**Digital Phase Locked Loop (PLL): DSP Implementation**

DSP Lab, Spring 2013

4/23/2013

Jacobs University Bremen

Joshan Chaudhary, Grishma Raj Pandeya

**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!).**

pll\_state\_def \*pll\_init(float f0, float T, float K, float damp, float mult)

{

pll\_state\_def \*s;

float wn, tau1, tau2;

/\* Allocate a new pll\_state\_def structure. Holds state and parameters. \*/

if ((s = (pll\_state\_def \*)MEM\_calloc(PLL\_SEG\_ID, sizeof(pll\_state\_def), PLL\_BUFFER\_ALIGN)) == NULL)

{

SYS\_error("Unable to create state structure for PLL.", SYS\_EUSER, 0);

return(0);

}

/\* Copy input parameters \*/

s->f0 = f0;

s->damp\_fact = damp;

s->K = K;

s->mult = mult;

/\* Compute the filter coefficients \*/

/\* Add your code here !!! \*/

wn = 2.0\*M\_PI/100.0;

tau1 = K/(wn\*wn);

tau2 = 2.0\*damp/wn - 1.0/K;

s->a1 = -(T - 2.0\*tau1)/(T + 2.0\*tau1);

s->b0 = (T + 2.0\*tau2)/(T + 2.0\*tau1);

s->b1 = (T - 2.0\*tau2)/(T + 2.0\*tau1);

/\* Set state variables (initially all 0) \*/

s->z\_nm1 = 0;

s->v\_nm1 = 0;

s->x\_nm1 = 0;

s->y\_nm1 = 0;

s->accum = 0;

s->accum2 = 0;

/\* Set initial block amplitude (cannot be 0!) \*/

s->Ap = 1.0;

return(s);

}

/\*---------------------------------------------------------------------------

\* pll

\* PLL process function.

\*--------------------------------------------------------------------------\*/

void pll(pll\_state\_def \*s, const float x\_in[], float y\_out[])

{

int n;

float A;

float x\_n;

float z\_n;

float v\_n;

float y\_n;

float y\_n2;

float daccum;

/\* Add other temporary variables as needed. \*/

/\* Do not put any arrays as local variables! \*/

/\*

\* If signal level is below some threshold, make Ap large, which has the

\* effect of just doing holdover mode.

\*/

if (s->Ap < 1.0E-3)

{

s->Ap = 100.;

}

A = 0.0; /\* Variable for computing amplitude \*/

for (n=0; n<BUFFER\_SAMPLES; n++)

{

/\* Input sample (input reference) \*/

/\* Take the sign of the input signal. \*/

x\_n = x\_in[n];

/\* Estimate amplitude from summed |x| \*/

A = A + fabs(x\_n);

/\* Add your code here to do PLL operation !!! \*/

/\* Code should generate y\_n from x\_n. \*/

z\_n = s->x\_nm1/s->Ap \* s->y\_nm1;

v\_n = s->a1\*s->v\_nm1 + s->b0\*z\_n + s->b1\*s->z\_nm1;

/\* Update the accumulator \*/

daccum = s->f0 - s->K/(2.0\*M\_PI)\*v\_n;

s->accum = s->accum + daccum ;

s->accum = s->accum - floor(s->accum);

y\_n = sin\_table[(int)(((float)SIN\_TABLE\_SIZE \* s->accum))];

/\* Put output sample \*/

if (s->mult == 2.0) {

s->accum2 = s->accum2 + daccum \* s->mult;

s->accum2 = s->accum2 - floor(s->accum2);

y\_n2 = sin\_table[(int)(((float)SIN\_TABLE\_SIZE \* s->accum2))];

y\_out[n] = y\_n2\*0.2;

}

else if (s->mult == 0.5) {

s->accum2 = s->accum2 + daccum;

s->accum2 = s->accum2 - 2.0\*floor(s->accum2/2.0);

y\_n2 = sin\_table[(int)(((float)SIN\_TABLE\_SIZE\* 0.5 \* s->accum2))];

y\_out[n] = y\_n2\*0.2;

}

else {

y\_out[n] = y\_n\*0.2;

