 2013

Software Phase Locked Loop Design Using C2000™ Microcontrollers for Three Phase Grid Connected Applications

Manish Bhardwaj

ABSTRACT

Grid connected applications require an accurate estimate of the grid angle to feed power synchronous to the grid. This is achieved using a software phase locked loop (PLL). This application report discusses the different challenges in the design of software phase locked loops for three phase grid connected inverters and presents a methodology to design phase locked loops using C2000 controllers.

To illustrate, two well known PLL structures [1], [2] are discussed and designed using C2000 MUCs.

Contents

List of Figures

1 Introduction

The phase angle of the utility is a critical piece of information for the operation of power devices like PV Inverters that feed power into the grid. A PLL is a closed loop system in which an internal oscillator is controlled to keep the time and phase of an external periodical signal using a feedback loop. The quality of the lock directly affects the performance of the control loop in grid tied applications. As line notching, voltage unbalance, line dips, phase loss and frequency variations are common conditions faced by equipment interfacing with the electric grid. The PLL needs to be able to reject these sources of error and maintain a clean phase lock to the grid voltage.

In three phase systems, it is common to transform three phase time varying quantities to a dc system (in a rotating reference frame).

Equation 1 shows that the sequence of the voltages is $V_a \rightarrow V_c \rightarrow V_b$ and the frequency is ω.

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = V \begin{bmatrix} \cos(\omega t) \\ \cos(\omega t - 2\pi / 3) \\ \cos(wt - 4\pi / 3) \end{bmatrix} \quad (1)$$

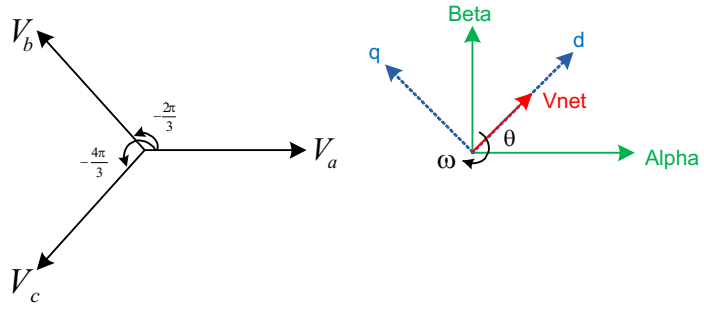


Figure 1. Transformation of Voltage From Three Phase to Stationary and Rotating Reference Frame

For transforming the three phase quantities to rotating reference frame, the first step is to transform the three phase quantities into an orthogonal component system (alpha, beta also called stationary reference frame) by taking the projections of the three phase quantities on an orthogonal axis. This is called the Clarke transform (see Equation 2).

$$V_{\alpha\beta 0} = T_{abc \rightarrow \alpha\beta 0} V_{abc}$$

$$\begin{bmatrix} V_\alpha \\ V_\beta \\ V_o \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & \cos(2\pi/3) & \cos(4\pi/3) \\ 0 & \sin(2\pi/3) & \sin(4\pi/3) \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} X \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(\omega t) \\ \sin(\omega t) \\ 0 \end{bmatrix} V \quad (2)$$

In the stationary reference frame, the net voltage vector makes an angle θ with the orthogonal reference frame and rotates at a frequency of ω. The system can then be reduced to DC by taking the projection of the stationary reference frame components on the rotating reference frame. This is called the Park transform (see Equation 3).

$$V_{dq0} = T_{\alpha\beta 0 \rightarrow dq0} V_{\alpha\beta 0}$$

$$\begin{bmatrix} v_d \\ v_q \\ v_o \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} X \begin{bmatrix} v_\alpha \\ v_\beta \\ v_o \end{bmatrix} \quad (3)$$

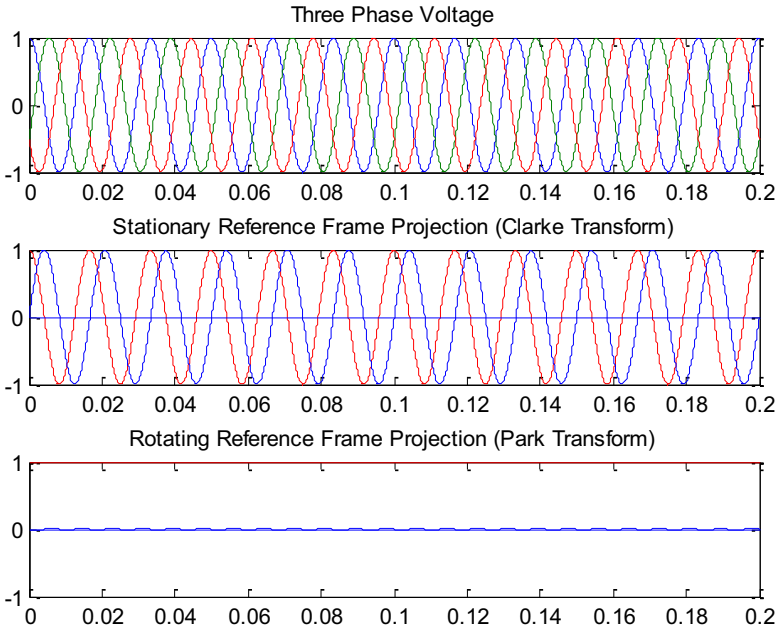


Figure 2. Simulation of Transforms From Three Phase to Rotating Reference Frame

2 Synchronous Reference Frame PLL

The role of the PLL in three phase context is to accurately estimate the angle the net voltage vector is making by measuring the instantaneous voltage waveforms. Assuming the angle the PLL estimates is θ and the actual angle is ω*t, ABC->DQ0 transform can be written using Equation 2 and Equation 3:

$$\begin{bmatrix} v_d \\ v_q \\ v_o \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} X \begin{bmatrix} \cos(\omega t) \\ \sin(\omega t) \\ 0 \end{bmatrix} V = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(\theta) * \cos(\omega t) + \sin(\theta)\sin(\omega t) \\ -\sin(\theta) * \cos(\omega t) + \cos(\theta)\sin(\omega t) \\ 0 \end{bmatrix} V \tag{4}$$

Using trigonometric identities, Equation 5 can be reduced to:

$$\begin{bmatrix} v_d \\ v_q \\ v_o \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(\omega t - \theta) \\ \sin(\omega t - \theta) \\ 0 \end{bmatrix} \tag{5}$$

When PLL angle is close to the actual voltage vector angle, (ω*t-θ) is small or close to zero then sin(ω*t-θ) ~ = (ω*t-θ). Therefore, it can be said for a balanced three phase system when PLL is locked, the q-axis component in the rotating reference frame reduces to zero and when it is not locked or has small error the q-axis component is linearly proportional to the error:

$$V_q \approx (wt - \theta) \tag{6}$$

This property is used in the Synchronous Reference Frame PLL for three phase grid connected application. The three phase quantities are transformed into the rotating reference frame and the q component is used as the phase detect value. A low pass filter/PI is then used to eliminate steady state error and the output fed to a VCO, which generates the angle and sine values.

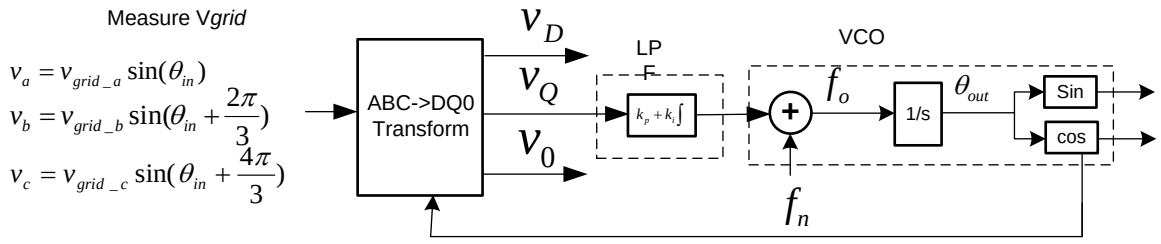


Figure 3. SPLL for 3ph-Based on Stationary Reference Frame

It is known from Equation 6 that any error in the angle lock will show up on the q term and that the relation between the error for small values is linear, see Equation 7:

$$err \propto V_{grid}\,(\theta_{grid} - \theta_{PLL}) \tag{7}$$

Small signal analysis is done using the network theory, where the feedback loop is broken to get the open loop transfer equation and then the closed loop transfer function is given by:

$$\text{Closed Loop TF} = \text{Open Loop TF} / (1 + \text{OpenLoopTF}) \tag{8}$$

The PLL transfer function can be written as follows:

$$\textit{Closed loop Phase TF}: \quad H_O(s) = \frac{\theta_{out}(s)}{\theta_{in}(s)} = \frac{LF(s)}{s + LF(s)} = \frac{v_{grid}(k_p s + \frac{k_p}{T_i})}{s^2 + v_{grid}k_p s + v_{grid}\frac{k_p}{T_i}}$$

$$\textit{Closed loop error transfer function}: \quad E_O(s)\frac{V_d(s)}{\theta_{in}(s)} = 1 - H_O(s) = \frac{s}{s + LF(s)} = \frac{s^2}{s^2 + k_p s + \frac{k_p}{T_i}} \tag{9}$$

Comparing the closed loop phase transfer function to the generic second order system transfer function:

$$H(s) = \frac{2\zeta\omega_n s + \omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{10}$$

Now, comparing this with the closed loop phase transfer function, the natural frequency and the damping ration of the lineralized PLL is given as:

$$\omega_n = \sqrt{\frac{v_{grid}K_p}{T_i}}$$

$$\zeta = \sqrt{\frac{v_{grid}T_i K_p}{4}} \tag{11}$$

2.1 Discrete Implementation of PI Controller

The loop filter or PI is implemented as a digital controller as shown in Equation 12:

$$ylf[n] = ylf[n-1] * A1 + ynotch[n] * B0 + ynotch[n-1] * B1 \tag{12}$$

Using z transform, Equation 13 can be re-written as:

$$\frac{ylf(z)}{ynotch(z)} = \frac{B0 + B1 * z^{-1}}{1 - z^{-1}} \tag{13}$$

It is known that the Laplace Transform of the loop filter (PI controller) is given by:

$$\frac{ylf(s)}{ynotch(s)} = K_p + \frac{K_i}{s} \tag{14}$$

Now, using Bi-linear transformation, replace $s = \frac{2}{T}\left(\frac{z-1}{z+1}\right)$, where T = Sampling Time.

$$\frac{ylf(z)}{ynotch(z)} = \frac{\left(\frac{2*K_p + K_i*T}{2}\right) - \left(\frac{2*K_p - K_i*T}{2}\right)z^{-1}}{1 - z^{-1}} \tag{15}$$

Equation 2 and Equation 3 can be compared to map the proportional and integral gain of the PI controller into the digital domain. The next challenge is selecting an appropriate value for the proportional and integral gain.

