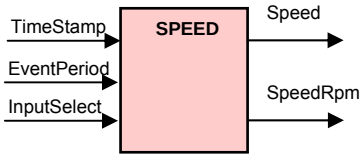
SPEED	*Speed Calculator Based on Period Measurement*

Description

This module calculates the motor speed based on a signal's period measurement. Such a signal, for which the period is measured, can be the periodic output pulses from a motor speed sensor.



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: speed_pr.c, speed_pr.h

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

Item	C version	Comments
Code Size□ (x281x/x280x)	45/45 words	
Data RAM	0 words•	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

• Each pre-initialized "_iq" SPEED_MEAS_CAP structure consumes 19 words in the data memory

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

C Interface

Object Definition

The structure of SPEED_MEAS_CAP object is defined by following structure definition

```
typedef struct {  Uint32 NewTimeStamp; // Variable: New 'Timestamp' for a capture event (Q0)
                  Uint32 OldTimeStamp;  // Variable : Old 'Timestamp' for a capture event (Q0)
                  Uint32 TimeStamp;     // Input: Current 'Timestamp' for a capture event (Q0)
                  Uint32 SpeedScaler;   // Parameter:  Scaler converting 1/N cycles (Q0)
                   int32 EventPeriod;   // Input/Variable :  Event Period (Q0)
                   int16 InputSelect;   // Input : Input selection (1 or 0)
                  _iq Speed;            // Output:  speed in per-unit
                  Uint32 BaseRpm;       // Parameter : Scaler converting to rpm (Q0)
                  int32 SpeedRpm;       // Output : speed in r.p.m. (Q0)
                  void (*calc)();       // Pointer to the calculation function
                } SPEED_MEAS_CAP;   // Data type created

typedef SPEED_MEAS_CAP * SPEED_MEAS_CAP_handle;
```

Item	Name	Description	Format*	Range(Hex)
Inputs	TimeStamp	Current 'Timestamp' for a capture event	Q0	80000000-7FFFFFFF
	InputSelect	TimeStamp (InputSelect=0) and EventPeriod (InputSelect=1)	Q0	0 or 1
	EventPeriod	Event period between time stamp	Q0	80000000-7FFFFFFF
Outputs	Speed	Computed speed in per-unit	GLOBAL_Q	80000000-7FFFFFFF
	SpeedRpm	Speed in rpm	Q0	80000000-7FFFFFFF
SPEED_CAP parameter	SpeedScaler	SpeedScaler = 60*CLK_freq/(128*N*rmp_max), N = number of sprocket teeth	Q0	80000000-7FFFFFFF
	BaseRpm	rmp_max = 120fb/p	Q0	80000000-7FFFFFFF
Internal	NewTimeStamp	New 'Timestamp' for a capture event	Q0	80000000-7FFFFFFF
	OldTimeStamp	Old 'Timestamp' for a capture event	Q0	80000000-7FFFFFFF

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

Special Constants and Data types

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

SPEED_MEAS_CAP_handle
User defined Data type of pointer to SPEED_MEAS_CAP module

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

Methods

void Speed_calc(SPEED_MEAS_CAP_handle);

This definition implements one method viz., the speed calculation based on period computation function. The input argument to this function is the module handle.

Module Usage

Instantiation
The following example instances two SPEED_MEAS_CAP objects
SPEED_MEAS_CAP speed1, speed2;

Initialization
To Instance pre-initialized objects
SPEED_MEAS_CAP speed1 = SPEED_MEAS_CAP_DEFAULTS;
SPEED_MEAS_CAP speed2 = SPEED_MEAS_CAP_DEFAULTS;

Invoking the computation function
speed1.calc(&speed1);
speed2.calc(&speed2);

Example

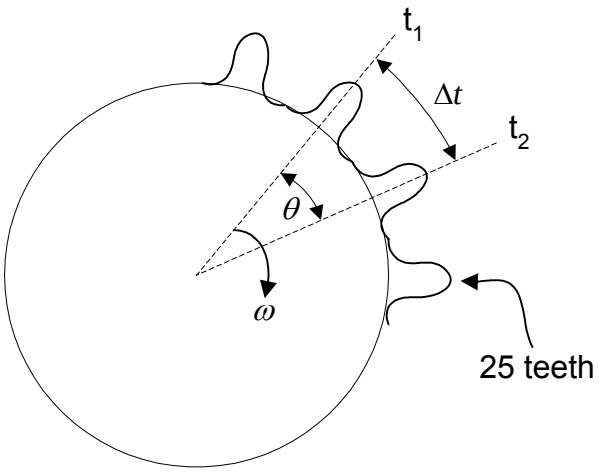
The following pseudo code provides the information about the module usage.

```
main()
{

}

void interrupt periodic_interrupt_isr()
{
        speed1.TimeStamp = TimeStamp1;      // Pass inputs to speed1
        speed2.TimeStamp = TimeStamp2;      // Pass inputs to speed2

        speed1.calc(&speed1);               // Call compute function for speed1
        speed2.calc(&speed2);               // Call compute function for speed2

        measured_spd1 = speed1.Speed;       // Access the outputs of speed1
        measured_spd2 = speed2.Speed;       // Access the outputs of speed2
}
```

Technical Background

A low cost shaft sprocket with n teeth and a Hall effect gear tooth sensor is used to measure the motor speed. Fig. 1 shows the physical details associated with the sprocket. The Hall effect sensor outputs a square wave pulse every time a tooth rotates within its proximity. The resultant pulse rate is n pulses per revolution. The Hall effect sensor output is fed directly to the 281x Capture input pin. The capture unit will capture (the value of it's base timer counter) on either the rising or the falling edges (whichever is specified) of the Hall effect sensor output. The captured value is passed to this s/w module through the variable called *TimeStamp*.