}

//y\_out[n] = y\_n;

/\* Shift current variables to previous values. \*/

s->z\_nm1 = z\_n;

s->v\_nm1 = v\_n;

s->x\_nm1 = x\_n;

s->y\_nm1 = y\_n;

}

/\* Get amplitude estimate for next block (compute from A) \*/

s->Ap = A/(BUFFER\_SAMPLES)/(2/M\_PI);

}

3. A screen shot (like from the Scope program) showing that the PLL is tracking your input signal.

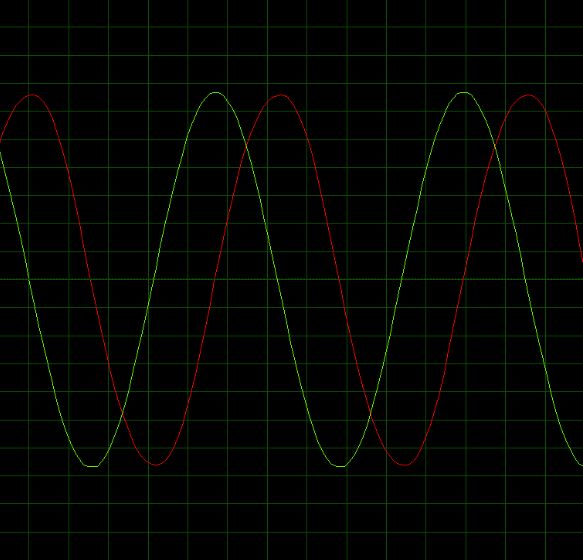


Fig: tracking at 800 Hz

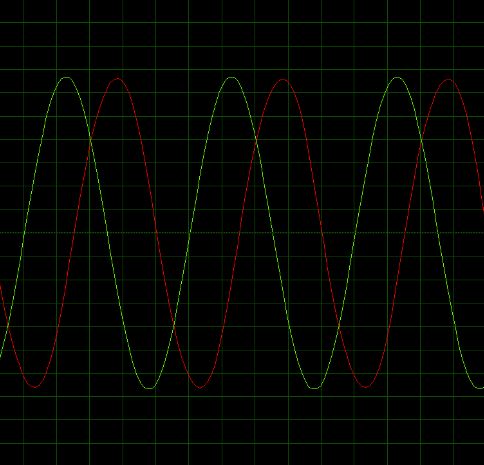


Fig: tracking at 1000 Hz

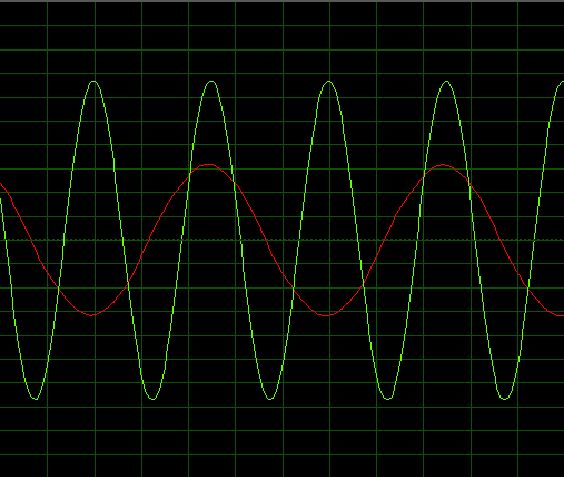


Fig: tracking for at half the input freq

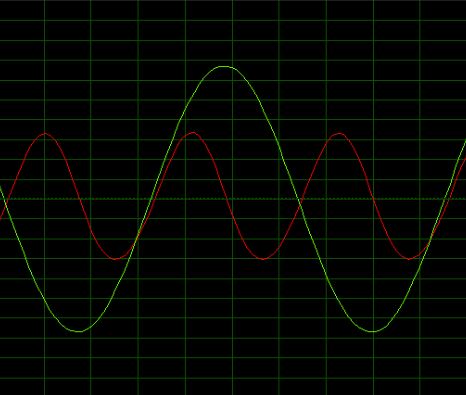


Fig: tracking at double the input frequency.