It is known that the step response to a general second order equation:

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{16}$$

is given as:

$$y(t) = 1 - ce^{-\sigma t}\sin(\omega_d t + \varphi) \tag{17}$$

Ignoring the LHP zero from Equation 17, the settling time is given as the time it takes for the response to settle between an error band, say this error is ∂, then:

$$1 - \partial = 1 - ce^{-\sigma t} \Rightarrow \partial = ce^{-\sigma t} \Rightarrow t_s = \frac{1}{\sigma} * \ln\left(\frac{c}{\sigma}\right)$$

$$\text{Where } \sigma = \varsigma\omega_n \text{ and } c = \frac{\omega_n}{\omega_d} \text{ and } \omega_d = \sqrt{1-\varsigma^2}\,\omega_n \tag{18}$$

Using settling time of 30 ms, error band of 5% and damping ratio of 0.7, the natural frequency of 158.6859 can be obtained; and then back substituting you get K_p = 222.1603 and K_1 = 25181.22.

Back substituting these values into the digital loop filter coefficients, you get:

$$B0 = \left(\frac{2*K_p + K_i*T}{2}\right) \text{ and } B1 = -\left(\frac{2*K_p - K_i*T}{2}\right) \tag{19}$$

For 10 Khz, run rate for the PLL, B0 = 223.4194 and B1 = - 220.901.

2.2 Simulating the Phase Locked Loop for Varying Conditions

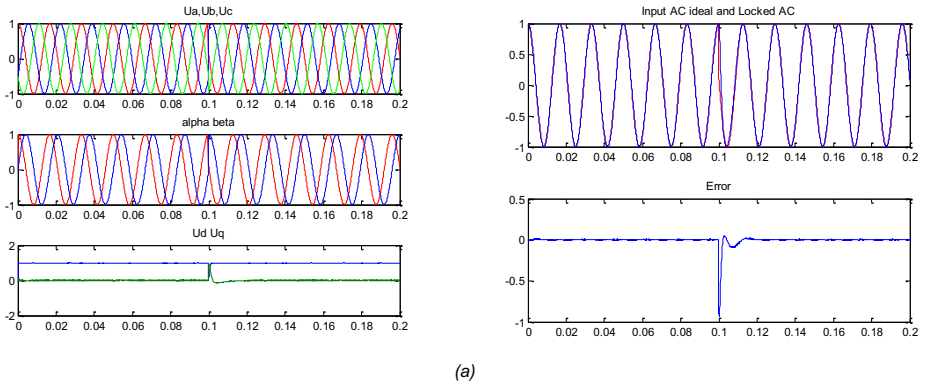
Before coding the SPLL structure, it is essential to simulate the behavior of the PLL for different conditions on the grid. Fixed point processors are used for lower cost in many grid tied converters. IQ Math is a convenient way to look at fixed point numbers with a decimal point. C2000 IQ math library provides built in functions that can simplify handling of the decimal point by the programmer. First MATLAB is used to simulate and identify the Q point the algorithm needs to be run at. Below is the MATLAB script using the fixed point tool box that tests the PLL algorithm with varying grid conditions.

```
%%%%%%%%%%%%%%%%%%% PLL 3ph Modeling %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Texas Instruments
% Digital Control Systems Group, Houston, TX
% Manish Bhardwaj
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  This software is licensed for use with Texas Instruments C28x
%  family DSCs.  This license was provided to you prior to installing
%  the software.
% ------------------------------------------------------------------------
%          Copyright (C) 2010-2012 Texas Instruments, Incorporated.
%                          All Rights Reserved.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
close all;
clc;

%Select numeric type,
T=numerictype('WordLength',32,'FractionLength',22);

%Specify math attributes to the fimath object
F=fimath('RoundMode','floor','OverflowMode','wrap');
F.ProductMode='SpecifyPrecision';
F.ProductWordLength=32;
F.ProductFractionLength=22;
F.SumMode='SpecifyPrecision';
F.SumWordLength=32;
F.SumFractionLength=22;

%specify fipref object, to display warning in cases of overflow and
%underflow
P=fipref;
P.LoggingMode='on';
P.NumericTypeDisplay='none';
P.FimathDisplay='none';

%PLL Modelling starts from here
Fs=10000;
Ts=1/Fs;            %Sampling Time = (1/fs) , Note Ts is related to the frequency
                    % the ISR would run in the solar application as well, assuming
                    % 10Khz control loop for the inverter

Tfinal=0.2;         % Total time the simulation is run for
t=0:Ts:Tfinal;      % time vector
GridFreq=60;        % nominal grid frequency
wn=2*pi*GridFreq;   % nominal frequency in radians
fn=GridFreq;
wu=2*pi*GridFreq;
theta=rem((t*2*pi*GridFreq),2*pi);
L=length(t);

%generate input signal and create a fi object of it

% CASE 1 : Phase Jump at the Mid Point
%
```

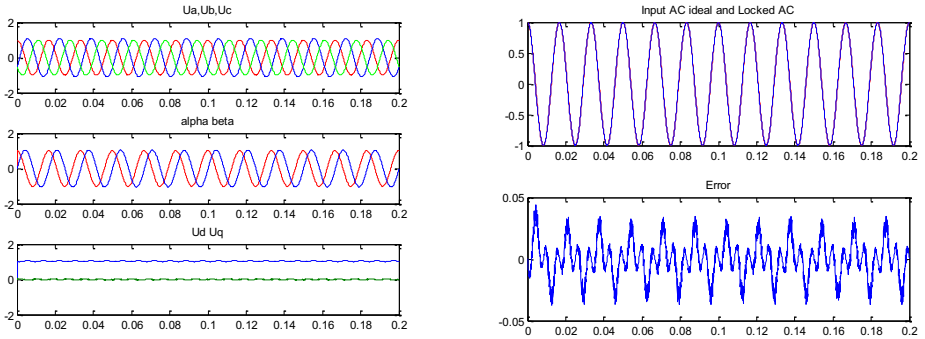
```
% for n=1:floor(L/2)
%     Ua(n)=cos(theta(n));
%     Ub(n)=cos(theta(n)-2*pi/3);
%     Uc(n)=cos(theta(n)-4*pi/3);
% end
% for n=floor(L/2):L
%     Ua(n)=cos(theta(n)+1.5);
%     Ub(n)=cos(theta(n)-2*pi/3+1.5);
%     Uc(n)=cos(theta(n)-4*pi/3+1.5);
% end
% Ua_ideal=Ua;

%CASE 2: Unbalanced Grid
Ua_ideal=cos(theta);


Ua=fi(Ua,T,F);
Ub=fi(Ub,T,F);
Uc=fi(Uc,T,F);
Ua_ideal=fi(Ua_ideal,T,F);

%declare arrays used by the PLL process
ylf =fi([0,0],T,F);
u_q =fi([0,0],T,F);
Theta=fi([0,0,0],T,F);
fo=fi(0,T,F);
Ualpha=fi([0,0],T,F);
Ubeta=fi([0,0],T,F);
Ud_plus=fi([0,0],T,F);
Uq_plus=fi([0,0],T,F);
fn=fi(fn,T,F);
wo=fi(0,T,F);

% simulate the PLL process
for n=3:L    % No of iteration of the PLL process in the simulation time

    Ualpha(n) =fi((2.0/3.0),T,F)*(Ua(n)-fi(0.5,T,F)*(Ub(n)+Uc(n)));
    Ubeta(n)  =fi((2.0/3.0)*0.866,T,F)*(Ub(n)-Uc(n));

    Ud_plus(n)=Ualpha(n)*cos(Theta(n))+Ubeta(n)*sin(Theta(n));
    Uq_plus(n)=-Ualpha(n)*sin(Theta(n))+Ubeta(n)*cos(Theta(n));

    u_q(1)=Uq_plus(n);

    %Loop Filter
    ylf(1)=ylf(2)+fi(167.9877775,T,F)*u_q(1)-fi(165.2122225,T,F)*u_q(2);
    %ylf(1)=ylf(2)+fi(1000.00005,T,F)*u_q(1)-fi(999.99995,T,F)*u_q(2);

    ylf(1)=min([ylf(1) fi(200.0,T,F)]);

    %update u_q for future use
    u_q(2)=u_q(1);

    ylf(2)=ylf(1);

    %update output frequency
    fo=fn+ylf(1);

    Theta(n+1)=Theta(n)+fi(2*pi,T,F)*(fo*Ts);

    if(Theta(n+1)>=(fi(2*pi,T,F)-fi(2*pi,T,F)*(fo*Ts)))
        Theta(n+1)=0;
    end

end
```

```
figure,subplot(3,1,1),plot(t,Ua,'r',t,Ub,'b',t,Uc,'g'),title('Ua,Ub,Uc');
subplot(3,1,2),plot(t,Ualpha,'r',t,Ubeta),title('alpha beta');
subplot(3,1,3),plot(t,Ud_plus(1:L),t,Uq_plus(1:L)), title('Ud Uq') ;

figure,subplot(2,1,1),plot(t,Ua_ideal,'r',t,cos(Theta(1:L)),'b'),title('Input AC ideal and Locked
AC');
subplot(2,1,2),plot(t,Ua_ideal-cos(Theta(1:L))),title('Error');
% for n=1:floor(L)
% Ua(n)=cos(theta(n));
% Ub(n)=1.1*cos(theta(n)-2*pi/3);
% Uc(n)=cos(theta(n)-4*pi/3);
% end
% Ua_ideal=cos(theta);

%CASE 3: voltage harmonics
% for n=1:floor(L)
% Ua(n)=cos(theta(n))+0.05*cos(5*theta(n));
% Ub(n)=cos(theta(n)-2*pi/3)+0.05*(cos(5*(theta(n)-2*pi/3)));
% Uc(n)=cos(theta(n)-4*pi/3)+0.05*(cos(5*(theta(n)-4*pi/3)));
% end
% Ua_ideal=cos(theta);

% CASE 4: voltage dips and swells

for n=1:floor(L/2)
    Ua(n)=cos(theta(n));
    Ub(n)=cos(theta(n)-2*pi/3);
    Uc(n)=cos(theta(n)-4*pi/3);
end
for n=floor(L/2):L
    Ua(n)=0.7*cos(theta(n));
    Ub(n)=0.7*cos(theta(n)-2*pi/3);
    Uc(n)=0.7*cos(theta(n)-4*pi/3);
end
```

