khu\_sensor\_2048MHz

float \_Old\_HFP=0.0, \_Old\_Data=0.0;

main()

{

float ECG\_LPF;

float ECG\_notch;

float ECG\_HPF;

int32\_t ulData; //input data

ECG\_LPF = iir\_LPF\_10Hz\_Sample\_250Hz((float)ulData);

ECG\_notch = iir\_Notch\_60Hz\_Sample\_250Hz(ECG\_LPF);

ECG\_HPF = iir\_HPF(ECG\_notch,\_Old\_Data, \_Old\_HFP, 5.0, 250.0);

\_Old\_Data = ECG\_notch;

\_Old\_HFP = ECG\_HPF;

}

#define NCoef 5 // LPF Order

float iir\_LPF\_10Hz\_Sample\_250Hz(float NewSample)

{

float ACoef[NCoef+1] = {

0.01336978300344553300,

0.02673956600689106700,

0.01336978300344553300

};

float BCoef[NCoef+1] = {

1.00000000000000000000,

-1.64745998107697660000,

0.70089678118840248000

};

static float y\_LPF[NCoef+1]; //output samples

static float x\_LPF[NCoef+1]; //input samples

int n;

//shift the old samples

for(n=NCoef; n>0; n--) {

x\_LPF[n] = x\_LPF[n-1];

y\_LPF[n] = y\_LPF[n-1];

}

//Calculate the new output

x\_LPF[0] = NewSample;

y\_LPF[0] = ACoef[0] \* x\_LPF[0];

for(n=1; n<=NCoef; n++)

y\_LPF[0] += ACoef[n] \* x\_LPF[n] - BCoef[n] \* y\_LPF[n];

return y\_LPF[0];

}

float iir\_Notch\_60Hz\_Sample\_250Hz(float NewSample)

{

float ACoef[NCoef\_Stop+1] =

{

0.90058301430561660000,

-0.22683676727047436000,

1.81544980872044200000,

-0.22683676727047436000,

0.90058301430561660000

};

float BCoef[NCoef\_Stop+1] =

{

1.00000000000000000000,

-0.23849243624947844000,

1.80168874042915350000,

-0.21430582988195795000,

0.80794959133320132000

};

static float y[NCoef\_Stop+1]; //output samples

static float x[NCoef\_Stop+1]; //input samples

int n;

//shift the old samples

for(n=NCoef\_Stop; n>0; n--) {

x[n] = x[n-1];

y[n] = y[n-1];

}

//Calculate the new output

x[0] = NewSample;

y[0] = ACoef[0] \* x[0];

for(n=1; n<=NCoef\_Stop; n++)

y[0] += ACoef[n] \* x[n] - BCoef[n] \* y[n];

return y[0];

}

#define FILTER\_CAP 0.0001

float iir\_HPF(float NewSample, float OldSample, float OldResult, float cutoff, float SPR)

{

float tau, ts, omega, filter\_R, HPF;

ts = 1.0 / SPR;

filter\_R = 1.0 / (cutoff \* PI2 \* FILTER\_CAP);

tau = FILTER\_CAP \* filter\_R;

omega = tau / (tau + ts);

HPF = omega \* OldResult + omega \* (NewSample-OldSample);

return HPF;

}