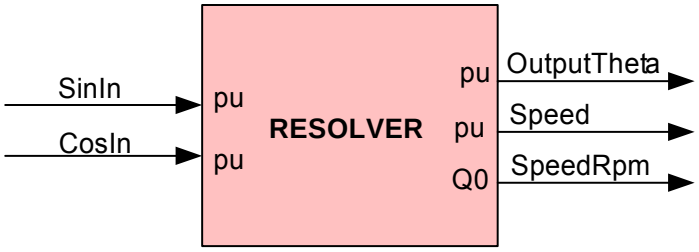
RESOLVER	*Position and Speed Calculations for Resolver Sensor*

Description

This software module calculates the rotor position and motor speed based on two resolver signals (i.e., sin and cos signals).



Availability

This IQ module is available in one interface format:

1) The C interface version

Module Properties

Type: Target Independent, Application Dependent

Target Devices: x281x or x280x

C Version File Names: resolver.c, resolver.h

IQmath library files for C: IQmathLib.h, IQmath.lib

Item	C version	Comments
Code Size□ (x281x/x280x)	877/877 words	
Data RAM	0 words•	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

• Each pre-initialized "_iq" RESOLVER structure consumes 157 words in the data memory

□ Code size mentioned here is the size of the ***calc()*** function

C Interface

Object Definition

The structure of RESOLVER object is defined by following structure definition

```
typedef struct {
      _iq SinIn;                // Input: Measured sine signal (pu)
      _iq CosIn;                // Input: Measured cos signal (pu)
      _iq FilteredSin;          // Variable: Filtered sine signal (pu)
      _iq FilteredCos;          // Variable: Filtered cos signal (pu)
      _iq DemodSin;             // Variable: Demodulated sine signal (pu)
      _iq DemodCos;             // Variable: Demodulated cos signal (pu)
      _iq PreviousSin[18];      // History: Past sine signals (pu)
      _iq PreviousCos[18];      // History: Past cos signals (pu)
      _iq FilterGain[18];       // Parameter: FIR 17-order low-pass, Hamming Windows
      _iq OutputTheta;          // Output: Motor electrical position angle (pu)
      _iq FilterTheta;          // Variable: FilterTheta = FilterTheta1 - FilterTheta2 (pu)
      _iq FilterTheta1;         // Variable: Mechanical angle by filtered sin/cos signal (pu)
      _iq FilterTheta2;         // Variable Mechanical angle by filtered cos/sin signal (pu)
     Uint32 DemodConst;         // Variable: Downsampling by factor K
      _iq SignalGain;           // Parameter: Sin and cos gain (pu)
      _iq SpeedGain;            // Parameter: Speed calculation gain (pu)
     Uint32 CounterMax;         // Parameter: Maximum count for finding zero crossing (Q0)
      _iq AmplifiedSin;         // Variable: Amplified sine signal (pu)
      _iq AmplifiedCos;         // Variable: Amplified cos signal (pu)
      _iq PreviousDemodSin[4];  // History: Past demodulated sine signals (pu)
      _iq PreviousDemodCos[4];  // History: Past demodulated cos signals (pu)
     Uint32 ThetaCounter;       // Variable: Counter for finding zero crossing (Q0)
     Uint32 ThetaCounterMax; // Variable: Maximum counter for finding zero crossing (Q0)
      _iq ElecTheta;            // Variable: Electrical angle before angle compensation (pu)
      _iq MechTheta;            // Variable: Mechanical angle before angle comp. (pu)
      _iq Speed;                // Output: Speed in per-unit  (pu)
      _iq OldTheta;             // History: Mechanical angle at previous step (pu)
      Uint32 PolePairs;         // Parameter: Number of pole pairs (Q0)
      Uint32 BaseRpm;           // Parameter: Base speed in rpm (Q0)
      _iq21 K1;                 // Parameter: Constant for differentiator (Q21)
      int32 SpeedRpm;           // Output : Motor speed in rpm  (Q0)
      Uint32 SignalCounter; // Variable: Sign change counter 0-3 (Q0)
      _iq RefTheta;             // Variable: Filtered mechanical angle before angle comp. (pu)
     Uint32 SignFlag;           // Variable: Sign change of FilterTheta (pu)
     Uint32 OldSignFlag;        // History: Sign change of FilterTheta at previous step (pu)
     _iq CalibratedAngle;   // Parameter: Offset between resolver's signal & winding (pu)
      void (*calc)();           // Pointer to the calculation function
      } RESOLVER;               // Data type created


typedef RESOLVER *RESOLVER_handle;
```