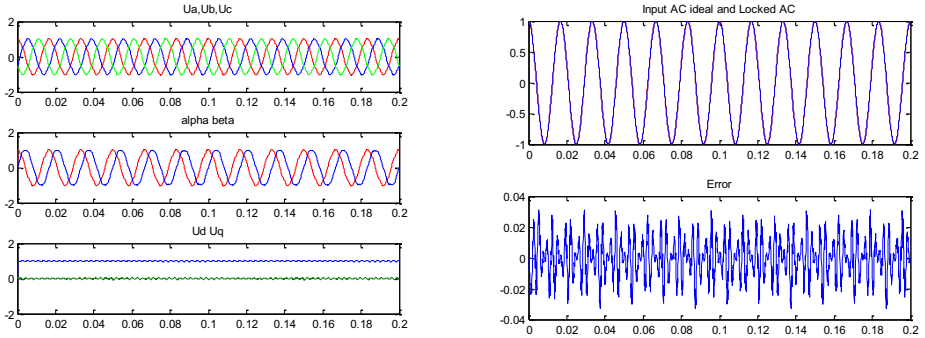
Below are the results of the varying grid conditions on the PLL:



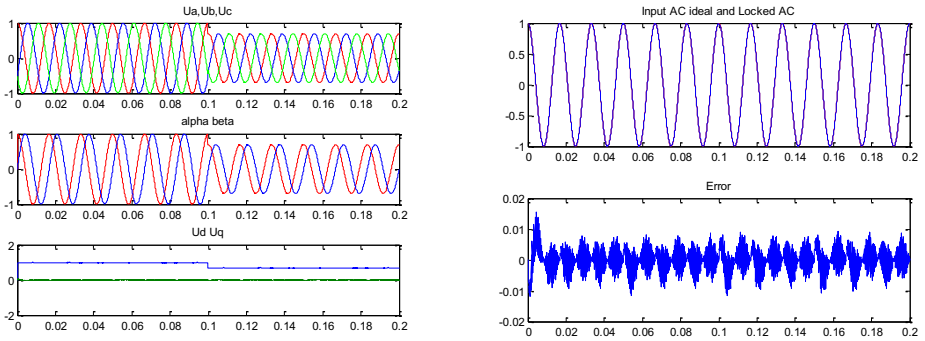
(a)



(b)



(c)



(d)

A Phase Jump of 90°

B Voltage Imbalance of 10% on one phase

C 5% 5th Harmonic content

D Amplitude change (Voltage Sags and Dips)

Figure 4. PLL Response to Varying Grid Conditions

2.3 Implementing PLL on C2000 Controller Using IQ Math

As the regular inverter typically uses IQ24, and the Q point chosen for the PLL is IQ21 as shown in the above section , the code for the PLL can be written as follows.

SPLL_3ph_SRF_IQ.h header file

:

```
#ifndef SPLL_3ph_SRF_IQ_H_
#define SPLL_3ph_SRF_IQ_H_

#define SPLL_SRF_Q _IQ21
#define SPLL_SRF_Qmpy _IQ21mpy

//*********** Structure Definition ********//
typedef struct{
           int32      B1_lf;
           int32      B0_lf;
           int32      A1_lf;
}SPLL_3ph_SRF_IQ_LPF_COEFF;

typedef struct{
           int32 v_q[2];
           int32 ylf[2];
           int32 fo; // output frequency of PLL
           int32 fn; //nominal frequency
           int32 theta[2];
           int32 delta_T;
           SPLL_3ph_SRF_IQ_LPF_COEFF lpf_coeff;
}SPLL_3ph_SRF_IQ;

//*********** Function Declarations *******//
void SPLL_3ph_SRF_IQ_init(int Grid_freq, long DELTA_T, SPLL_3ph_SRF_IQ *spll);
void SPLL_3ph_SRF_IQ_FUNC(SPLL_3ph_SRF_IQ *spll_obj);

//*********** Macro Definition ***********//
#define SPLL_3ph_SRF_IQ_MACRO(spll_obj)\
           /*update v_q[0] before calling the routine*/ \
           /* Loop Filter                    */ \

spll_obj.ylf[0]=spll_obj.ylf[1]+SPLL_SRF_Qmpy(spll_obj.lpf_coeff.B0_lf,spll_obj.v_q[0])+SPLL_SRF_Q
mpy(spll_obj.lpf_coeff.B1_lf,spll_obj.v_q[1]);             \
           spll_obj.ylf[1]=spll_obj.ylf[0];  \
           spll_obj.v_q[1]=spll_obj.v_q[0];  \
           spll_obj.ylf[0]=(spll_obj.ylf[0]>SPLL_SRF_Q(200.0))?SPLL_SRF_Q(200.0):spll_obj.ylf[0];\
           /* VCO                            */ \
           spll_obj.fo=spll_obj.fn+spll_obj.ylf[0]; \

spll_obj.theta[0]=spll_obj.theta[1]+SPLL_SRF_Qmpy(SPLL_SRF_Qmpy(spll_obj.fo,spll_obj.delta_T),SPLL
_SRF_Q(2*3.1415926));                                          \
           if(spll_obj.theta[0]>SPLL_SRF_Q(2*3.1415926)) \
                     spll_obj.theta[0]=spll_obj.theta[0]-SPLL_SRF_Q(2*3.1415926); \
           spll_obj.theta[1]=spll_obj.theta[0];\
#endif /* SPLL_3ph_SRF_IQ_H_ */
```

SPLL_3ph_SRF_IQ.c source file:

```
#include "Solar_IQ.h"

//*********** Structure Init Function ****//
void SPLL_3ph_SRF_IQ_init(int Grid_freq, long DELTA_T, SPLL_3ph_SRF_IQ *spll_obj)
{
           spll_obj->v_q[0]=SPLL_SRF_Q(0.0);
           spll_obj->v_q[1]=SPLL_SRF_Q(0.0);

           spll_obj->ylf[0]=SPLL_SRF_Q(0.0);
           spll_obj->ylf[1]=SPLL_SRF_Q(0.0);
```

```
            spll_obj->fo=SPLL_SRF_Q(0.0);
            spll_obj->fn=SPLL_SRF_Q(Grid_freq);

            spll_obj->theta[0]=SPLL_SRF_Q(0.0);
            spll_obj->theta[1]=SPLL_SRF_Q(0.0);

            // loop filter coefficients for 20kHz
            spll_obj->lpf_coeff.B0_lf=_IQ21(166.9743);
            spll_obj->lpf_coeff.B1_lf=_IQ21(-166.266);
            spll_obj->lpf_coeff.A1_lf=_IQ21(-1.0);

            spll_obj->delta_T=DELTA_T;
}

//*********** Function Definition ********//
void SPLL_3ph_SRF_IQ_FUNC(SPLL_3ph_SRF_IQ *spll_obj)
{
            //update v_q[0] before calling the routine
            //---------------------------------//
            // Loop Filter                     //
            //---------------------------------//
            spll_obj->ylf[0]=spll_obj->ylf[1]+SPLL_SRF_Qmpy(spll_obj->lpf_coeff.B0_lf,spll_obj-
>v_q[0])+SPLL_SRF_Qmpy(spll_obj->lpf_coeff.B1_lf,spll_obj->v_q[1]);
            spll_obj->ylf[1]=spll_obj->ylf[0];
            spll_obj->v_q[1]=spll_obj->v_q[0];

            spll_obj->ylf[0]=(spll_obj->ylf[0]>SPLL_SRF_Q(200.0))?SPLL_SRF_Q(200.0):spll_obj-
>ylf[0];
            //---------------------------------//
            // VCO                             //
            //---------------------------------//
            spll_obj->fo=spll_obj->fn+spll_obj->ylf[0];

            spll_obj->theta[0]=spll_obj->theta[1]+SPLL_SRF_Qmpy(SPLL_SRF_Qmpy(spll_obj-
>fo,spll_obj->delta_T),SPLL_SRF_Q(2*3.1415926));
            if(spll_obj->theta[0]>SPLL_SRF_Q(2*3.1415926))
                        spll_obj->theta[0]=spll_obj->theta[0]-SPLL_SRF_Q(2*3.1415926);

            spll_obj->theta[1]=spll_obj->theta[0];
}
```