In this module, every time a new input *TimeStamp* becomes available it is compared with the previous *TimeStamp*. Thus, the tooth-to-tooth period (t_2-t_1) value is calculated. In order to reduce jitter or period fluctuation, an average of the most recent n period measurements can be performed each time a new pulse is detected.



$$\Delta t = t_2 - t_1 \quad \text{sec}$$

$$\theta = \frac{360}{25} = 14.4^0$$

$$\omega = \frac{2\pi}{\sum_{n=1}^{25} \Delta t_n} \quad \text{rad/sec}$$

Fig. 1 Speed measurement with a sprocket

From the two consecutive *TimeStamp* values the difference between the captured values are calculated as,

Δ = TimeStamp(new) – TimeStamp(old)

Then the time period in sec is given by,

$$\Delta t = t_2 - t_1 = \mathrm{K_P} \times \mathrm{T}_{CLK} \times \Delta$$

where,

K_P = Prescaler value for the Capture unit time base

T_{CLK} = CPU clock period in sec

From Fig. 1, the angle θ in radian is given by,

$$\theta = \frac{2\pi}{n}$$

where, n = number of teeth in the sprocket, i.e., the number of pulses per revolution.

Then the speed ω in radian/sec and the normalized speed ω_N are calculated as,

$$\omega = \frac{\theta}{\Delta t} = \frac{2\pi}{n\Delta t} = \frac{2\pi}{n \times \mathrm{K}_P \times \mathrm{T}_{CLK} \times \Delta}$$

$$\Rightarrow \omega_N = \frac{\omega}{\omega_{\max}} = \frac{\omega}{2\pi\left(\frac{1}{n \times \mathrm{K}_P \times \mathrm{T}_{CLK}}\right)} = \frac{1}{\Delta}$$

Where, ω_{max} is the maximum value of ω which occurs when Δ=1. Therefore,

$$\omega_{\max} = \frac{2\pi}{nK_P T_{CLK}}$$

For, n=25, K_P=32 and T_{CLK}=50x10^{-9} sec (20MHz CPU clock), the normalized speed ω_N is given by,

$$\omega_N = \frac{\omega}{2\pi(25000)} = \frac{1}{\Delta}$$

The system parameters chosen above allows maximum speed measurement of 1500,000 rpm. Now, in any practical implementation the maximum motor speed will be significantly lower than this maximum measurable speed. So, for example, if the motor used has a maximum operating speed of 23000 rpm, then the calculated speed can be expressed as a normalized value with a base value of normalization of at least 23000 RPM. If we choose this base value of normalization as 23438 rpm, then the corresponding base value of normalization, in rad/sec, is,

$$\omega_{\max 1} = \frac{23438 \times 2\pi}{60} \approx 2\pi(390)$$

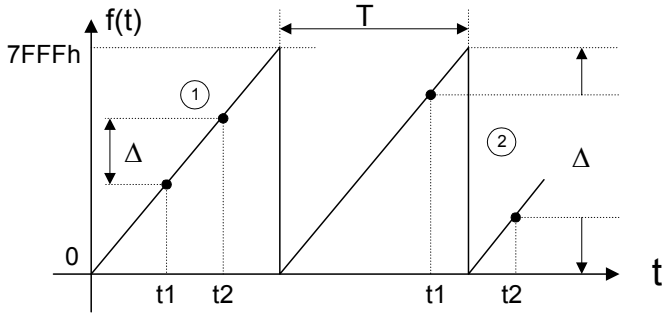
Therefore, the scaled normalized speed in per-unit is calculated as,

$$\omega_{N1} = \frac{\omega}{2\pi(390)} \approx \frac{64}{\Delta} = 64 \times \omega_N = \text{SpeedScaler} \times \omega_N$$

This shows that in this case the scaling factor is 64. The speed, in rpm, is calculated as,

$$N_1 = 23438 \times \omega_{N1} = 23438 \times \frac{64}{\Delta} = \text{BaseRpm} \times \omega_{N1}$$

The capture unit allows accurate time measurement (in multiples of clock cycles and defined by a prescaler selection) between events. In this case the events are selected to be the rising edge of the incoming pulse train. What we are interested in is the delta time between events and hence for this implementation Timer 1 is allowed to free run with a prescale of 32 (1.6uS resolution for 20MHz CPU clock) and the delta time Δ, in scaled clock counts, is calculated as shown in Fig. 2.



Case ① $f(t_2) \geq f(t_1)$

$\Delta = f(t_2) - f(t_1)$

Case ② $f(t_2) \leq f(t_1)$

$\Delta = 1 + f(t_2) - f(t_1)$

Note: only true if $t_2 - t_1 < T$

Fig. 2 Calculation of speed

In Fig. 2, the vertical axis *f(t)* represents the value of the Timer counter which is running in continuous up count mode and resetting when the period register = 7FFFh. Note that two cases need to be accounted for: the simple case where the Timer has not wrapped around and where it has wrapped around. By keeping the current and previous capture values it is easy to test for each of these cases.

Once a "robust" period measurement is extracted from the averaging algorithm, the speed is calculated using the appropriate equations explained before

Finally, for a base speed in rpm (BaseRpm), the *SpeedScaler* is basically $1/\omega_N$ (with $\omega = \omega_{max1}$) for the number of teeth in the sprocket, prescaler value for the capture unit time base, and CPU clock period (sec) is therefore computed as follows:

$$\text{SpeedScaler} = \frac{60}{T_{CLK} \times K_p \times n \times \text{BaseRpm}}$$