Module Terminal Variables/Functions

Item	Name	Description	Format*	Range(Hex)
Inputs	SinIn	Measured sin signal	GLOBAL_Q	80000000-7FFFFFFF
	CosIn	Measured cos signal	GLOBAL_Q	80000000-7FFFFFFF
Outputs	OutputTheta	Motor electrical position angle	GLOBAL_Q	00000000-7FFFFFFF (0 – 360 degree)
	Speed	Motor speed in pu	GLOBAL_Q	80000000-7FFFFFFF
	SpeedRpm	Motor speed in rpm	Q0	00000000-7FFFFFFF
RESOLVER parameter	FilterGain[18]	Low-pass filter gains (FIR 17-order, Hamming windows)	GLOBAL_Q	80000000-7FFFFFFF
	SignalGain	Sin and cos gain	GLOBAL_Q	80000000-7FFFFFFF
	SpeedGain	Speed calculation gain	GLOBAL_Q	80000000-7FFFFFFF
	PolePairs	Number of pole pairs	Q0	00000000-7FFFFFFF
	BaseRpm	Base speed in rpm	Q0	00000000-7FFFFFFF
	K1	Constant for differentiator	Q21	00000000-7FFFFFFF
	CalibratedAngle	Offset between resolver's signal and phase-a winding	GLOBAL_Q	80000000-7FFFFFFF
Internal	FilteredSin	Filtered sin signal	GLOBAL_Q	80000000-7FFFFFFF
	FilteredCos	Filtered cos signal	GLOBAL_Q	80000000-7FFFFFFF
	DemodSin	Demodulated sin signal	GLOBAL_Q	80000000-7FFFFFFF
	DemodCos	Demodulated cos signal	GLOBAL_Q	80000000-7FFFFFFF
	AmplifiedSin	Amplified sin signal	GLOBAL_Q	80000000-7FFFFFFF
	AmplifiedCos	Amplified cos signal	GLOBAL_Q	80000000-7FFFFFFF
	FilterTheta	FilterTheta1 – FilterTheta2	GLOBAL_Q	80000000-7FFFFFFF
	FilterTheta1	Filtered atan(sin/cos)	GLOBAL_Q	80000000-7FFFFFFF
	FilterTheta2	Filtered atan(cos/sin)	GLOBAL_Q	80000000-7FFFFFFF
	ElecTheta	Electrical angle before comp.	GLOBAL_Q	80000000-7FFFFFFF
	MechTheta	Mechanical angle before comp.	GLOBAL_Q	80000000-7FFFFFFF
	RefTheta	Filtered mechanical angle before compensation	GLOBAL_Q	80000000-7FFFFFFF
	SignFlag	Sign change of FilterTheta	Q0	0 or 1
	SignalCounter	Sign change counter	Q0	0,1,2, or 3

*GLOBAL_Q valued between 1 and 30 is defined in the IQmathLib.h header file.

Special Constants and Data types

RESOLVER
The module definition is created as a data type. This makes it convenient to instance an interface to the position and speed calculation module based on resolver signals. To create multiple instances of the module simply declare variables of type RESOLVER.

RESOLVER_handle
User defined Data type of pointer to RESOLVER module

RESOLVER_DEFAULTS
Structure symbolic constant to initialize RESOLVER module. This provides the initial values to the terminal variables as well as method pointers.