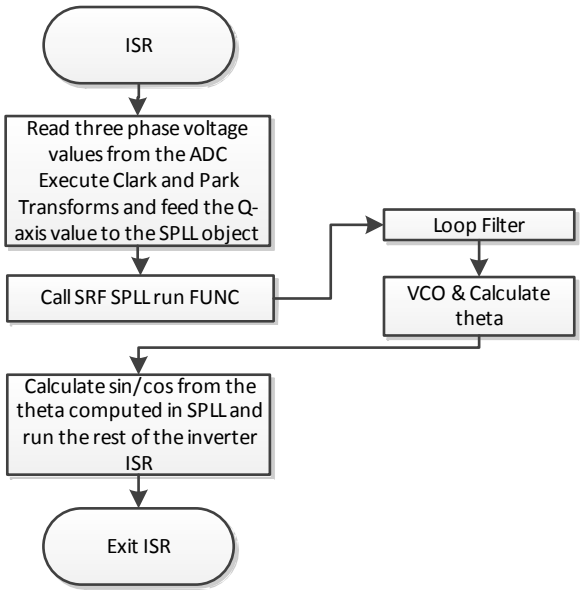


Figure 5. SRF SPLL Usage Flowchart

To use this block in an end application, declare objects for the SPLL structure, loop filter coefficients and notch filter coefficients.

```
// ------------- Software PLL for Grid Tie Applications ----------
SPLL_3ph_SRF_IQ spll1;
```

Call the SPLL_3ph_SRF_init routine with the frequency of the ISR, the SPLL will be executed in as parameter and the grid frequency and then call the notch filter update coefficient update routine.

```
SPLL_3ph_SRF_init(GRID_FREQ,_IQ21((float)(1.0/ISR_FREQUENCY)),&spll1;
```

In the ISR, read the sinusoidal input voltage from the ADC and feed it into the SPLL block; write to invsine value with the sinus of the current grid angle. This can then be used in control operations.

```
abc_dq0_pos1.a = _IQmpy(GridMeas1,_IQ(0.5));
abc_dq0_pos1.b = _IQmpy(GridMeas2,_IQ(0.5));
abc_dq0_pos1.c = _IQmpy(GridMeas3,_IQ(0.5));
abc_dq0_pos1.sin=_IQsin((spll1.theta[1])<<3);// Q24 to Q21
abc_dq0_pos1.cos=_IQcos((spll1.theta[1])<<3);// Q24 to Q21
ABC_DQ0_POS_IQ_MACRO(abc_dq0_pos1);

// Q24 to Q21
spll1.v_q[0] = (int32)(_IQtoIQ21(abc_dq0_pos1.q));

// SPLL call
SPLL_3ph_SRF_IQ_FUNC(&spll1;
```

3 Imbalances in Three Phase Grid

The grid is subject to varying conditions, which results in imbalances on the phase voltages. From the theory of symmetrical components, it is known that any unbalanced three phase system can be reduced to two symmetrical systems and zero component: one rotating in the positive direction called the positive sequence and the other rotating in the negative sequence called the negative sequence (see Figure 6). The behavior of unbalanced voltages on Park and Clarke transform is analyzed in the section below.

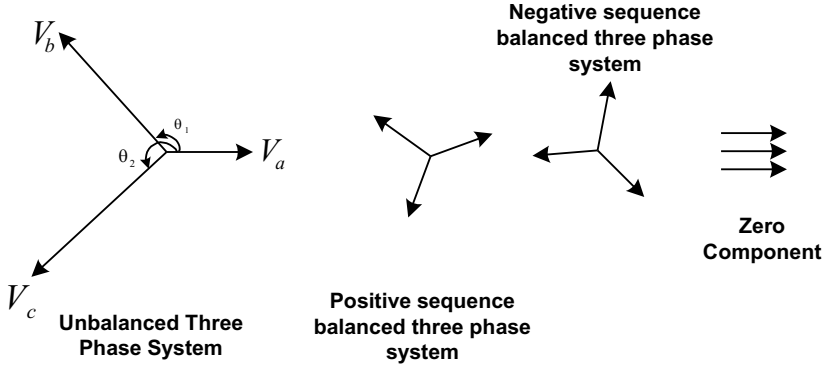


Figure 6. Imbalances in Three Phase Grid

.

$$v = V^{+1}\begin{bmatrix}\cos(wt)\\ \cos(wt-2\pi/3)\\ \cos(wt-4\pi/3)\end{bmatrix} + V^{-1}\begin{bmatrix}\cos(wt)\\ \cos(wt-4\pi/3)\\ \cos(wt-2\pi/3)\end{bmatrix} + V^{0}\begin{bmatrix}1\\1\\1\end{bmatrix} \quad (20)$$

Using Clarke transform, ω.v.t the positive sequence.

$$v_{\alpha\beta} = T_{abc->\alpha\beta} * v = \frac{2}{3}\begin{bmatrix}1 & -\frac{1}{2} & -\frac{1}{2}\\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2}\\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2}\end{bmatrix} * v = V^{+1}\begin{bmatrix}\cos(wt)\\ \sin(wt)\end{bmatrix} + V^{-1}\begin{bmatrix}\cos(-wt)\\ \sin(-wt)\end{bmatrix} \quad (21)$$

Furthermore, taking Park's transform and projections on the rotating reference frame, it is observed that any negative sequence component appears with twice the frequency on the positive sequence rotating frame axis and vice versa.

$$v_{dq+} = T_{abc->dq0+} * v_{\alpha\beta} = \begin{bmatrix} \cos(\omega t) & \sin(\omega t) \\ -\sin(\omega t) & \cos(\omega t) \end{bmatrix} * \left(V^{+1}\begin{bmatrix} \cos(wt) \\ \sin(wt) \end{bmatrix} + V^{-1}\begin{bmatrix} \cos(-wt) \\ \sin(-wt) \end{bmatrix} \right) = V^{+1}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + V^{-1}\begin{bmatrix} \cos(-2wt) \\ \sin(-2wt) \end{bmatrix}$$

$$v_{dq-} = T_{abc->dq0-} * v_{\alpha\beta} = \begin{bmatrix} \cos(\omega t) & -\sin(\omega t) \\ \sin(\omega t) & \cos(\omega t) \end{bmatrix} * \left(V^{+1}\begin{bmatrix} \cos(wt) \\ \sin(wt) \end{bmatrix} + V^{-1}\begin{bmatrix} \cos(-wt) \\ \sin(-wt) \end{bmatrix} \right) = V^{+1}\begin{bmatrix} \cos(-2wt) \\ \sin(-2wt) \end{bmatrix} + V^{-1}\begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad (22)$$

This causes errors in the estimation of the grid angle, and needs to be taken into account while designing a phase locked loop for three phase grid connected application.

4 Decoupled Double Synchronous Reference Frame PLL

As seen in Section 3, the unbalanced three phase system can be resolved into two balanced three phase systems: one rotating in the positive sequence and the other in the negative sequence [1]. Before the PLL has been locked, the positive and negative vector can be written as Equation 23:

$$v = V^{+1}\begin{bmatrix} \cos(wt+\varphi_1) \\ \cos(wt-2\pi/3+\varphi_1) \\ \cos(wt-4\pi/3+\varphi_1) \end{bmatrix} + V^{-1}\begin{bmatrix} \cos(wt+\varphi_{-1}) \\ \cos(wt-4\pi/3+\varphi_{-1}) \\ \cos(wt-2\pi/3+\varphi_{-1}) \end{bmatrix} + V^{0}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad (23)$$

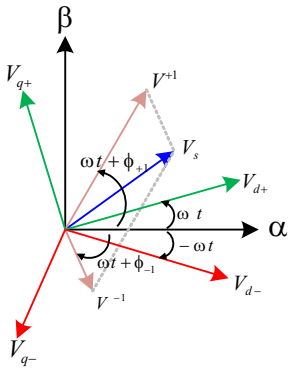


Figure 7. Positive and Negative Sequence of Unbalance Three Phase System

Use the Park transform:

$$v_{\alpha\beta} = V^{+1}\begin{bmatrix} \cos(\omega t+\varphi_{+1}) \\ \sin(\omega t+\varphi_{+1}) \end{bmatrix} + V^{-1}\begin{bmatrix} \cos(-\omega t+\varphi_{-1}) \\ \sin(-\omega t+\varphi_{-1}) \end{bmatrix} \quad (24)$$

Projections of the orthogonal system on the rotating reference frame of the positive recurrence can be written as shown in Equation 25:

$$v_{dq+} = \left(V^{+1}\begin{bmatrix} \cos(\omega t+\varphi_{+1}) \\ \sin(\omega t+\varphi_{+1}) \end{bmatrix} + V^{-1}\begin{bmatrix} \cos(-\omega t+\varphi_{-1}) \\ \sin(-\omega t+\varphi_{-1}) \end{bmatrix} \right) * \begin{bmatrix} \cos(\omega t) & \sin(\omega t) \\ -\sin(\omega t) & \cos(\omega t) \end{bmatrix} \quad (25)$$

Simplifying,

$$\Rightarrow v_{dq+} = \left(V^{+1}\begin{bmatrix}\cos(\varphi_{+1})\\ \sin(\varphi_{+1})\end{bmatrix} + V^{-1}\begin{bmatrix}\cos(-\omega t+\varphi_{-1})\\ \sin(-\omega t+\varphi_{-1})\end{bmatrix} * \begin{bmatrix}\cos(\omega t) & \sin(\omega t)\\ -\sin(\omega t) & \cos(\omega t)\end{bmatrix}\right)$$

$$\Rightarrow v_{dq+} = \left(V^{+1}\begin{bmatrix}\cos(\varphi_{+1})\\ \sin(\varphi_{+1})\end{bmatrix} + V^{-1}\begin{bmatrix}\cos(-\omega t+\varphi_{-1})\\ \sin(-\omega t+\varphi_{-1})\end{bmatrix} * \begin{bmatrix}\cos(\omega t) & \sin(\omega t)\\ -\sin(\omega t) & \cos(\omega t)\end{bmatrix}\right)$$

$$\Rightarrow v_{dq+} = \left(V^{+1}\begin{bmatrix}\cos(\varphi_{+1})\\ \sin(\varphi_{+1})\end{bmatrix} + V^{-1}\begin{bmatrix}\cos(-\omega t+\varphi_{-1})\cos(\omega t)+\sin(-\omega t+\varphi_{-1})\sin(\omega t)\\ -\cos(-\omega t+\varphi_{-1})\sin(\omega t)+\sin(-\omega t+\varphi_{-1})\cos(\omega t)\end{bmatrix}\right)$$

$$\Rightarrow v_{dq+} = \left(V^{+1}\begin{bmatrix}\cos(\varphi_{+1})\\ \sin(\varphi_{+1})\end{bmatrix} + V^{-1}\begin{bmatrix}\cos(-2\omega t+\varphi_{-1})\\ \sin(-2\omega t+\varphi_{-1})\end{bmatrix}\right)$$

$$\Rightarrow v_{dq+} = \left(V^{+1}\begin{bmatrix}\cos(\varphi_{+1})\\ \sin(\varphi_{+1})\end{bmatrix} + V^{-1}\begin{bmatrix}\cos(\varphi_{-1})\cos(2\omega t)+\sin(\varphi_{-1})\sin(2\omega t)\\ \sin(\varphi_{-1})\cos(2\omega t)-\cos(\varphi_{-1})\sin(2\omega t)\end{bmatrix}\right)$$

$$\Rightarrow v_{dq+} = \left(V^{+1}\begin{bmatrix}\cos(\varphi_{+1})\\ \sin(\varphi_{+1})\end{bmatrix} + V^{-1}\cos(\varphi_{-1})\begin{bmatrix}\cos(2\omega t)\\ -\sin(2\omega t)\end{bmatrix} + V^{-1}\sin(\varphi_{-1})\begin{bmatrix}\sin(2\omega t)\\ \cos(2\omega t)\end{bmatrix}\right)$$

$$\Rightarrow v_{dq+} = \left(V^{+1}\begin{bmatrix}\cos(\varphi_{+1})\\ \sin(\varphi_{+1})\end{bmatrix} + \bar{v}_{d-}\begin{bmatrix}\cos(2\omega t)\\ -\sin(2\omega t)\end{bmatrix} + \bar{v}_{q-}\begin{bmatrix}\sin(2\omega t)\\ \cos(2\omega t)\end{bmatrix}\right) \quad (26)$$

The d and q axis components are written as:

$$v_{d+\_decoupled} = V^{+1}\cos(\varphi_{+1}) = v_{d+} - \bar{v}_{d-}\cos(2\omega t) - \bar{v}_{q-}\sin(2\omega t)$$

$$v_{q+\_decoupled} = V^{+1}\sin(\varphi_{+1}) = v_{q+} + \bar{v}_{d-}\sin(2\omega t) - \bar{v}_{q-}\cos(2\omega t) \quad (27)$$

A decoupling network is used to eliminate these values [1].

Once decoupled, the same property of the q component that was used to design the SRF PLL can be used (see Section 2.1), where the positive sequence q axis component, when the PLL is locked, is zero or the value is linear to the angle error. The PLL using the decoupling network is shown in Figure 8.

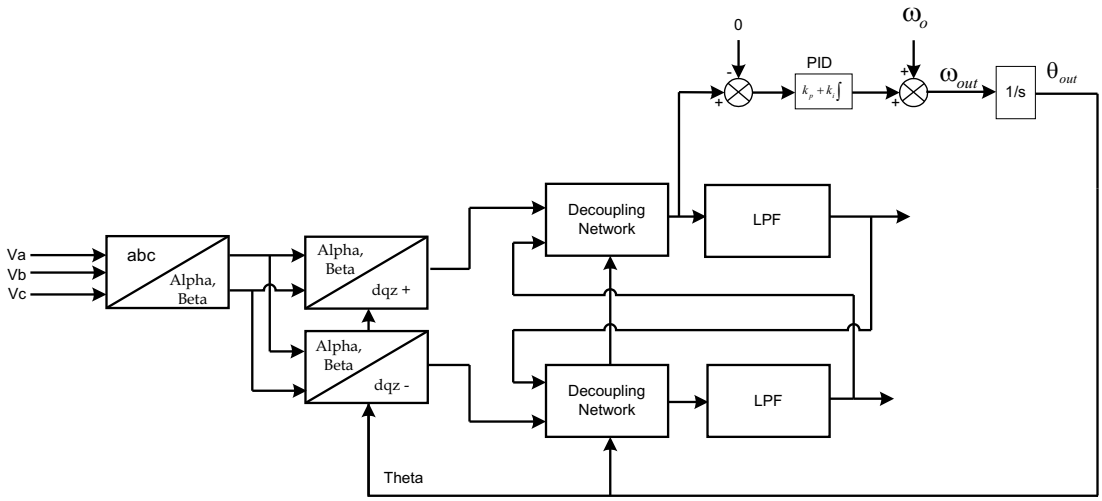


Figure 8. DDSRF PLL Structure

4.1 Discrete Implementation of Low Pass Filter

Typical low pass filter transfer function is given by:

$$LPF(s) = \frac{\omega_f}{(s + \omega_f)} \tag{28}$$

In the analog domain, now using bilinear transformation:

$$LPF(z) = \frac{\omega_f}{(\frac{2}{T}\frac{(z-1)}{(z+1)} + \omega_f)} = \frac{\frac{\omega_f T}{(2 + T\omega_f)}(z+1)}{(z + \frac{(\omega_f T - 2)}{(\omega_f T + 2)})} = \frac{k_1(z+1)}{(z + k_2)} \tag{29}$$

Where T is the sampling period the digital low pass filter is run at. Using T=1/(10Khz)=0.0001 and from the discussion in [1] it is known that $\frac{\omega_f}{\omega} < \frac{1}{\sqrt{2}}$ for stable response of the PLL, therefore, choosing:

$\omega_f = 30$

results in:

$k_1 = 0.00933678$, $k_2 = -0.9813264$

4.2 Simulating the Phase Locked Loop for Varying Conditions

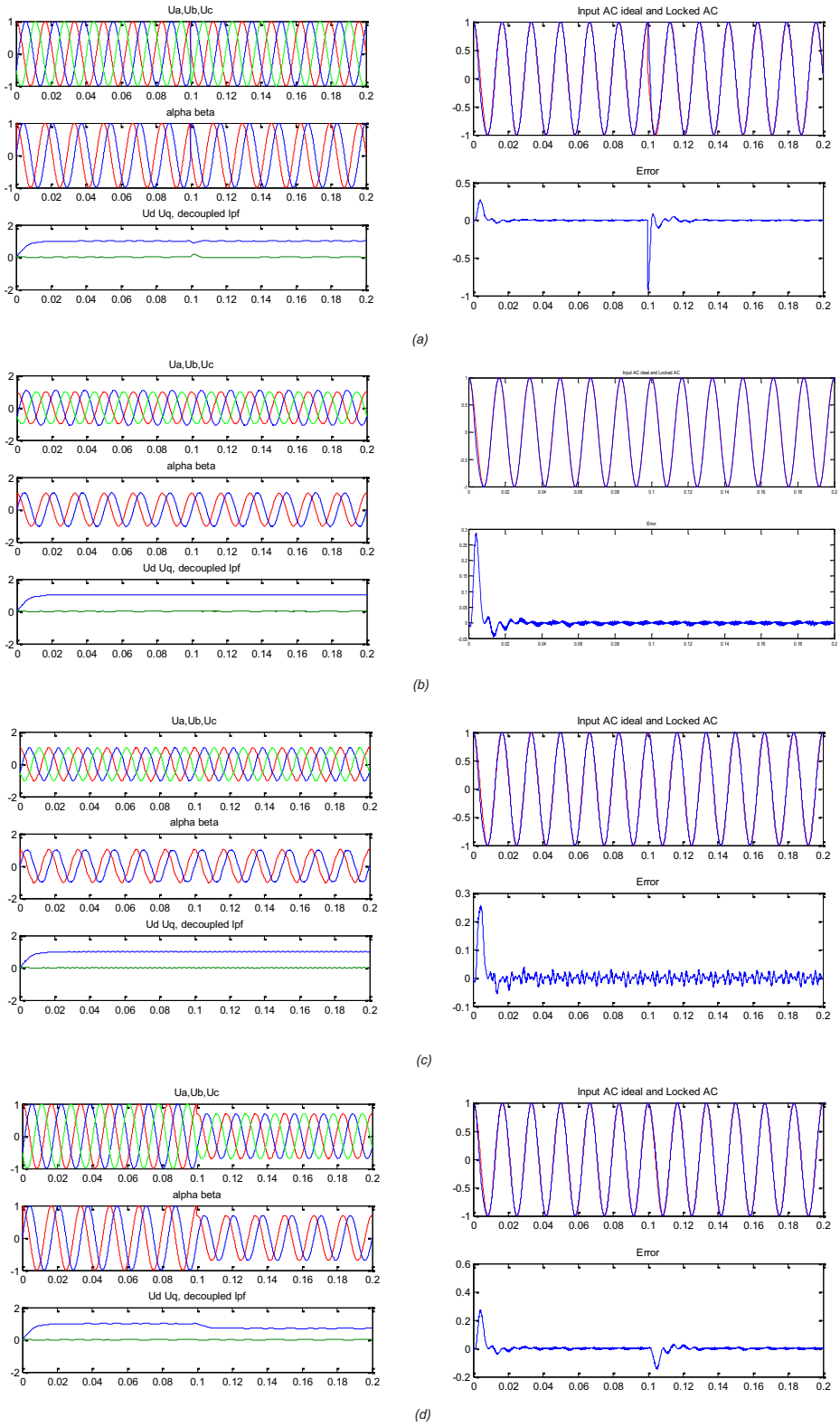
Before coding the SPLL structure, it is essential to simulate the behavior of the PLL for different conditions on the grid. Fixed point processors are used for lower cost in many grid tied converters. IQ Math is a convenient way to look at fixed point numbers with a decimal point. C2000 IQ math library provides built in functions that can simplify handling of the decimal point by the programmer. First, MATLAB is used to simulate and identify the Q point the algorithm needs to be run at. Below is the MATLAB script using the fixed point tool box that tests the PLL algorithm with varying grid condition.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  This software is licensed for use with Texas Instruments C28x
%  family DSCs.  This license was provided to you prior to installing
%  the software.
% ------------------------------------------------------------------
%          Copyright (C) 2010-2012 Texas Instruments, Incorporated.
%                          All Rights Reserved.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
close all;
clc;