Methods

void resolver_calc(RESOLVER_handle);

This definition implements one method viz., the position and speed calculation function based on resolver signals. The input argument to this function is the module handle.

Module Usage

Instantiation
The following example instances one RESOLVER objects
RESOLVER res1, res2;

Initialization
To Instance pre-initialized objects
RESOLVER res1 = RESOLVER_DEFAULTS;
RESOLVER res2 = RESOLVER_DEFAULTS;

Invoking the computation function
res1.calc(&res1);
res2.calc(&res2);

Example

The following pseudo code provides the information about the module usage.

```
main()
{
        res1.FilterGain[0] = parem1_1;          // Pass parameters to res1
                .                                       .
                .                                       .
                .                                       .
        res1.FilterGain[17] = parem1_17;        // Pass parameters to res1
        res1.SpeedGain = parem1_18;             // Pass parameters to res1
        res1.SignalGain = parem1_19;            // Pass parameters to res1
        res1.K1 = parem1_20;                    // Pass parameters to res1



        res2.FilterGain[0] = parem2_1;          // Pass parameters to res2
                .                                       .
                .                                       .
                .                                       .
        res2.FilterGain[17] = parem2_17;        // Pass parameters to res2
        res2.SpeedGain = parem2_18;             // Pass parameters to res2
        res2.SignalGain = parem2_19;            // Pass parameters to res2
        res2.K1 = parem2_20;                    // Pass parameters to res2
}
```

```
void interrupt periodic_interrupt_isr()
{
        res1.SinIn =  _IQ15toIQ(adc1_SinIn);     // Pass Q15 input to res1
        res1.CosIn =  _IQ15toIQ(adc1_CosIn);   // Pass Q15 input to res1

        res2.SinIn =  _IQ15toIQ(adc2_SinIn);     // Pass Q15 input to res2
        res2.CosIn =  _IQ15toIQ(adc2_CosIn);   // Pass Q15 input to res2

        res1.calc(&res1);                        // Call compute function for res1
        res2.calc(&res2);                        // Call compute function for res2

        speed1 = res1.Speed;                     // Access the outputs of res1
        position1 = res1.OutputTheta;            // Access the outputs of res1

        speed2 = res2.Speed;                     // Access the outputs of res2
        position2 = res2.OutputTheta;            // Access the outputs of res2

}
```

Technical Background

This software module consists of four functions explained as follows:

demodulator_calc()
This function is to demodulate the sin and cos input signals from the resolver. The excitation signal's peaks, which are the envelop of signals, are detected by finding the maximum value of absolute sin and cos input signals in the moving windows. The window length is the number of the sampling points per each period of excitation signal (typically, 2-10 kHz). For example, if the sin and cos signals from the resolver are sampled at a sampling rate of 20 KHz. And the frequency of excitation signal is 5 kHz. Then, the length of window would be 4. The absolute and demodulated sin signals can be seen in figure 1.

filter_calc()
The FIR 17-order low-pass filter (Hamming Windows) is primarily used for the demodulated signals performed by demodulator_calc() function. The objective of this low-pass filter is to filter out or smoothen the demodulated signals because the input signals would possibly contain the noises from the measurement. The filter gains are easily obtained by "Digital Filter Design" (FDA Tool) from Matlab/Simulink. The filtered sin signal can be seen in Figure 1.