%Select numeric type,
T=numerictype('WordLength',32,'FractionLength',22);

%Specify math attributes to the fimath object
F=fimath('RoundMode','floor','OverflowMode','wrap');
F.ProductMode='SpecifyPrecision';
F.ProductWordLength=32;
F.ProductFractionLength=22;
F.SumMode='SpecifyPrecision';
F.SumWordLength=32;
F.SumFractionLength=22;
%specify fipref object, to display warning in cases of overflow and
%underflow
P=fipref;
P.LoggingMode='on';
P.NumericTypeDisplay='none';
P.FimathDisplay='none';

%PLL Modelling starts from here
Fs=10000;
Ts=1/Fs;            %Sampling Time = (1/fs) , Note Ts is related to the frequency
                    % the ISR would run in the solar application as well, assuming
                    % 10Khz control loop for the inverter

Tfinal=0.2;         % Total time the simulation is run for
t=0:Ts:Tfinal;      % time vector
GridFreq=60;        % nominal grid frequency
wn=2*pi*GridFreq;   % nominal frequency in radians
fn=GridFreq;

%generate input signal
wu=2*pi*GridFreq;
theta=rem((t*2*pi*GridFreq),2*pi);
L=length(t);

%generate input signal and create a fi object of it

%CASE 1 : Phase Jump at the Mid Point

for n=1:floor(L/2)
    Ua(n)=cos(theta(n));
    Ub(n)=cos(theta(n)-2*pi/3);
    Uc(n)=cos(theta(n)-4*pi/3);
end
for n=floor(L/2):L
```

```
    Ua(n)=cos(theta(n)+1.5);
    Ub(n)=cos(theta(n)-2*pi/3+1.5);
    Uc(n)=cos(theta(n)-4*pi/3+1.5);
end
Ua_ideal=Ua;

%CASE 2: Unbalanced Grid

% for n=1:floor(L)
% Ua(n)=cos(theta(n));
% Ub(n)=1.1*cos(theta(n)-2*pi/3);
% Uc(n)=cos(theta(n)-4*pi/3);
% end
% Ua_ideal=cos(theta);

%CASE 3: voltage harmonics
% for n=1:floor(L)
% Ua(n)=cos(theta(n))+0.05*cos(5*theta(n));
% Ub(n)=cos(theta(n)-2*pi/3)+0.05*(cos(5*(theta(n)-2*pi/3)));
% Uc(n)=cos(theta(n)-4*pi/3)+0.05*(cos(5*(theta(n)-4*pi/3)));
% end
% Ua_ideal=cos(theta);

% CASE 4: voltage dips and swells
% %
% for n=1:floor(L/2)
%     Ua(n)=cos(theta(n));
%     Ub(n)=cos(theta(n)-2*pi/3);
%     Uc(n)=cos(theta(n)-4*pi/3);
% end
% for n=floor(L/2):L
%     Ua(n)=0.7*cos(theta(n));
%     Ub(n)=0.7*cos(theta(n)-2*pi/3);
%     Uc(n)=0.7*cos(theta(n)-4*pi/3);
% end
% Ua_ideal=cos(theta);

Ualpha=[0,0];
Ubeta=[0,0];

Ud_plus=[0,0];
Uq_plus=[0,0];

Ud_minus=[0,0];
Uq_minus=[0,0];

Ud_plus_decoupl=[0,0];
Uq_plus_decoupl=[0,0];

Ud_minus_decoupl=[0,0];
Uq_minus_decoupl=[0,0];

Ud_plus_decoupl_lpf=[0,0];
Uq_plus_decoupl_lpf=[0,0];

Ud_minus_decoupl_lpf=[0,0];
Uq_minus_decoupl_lpf=[0,0];

y=[0,0];
x=[0,0];
w=[0,0];
z=[0,0];

ylf =[0,0];
u_q =[0,0];
Theta=[0,0,0];
```

```
fo=0;

Ua=fi(Ua,T,F);
Ub=fi(Ub,T,F);
Uc=fi(Uc,T,F);
Ua_ideal=fi(Ua_ideal,T,F);

%declare arrays used by the PLL process
Ualpha  =fi(Ualpha,T,F);
Ubeta  =fi(Ubeta,T,F);
Ud_plus  =fi(Ud_plus,T,F);
Uq_plus  =fi(Uq_plus,T,F);
Ud_minus  =fi(Ud_minus,T,F);
Uq_minus  =fi(Uq_minus,T,F);
Ud_plus_decoupl  =fi(Ud_plus_decoupl,T,F);
Uq_plus_decoupl  =fi(Uq_plus_decoupl,T,F);
Ud_minus_decoupl =fi(Ud_minus_decoupl,T,F);
Uq_minus_decoupl =fi(Uq_minus_decoupl,T,F);
Ud_plus_decoupl_lpf  =fi(Ud_plus_decoupl_lpf,T,F);
Uq_plus_decoupl_lpf  =fi(Uq_plus_decoupl_lpf,T,F);
Ud_minus_decoupl_lpf =fi(Ud_minus_decoupl_lpf,T,F);
Uq_minus_decoupl_lpf =fi(Uq_minus_decoupl_lpf,T,F);
y=fi(y,T,F);
x=fi(x,T,F);
w=fi(w,T,F);
z=fi(z,T,F);
ylf=fi(ylf,T,F);
u_q=fi(u_q,T,F);
Theta=fi(Theta,T,F);
fo=fi(fo,T,F);

% simulate the PLL process
for n=3:L    % No of iteration of the PLL process in the simulation time


    Ualpha(n) =fi((2.0/3.0),T,F)*(Ua(n)-fi(0.5,T,F)*(Ub(n)+Uc(n)));
    Ubeta(n)  =fi((2.0/3.0)*0.866,T,F)*(Ub(n)-Uc(n));

    s1=sin(Theta(n));
    c1=cos(Theta(n));
    s2=sin(fi(2,T,F)*Theta(n));
    c2=cos(fi(2,T,F)*Theta(n));

    Ud_plus(n)=Ualpha(n)*c1+Ubeta(n)*s1;
    Uq_plus(n)=-Ualpha(n)*s1+Ubeta(n)*c1;

    Ud_minus(n)=Ualpha(n)*c1-Ubeta(n)*s1;
    Uq_minus(n)=Ualpha(n)*s1+Ubeta(n)*c1;

    Ud_plus_decoupl(n)=Ud_plus(n)-Ud_minus_decoupl_lpf(n-1)*c2-Uq_minus_decoupl_lpf(n-1)*s2;
    Uq_plus_decoupl(n)=Uq_plus(n)+Ud_minus_decoupl_lpf(n-1)*s2-Uq_minus_decoupl_lpf(n-1)*c2;

    Ud_minus_decoupl(n)=Ud_minus(n)-Ud_plus_decoupl_lpf(n-1)*c2+Uq_plus_decoupl_lpf(n-1)*s2;
    Uq_minus_decoupl(n)=Uq_minus(n)-Ud_plus_decoupl_lpf(n-1)*s2-Uq_plus_decoupl_lpf(n-1)*c2;

k1=fi(0.00933678,T,F);
k2=fi(-0.9813264,T,F);

    y(n)=Ud_plus_decoupl(n)*k1-k2*y(n-1);
    Ud_plus_decoupl_lpf(n) = y(n)+y(n-1);

    x(n)=Uq_plus_decoupl(n)*k1-k2*x(n-1);
    Uq_plus_decoupl_lpf(n) = x(n)+x(n-1);

    w(n)=Ud_minus_decoupl(n)*k1-k2*w(n-1);
    Ud_minus_decoupl_lpf(n) = w(n)+w(n-1);
```

```
        z(n)=Uq_minus_decoupl(n)*k1-k2*z(n-1);
        Uq_minus_decoupl_lpf(n) = z(n)+z(n-1);


        u_q(1)=Uq_plus_decoupl(n);

        %Loop Filter
        ylf(1)=ylf(2)+fi(167.9877775,T,F)*u_q(1)-fi(165.2122225,T,F)*u_q(2);

        %update u_q for future use
        u_q(2)=u_q(1);

        ylf(2)=ylf(1);

        %update output frequency
        fo=fn+ylf(1);

        Theta(n+1)=Theta(n)+fi(2*pi,T,F)*(fo*Ts);

        Theta(n+1)=Theta(n)+fi(2*pi,T,F)*(fo*Ts);

        if(Theta(n+1)>=(fi(2*pi,T,F)))
            Theta(n+1)=Theta(n+1)-fi(2*pi,T,F);
        end

    end

    figure,subplot(3,1,1),plot(t,Ua,'r',t,Ub,'b',t,Uc,'g'),title('Ua,Ub,Uc');
    subplot(3,1,2),plot(t,Ualpha,'r',t,Ubeta),title('alpha beta');
    subplot(3,1,3),plot(t,Ud_plus_decoupl_lpf(1:L),t,Uq_plus_decoupl_lpf(1:L)), title('Ud Uq,
    decoupled lpf') ;

    figure,subplot(2,1,1),plot(t,Ua_ideal,'r',t,cos(Theta(1:L)),'b'),title('Input AC ideal and Locked
    AC');
    subplot(2,1,2),plot(t,Ua_ideal-cos(Theta(1:L))),title('Error');
```



A Phase Jump of 90 degrees

B Voltage Imbalance of 10% on one phase

C 5% 5th Harmonic content

D Amplitude change (Voltage Sags and Dips)

Figure 9. PLL Response to Varying Grid Conditions

4.3 Implementing PLL on C2000 Controller Using IQ Math

As the regular inverter typically uses IQ24, and as shown in the above section, the Q point chosen for the PLL is IQ22, the code for the PLL can be written as follows.

SPLL_3ph_DDSRF_IQ.h Header File:

```
#ifndef SPLL_3ph_DDSRF_IQ_H_
#define SPLL_3ph_DDSRF_IQ_H_

#define SPLL_DDSRF_Q _IQ22
#define SPLL_DDSRF_Qmpy _IQ22mpy

//*********** Structure Definition ********//
typedef struct{
            int32    B1_lf;
            int32    B0_lf;
            int32    A1_lf;
}SPLL_3ph_DDSRF_IQ_LPF_COEFF;

typedef struct{
            int32 d_p;
            int32 d_n;
            int32 q_p;
            int32 q_n;

            int32 d_p_decoupl;
            int32 d_n_decoupl;
            int32 q_p_decoupl;
            int32 q_n_decoupl;

            int32 cos_2theta;
            int32 sin_2theta;

            int32 y[2];
            int32 x[2];
            int32 w[2];
            int32 z[2];
            int32 k1;
            int32 k2;
            int32 d_p_decoupl_lpf;
            int32 d_n_decoupl_lpf;
            int32 q_p_decoupl_lpf;
            int32 q_n_decoupl_lpf;

            int32 v_q[2];
            int32 theta[2];
            int32 ylf[2];
            int32 fo;
            int32 fn;
            int32 delta_T;
            SPLL_3ph_DDSRF_IQ_LPF_COEFF lpf_coeff;
}SPLL_3ph_DDSRF_IQ;

//*********** Function Declarations *******//
void SPLL_3ph_DDSRF_IQ_init(int Grid_freq, long DELTA_T, long k1, long k2, SPLL_3ph_DDSRF_IQ
*spll);
void SPLL_3ph_DDSRF_IQ_FUNC(SPLL_3ph_DDSRF_IQ *spll_obj);

//*********** Macro Definition **********//
#define SPLL_3ph_DDSRF_IQ_MACRO(spll_obj)
                                                                                    \
            spll_obj.d_p_decoupl=spll_obj.d_p -
 SPLL_DDSRF_Qmpy(spll_obj.d_n_decoupl_lpf,spll_obj.cos_2theta) -
 SPLL_DDSRF_Qmpy(spll_obj.q_n_decoupl,spll_obj.sin_2theta);     \
            spll_obj.q_p_decoupl=spll_obj.q_p +
SPLL_DDSRF_Qmpy(spll_obj.d_n_decoupl_lpf,spll_obj.sin_2theta) -
```

```
 SPLL_DDSRF_Qmpy(spll_obj.q_n_decoupl,spll_obj.cos_2theta);     \
            spll_obj.d_n_decoupl=spll_obj.d_n -
 SPLL_DDSRF_Qmpy(spll_obj.d_p_decoupl_lpf,spll_obj.cos_2theta) +
SPLL_DDSRF_Qmpy(spll_obj.q_p_decoupl,spll_obj.sin_2theta);      \
            spll_obj.q_n_decoupl=spll_obj.q_n -
 SPLL_DDSRF_Qmpy(spll_obj.d_p_decoupl_lpf,spll_obj.sin_2theta) -
 SPLL_DDSRF_Qmpy(spll_obj.q_p_decoupl,spll_obj.cos_2theta);      \
            spll_obj.y[1]=SPLL_DDSRF_Qmpy(spll_obj.d_p_decoupl,spll_obj.k1)-
SPLL_DDSRF_Qmpy(spll_obj.y[0],spll_obj.k2); \
            spll_obj.d_p_decoupl_lpf=spll_obj.y[1]+spll_obj.y[0];  \
            spll_obj.y[0]=spll_obj.y[1]; \
            spll_obj.x[1]=SPLL_DDSRF_Qmpy(spll_obj.q_p_decoupl,spll_obj.k1)-
SPLL_DDSRF_Qmpy(spll_obj.x[0],spll_obj.k2); \
            spll_obj.q_p_decoupl_lpf=spll_obj.x[1]+spll_obj.x[0]; \
            spll_obj.x[0]=spll_obj.x[1];  \
            spll_obj.w[1]=SPLL_DDSRF_Qmpy(spll_obj.d_n_decoupl,spll_obj.k1)-
SPLL_DDSRF_Qmpy(spll_obj.w[0],spll_obj.k2);  \
            spll_obj.d_n_decoupl_lpf=spll_obj.w[1]+spll_obj.w[0]; \
            spll_obj.w[0]=spll_obj.w[1]; \
            spll_obj.z[1]=SPLL_DDSRF_Qmpy(spll_obj.q_n_decoupl,spll_obj.k1)-
SPLL_DDSRF_Qmpy(spll_obj.z[0],spll_obj.k2);  \
            spll_obj.q_n_decoupl_lpf=spll_obj.z[1]+spll_obj.z[0];  \
            spll_obj.z[0]=spll_obj.z[1]; \
            spll_obj.v_q[0]=spll_obj.q_p_decoupl; \

spll_obj.ylf[0]=spll_obj.ylf[1]+SPLL_DDSRF_Qmpy(spll_obj.lpf_coeff.B0_lf,spll_obj.v_q[0])+SPLL_DDS
RF_Qmpy(spll_obj.lpf_coeff.B1_lf,spll_obj.v_q[1]);                                          \
            spll_obj.ylf[1]=spll_obj.ylf[0]; \
            spll_obj.v_q[1]=spll_obj.v_q[0]; \
            spll_obj.fo=spll_obj.fn+spll_obj.ylf[0];


            spll_obj.theta[0]=spll_obj.theta[1]+SPLL_DDSRF_Qmpy
            (SPLL_DDSRF_Qmpy(spll_obj.fo,spll_obj.delta_T),SPLL_DDSRF_Q(2*3.1415926));
            if(spll_obj.theta[0]>SPLL_DDSRF_Q(2*3.1415926)) \
                        spll_obj.theta[0]=spll_obj.theta[0]-SPLL_DDSRF_Q(2*3.1415926); \
            spll_obj.theta[1]=spll_obj.theta[0]; \
#endif /* SPLL_3ph_DDSRF_IQ_H_ */
```

SPLL_3ph_DDSRF_IQ.c source file:

```
#include "Solar_IQ.h"

//*********** Structure Init Function ****//
void SPLL_3ph_DDSRF_IQ_init(int Grid_freq, long DELTA_T, long k1, long k2, SPLL_3ph_DDSRF_IQ
*spll_obj)
{
            spll_obj->d_p=SPLL_DDSRF_Q(0.0);
            spll_obj->d_n=SPLL_DDSRF_Q(0.0);

            spll_obj->q_p=SPLL_DDSRF_Q(0.0);
            spll_obj->q_n=SPLL_DDSRF_Q(0.0);

            spll_obj->d_p_decoupl=SPLL_DDSRF_Q(0.0);
            spll_obj->d_n_decoupl=SPLL_DDSRF_Q(0.0);

            spll_obj->q_p_decoupl=SPLL_DDSRF_Q(0.0);
            spll_obj->q_n_decoupl=SPLL_DDSRF_Q(0.0);

            spll_obj->d_p_decoupl_lpf=SPLL_DDSRF_Q(0.0);
            spll_obj->d_n_decoupl_lpf=SPLL_DDSRF_Q(0.0);

            spll_obj->q_p_decoupl_lpf=SPLL_DDSRF_Q(0.0);
            spll_obj->q_n_decoupl_lpf=SPLL_DDSRF_Q(0.0);

            spll_obj->y[0]=SPLL_DDSRF_Q(0.0);
```

```
            spll_obj->y[1]=SPLL_DDSRF_Q(0.0);

            spll_obj->x[0]=SPLL_DDSRF_Q(0.0);
            spll_obj->x[1]=SPLL_DDSRF_Q(0.0);

            spll_obj->w[0]=SPLL_DDSRF_Q(0.0);
            spll_obj->w[1]=SPLL_DDSRF_Q(0.0);

            spll_obj->z[0]=SPLL_DDSRF_Q(0.0);
            spll_obj->z[1]=SPLL_DDSRF_Q(0.0);

            spll_obj->k1=k1;
            spll_obj->k2=k2;

            spll_obj->v_q[0]=SPLL_DDSRF_Q(0.0);
            spll_obj->v_q[1]=SPLL_DDSRF_Q(0.0);

            spll_obj->ylf[0]=SPLL_DDSRF_Q(0.0);
            spll_obj->ylf[1]=SPLL_DDSRF_Q(0.0);

            spll_obj->fo=SPLL_DDSRF_Q(0.0);
            spll_obj->fn=SPLL_DDSRF_Q(Grid_freq);

            spll_obj->theta[0]=SPLL_DDSRF_Q(0.0);
            spll_obj->theta[1]=SPLL_DDSRF_Q(0.0);

            // loop filter coefficients for 20kHz
            spll_obj->lpf_coeff.B0_lf=_IQ22(166.9743);
            spll_obj->lpf_coeff.B1_lf=_IQ22(-166.266);
            spll_obj->lpf_coeff.A1_lf=_IQ22(-1.0);

            spll_obj->delta_T=DELTA_T;
}

//*********** Function Definition ********//
void SPLL_3ph_DDSRF_IQ_FUNC(SPLL_3ph_DDSRF_IQ *spll_obj)
{
            // before calling this routine run the ABC_DQ0_Pos_Neg and update the values for
d_p,d_n,q_p,q_n
            // and update the cos_2theta and sin_2theta values with the prev angle

            //-------------------------//
            // Decoupling Network      //
            //-------------------------//
            spll_obj->d_p_decoupl=spll_obj->d_p - SPLL_DDSRF_Qmpy(spll_obj-
>d_n_decoupl_lpf,spll_obj->cos_2theta) - SPLL_DDSRF_Qmpy(spll_obj->q_n_decoupl,spll_obj-
>sin_2theta);
            spll_obj->q_p_decoupl=spll_obj->q_p + SPLL_DDSRF_Qmpy(spll_obj-
>d_n_decoupl_lpf,spll_obj->sin_2theta) - SPLL_DDSRF_Qmpy(spll_obj->q_n_decoupl,spll_obj-
>cos_2theta);

            spll_obj->d_n_decoupl=spll_obj->d_n - SPLL_DDSRF_Qmpy(spll_obj-
>d_p_decoupl_lpf,spll_obj->cos_2theta) + SPLL_DDSRF_Qmpy(spll_obj->q_p_decoupl,spll_obj-
>sin_2theta);
            spll_obj->q_n_decoupl=spll_obj->q_n - SPLL_DDSRF_Qmpy(spll_obj-
>d_p_decoupl_lpf,spll_obj->sin_2theta) - SPLL_DDSRF_Qmpy(spll_obj->q_p_decoupl,spll_obj-
>cos_2theta);