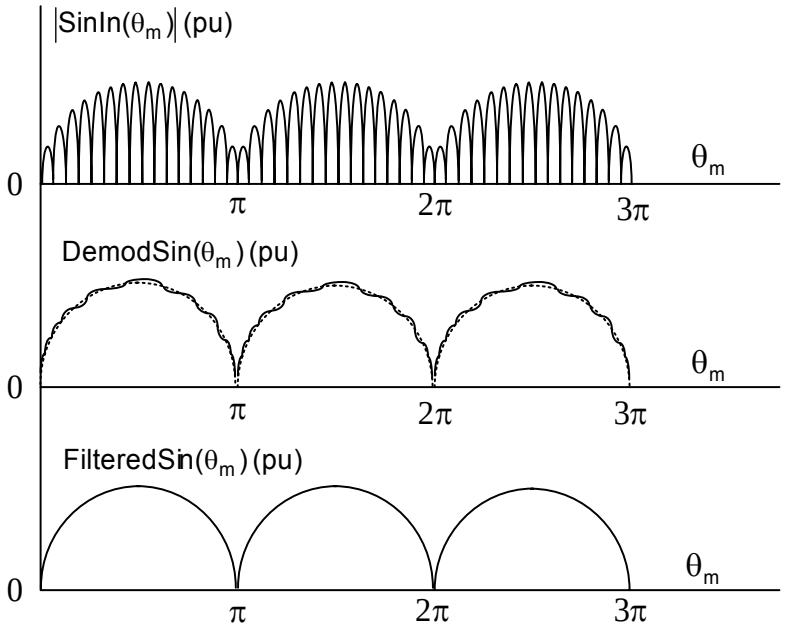


Figure 1: Absolute, demodulated, and filtered sin signals

position_speed_calc()
This function calculates the rotor position and motor speed. After the signals are filtered by the filter_calc() function, the mechanical angles are computed in the following ways:

$$\text{FilterTheta1} = \tan^{-1}(\text{FilteredSin}/\text{FilteredCos}) \quad (1)$$
$$\text{FilterTheta2} = \tan^{-1}(\text{FilteredCos}/\text{FilteredSin}) \quad (2)$$
$$\text{FilterTheta} = \text{FilterTheta1} - \text{FilterTheta2} \quad (3)$$

These mechanical angles can be shown in figure 2. As seen in this figure, the FilterTheta is the bipolar signal. The zero crossings of the FilterTheta is detected and counted by SignalCounter variable. One mechanical revolution has four zero-crossings, so this SignalCounter variable will be reset to zero at every fourth zero-crossing (see figure 2). The mechanical angle is computed in the following ways.

At every zero crossing, the MechTheta is set according to the SignalCounter value as

$$\text{MechTheta} = \text{SignalCounter}*0.25 \quad \text{pu.} \quad (4)$$

where SignalCounter = 0, 1, 2,or 3 and MechTheta is in per-unit (valued between 0 and 1).

Otherwise, the MechTheta will be computed as follows:

$$\text{MechTheta} = \text{MechTheta} + \text{SpeedGain}*\text{Speed} \quad \text{pu.} \quad (5)$$

where Speed is the computed motor speed (pu),
SpeedGain = BASE_MECHANICAL_FREQ*SAMPLING_PERIOD.

The motor speed (Speed) is simply calculated by differentiating the FilterTheta1 signal and then filtering the motor speed by a simple 1-order low-pass filter.

In practice, at every zero crossing the computed MechTheta in (5) may not be exactly the MechTheta set in (4), thus the MechTheta is averaged three values; that are MechTheta before zero-crossing, at zero-crossing, and after zero-crossing. Then, the MechTheta is filtered by a 1-order low-pass filter to smoothen the signal. The RefTheta variable is defined for the filtered MechTheta.

angle_comp_calc()
This function is to compensate the delays in the rotor position's calculations. Firstly, the delays due to the FIR 17-order and 1-order low-pass filters used are handled. These filters provide the linear phase delay which can be easily compensated by looking at frequency. As frequency increases, the delay is also increased. The angle delay of FIR 17-order low-pass filter can also be obtained by "Digital Filter Design" (FDA Tool) from Matlab/Simulink.

Secondly, since MechTheta is calculated based on FilterTheta signal rather than FilteredSin signal, thus the compensation of mechanical angle 45^{o} ($\pi/4$ rad) should be added (see MechTheta in figure 2).

Thirdly, the position compensation between the resolver's sin winding and phase-a motor winding should also be taken in the consideration. Because for the variable transformation (abc to dq axis) the zero angle is defined at phase-a axis.