            //-------------------------//
            // Low pass filter         //
            //-------------------------//

            spll_obj->y[1]=SPLL_DDSRF_Qmpy(spll_obj->d_p_decoupl,spll_obj->k1)-
SPLL_DDSRF_Qmpy(spll_obj->y[0],spll_obj->k2);
            spll_obj->d_p_decoupl_lpf=spll_obj->y[1]+spll_obj->y[0];
            spll_obj->y[0]=spll_obj->y[1];
```

```
            spll_obj->x[1]=SPLL_DDSRF_Qmpy(spll_obj->q_p_decoupl,spll_obj->k1)-
SPLL_DDSRF_Qmpy(spll_obj->x[0],spll_obj->k2);
            spll_obj->q_p_decoupl_lpf=spll_obj->x[1]+spll_obj->x[0];
            spll_obj->x[0]=spll_obj->x[1];

            spll_obj->w[1]=SPLL_DDSRF_Qmpy(spll_obj->d_n_decoupl,spll_obj->k1)-
SPLL_DDSRF_Qmpy(spll_obj->w[0],spll_obj->k2);
            spll_obj->d_n_decoupl_lpf=spll_obj->w[1]+spll_obj->w[0];
            spll_obj->w[0]=spll_obj->w[1];

            spll_obj->z[1]=SPLL_DDSRF_Qmpy(spll_obj->q_n_decoupl,spll_obj->k1)-
SPLL_DDSRF_Qmpy(spll_obj->z[0],spll_obj->k2);
            spll_obj->q_n_decoupl_lpf=spll_obj->z[1]+spll_obj->z[0];
            spll_obj->z[0]=spll_obj->z[1];

            spll_obj->v_q[0]=spll_obj->q_p_decoupl;

            //---------------------------------//
            // Loop Filter                     //
            //---------------------------------//
            spll_obj->ylf[0]=spll_obj->ylf[1]+SPLL_DDSRF_Qmpy(spll_obj->lpf_coeff.B0_lf,spll_obj-
>v_q[0])+SPLL_DDSRF_Qmpy(spll_obj->lpf_coeff.B1_lf,spll_obj->v_q[1]);
            spll_obj->ylf[1]=spll_obj->ylf[0];
            spll_obj->v_q[1]=spll_obj->v_q[0];

            //---------------------------------//
            // VCO                             //
            //---------------------------------//
            spll_obj->fo=spll_obj->fn+spll_obj->ylf[0];

            spll_obj->theta[0]=spll_obj->theta[1]+SPLL_DDSRF_Qmpy(SPLL_DDSRF_Qmpy(spll_obj-
>fo,spll_obj->delta_T),SPLL_DDSRF_Q(2*3.1415926));
            if(spll_obj->theta[0]>SPLL_DDSRF_Q(2*3.1415926))
                        spll_obj->theta[0]=spll_obj->theta[0]-SPLL_DDSRF_Q(2*3.1415926);

vspll_obj->theta[1]=spll_obj->theta[0];
}
```

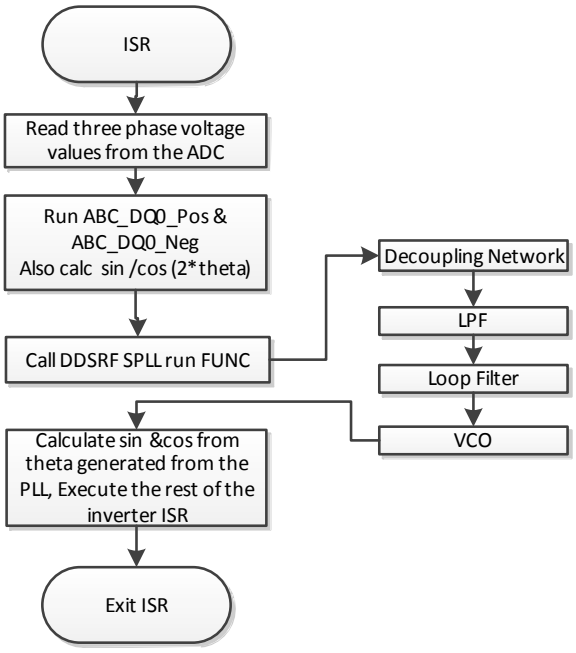


Figure 10. DDSRF SPLL Flowchart

To use this block in an end application, declare objects for the SPLL structure, loop filter coefficients and notch filter coefficients:

```
SPLL_3ph_DDSRF spll1;
```

Call the SPLL_3ph_DDSRF_IQ_init routine with the frequency of the ISR where SPLL will be executed in as parameter and the grid frequency and then call the notch filter update coefficient update routine.

```
SPLL_3ph_DDSRF_IQ_init(GRID_FREQ,_IQ22((float)(1.0/ISR_FREQUENCY)),SPLL_DDSRF_Q(0.00933678),
SPLL_DDSRF_Q(-0.9813264),&spll1);
```

In the ISR, read the sinusoidal input voltage from the ADC and feed it into the SPLL block; write to invsine value with the sinus of the current grid angle. This can then be used in control operations.

```
spll1.d_p =abc_dq0_pos1.d>>2;          // Convert Q24 to Q22
spll1.q_p =abc_dq0_pos1.q<< 2;         // Convert Q24 to Q22
spll1.d_n =abc_dq0_neg1.d<<2;          // Convert Q24 to Q22
spll1.q_n =abc_dq0_neg1.q<<2;          // Convert Q24 to Q22
spll1.q_n =abc_dq0_neg1.q(&spll1);
spll1.cos_2theta = IQ22 cos (_IQ22mpy (_IQ22(2), Spll1.theta[0]));
Spll1.sin_2theta = IQ22 sin (_IQ22mpy (_IQ22(2), Spll1.theta[0]));
//Spll call
SPLL_3ph_DDSRF_IQ_FUNC (&Spll1);
```

5 Solar Library and ControlSuite™

C2000 provides software for development of control application using the C2000 device. The software is available for download from www.ti.com/controlSUITE.

Various libraries like the IQ math Library and Solar Library simplify the design and developments of algorithms on the C2000 device. The IQ math library provides a convenient way to easily translate the fixed point tool box code into controller code.

The solar library provides the software blocks and more information that is discussed in this application report.

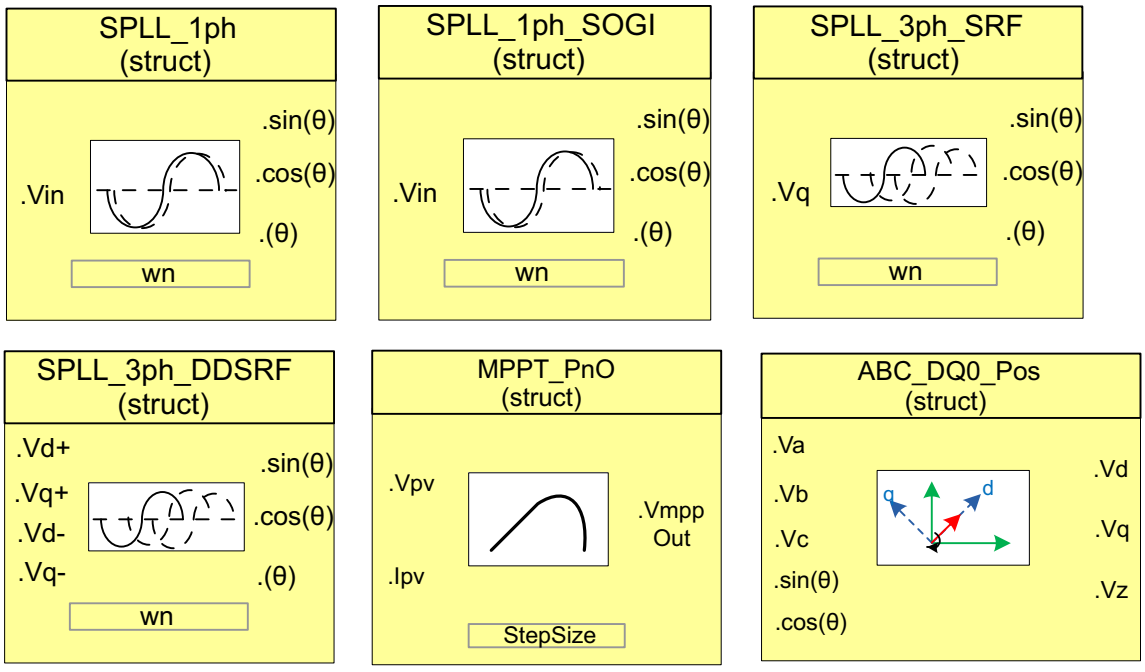


Figure 11. Solar Library and ControlSuite

6 References

1. P. R. e. al, "Decoupled Double Synchronous Reference Frame PLL for Power Converters Control," IEEE Transaction on Power Electronics, vol. 22, no. 2, 2007.
2. Blaabjerg, F., R. Teodorescu, M. Liserre, and A. Timbus, "Overview of Control and Grid Synchronization for Distributed Power Generation Systems," IEEE Transactions on Industrial Electronics, vol 53, no 5, Oct 2006.
3. P. R. &. R. T. Marco Liserre, Grid Converters for Photovoltaic and Wind Power Systems, John Wiley & Sons Ltd., 2011.
4. *TMS320F28032, TMS320F28033, TMS320F28034, TMS320F28035, Piccolo Microcontrollers Data Manual* (SPRS584), available at: www.ti.com.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265