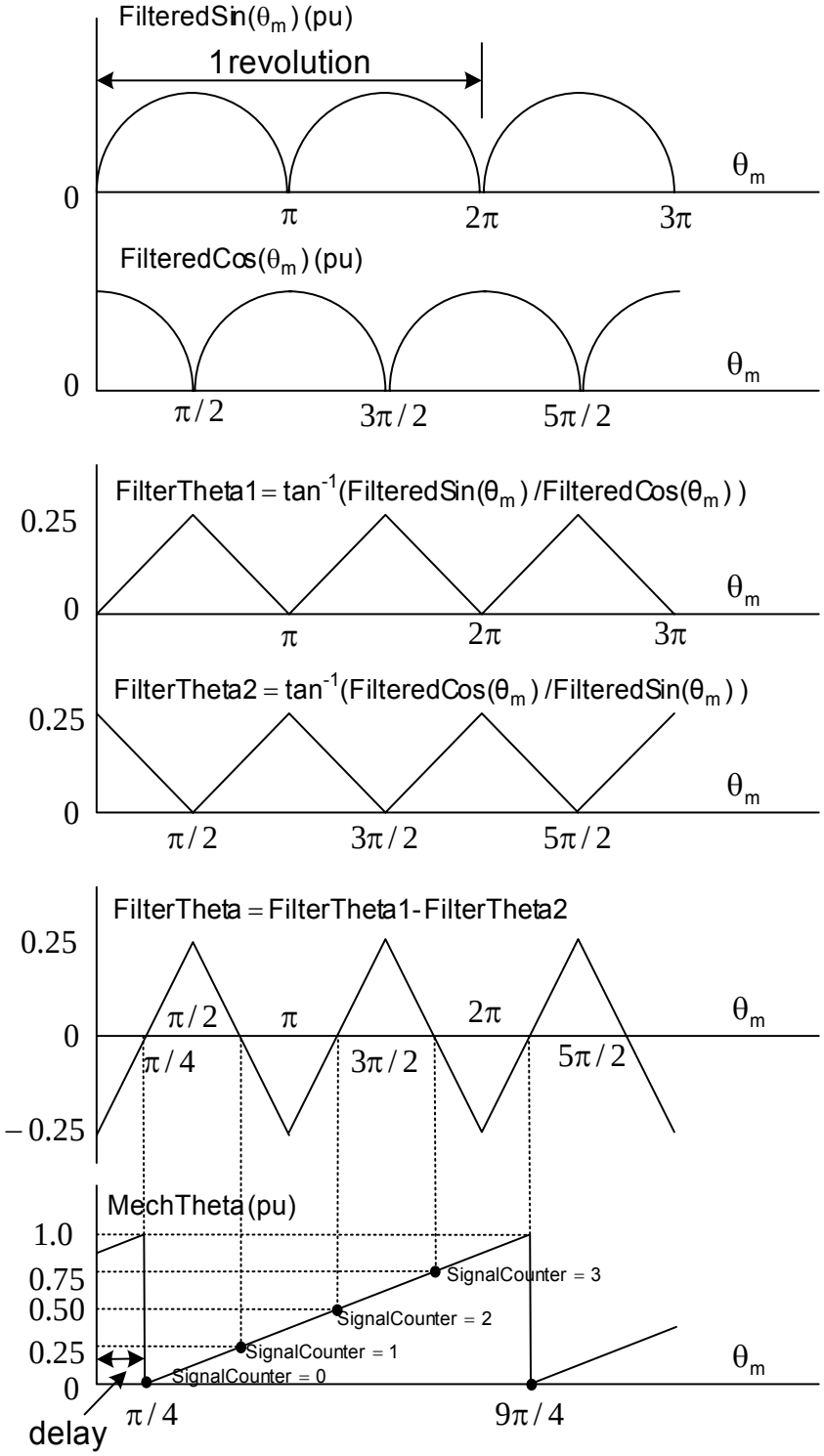


Figure 2: FilteredSin, FilteredCos, FilterTheta1, FilterTheta2, FilterTheta, and MechTheta