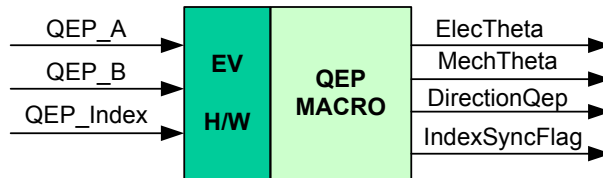


Description

This module determines the rotor position and generates a direction (of rotation) signal from the shaft position encoder pulses.

**Availability**

This 16-bit module is available in one interface format:

- 1) The C interface version

Module Properties

Type: Target Dependent, Application Independent

Target Devices: x2833x Floating Point device

C Version File Names: f2833xqep.h (for x2833x)

IQmath library files for C: N/A

C Interface

C Interface

Object Definition

The structure of QEP object is defined by following structure definition

x2833x series

```
typedef struct { int32 ElecTheta;          // Output: Motor Electrical angle (Q24)
                int32 MechTheta;          // Output: Motor Mechanical Angle (Q24)
                Uint16 DirectionQep;       // Output: Motor rotation direction (Q0)
                Uint16 QepPeriod;          // Output: Capture period of QEP signal (Q0)
                Uint32 QepCountIndex;      // Variable: Encoder counter index (Q0)
                int32 RawTheta;            // Variable: Raw angle from EQEP position counter (Q0)
                Uint32 MechScaler;          // Parameter: 0.9999/total count (Q30)
                Uint16 LineEncoder;        // Parameter: Number of line encoder (Q0)
                Uint16 PolePairs;          // Parameter: Number of pole pairs (Q0)
                int32 CalibratedAngle;      // Parameter: Raw offset between encoder and ph-a (Q0)
                Uint16 IndexSyncFlag;      // Output: Index sync status (Q0)
            } QEP;

typedef QEP *QEP_handle;
```

Item	Name	Description	Format	Range(Hex)
Inputs	QEP_A	QEP_A signal applied to QEP1-A	N/A	0-3.3 v
	QEP_B	QEP_A signal applied to QEP1-B	N/A	0-3.3 v
	QEP_Index	QEP_Index signal applied to QEP1-I	N/A	0-3.3 v
Outputs	ElecTheta	Motor Electrical angle	Q24 (2833x)	00000000- 7FFFFFFF (0 – 360 degree)
	MechTheta	Motor Mechanical Angle	Q24 (2833x)	00000000- 7FFFFFFF (0 – 360 degree)
	DirectionQep	Motor rotation direction	Q0	0 or 1
	IndexSyncFlag	Index sync status	Q0	0 or F0
QEP parameter	MechScaler*	MechScaler = 1/total count, total count = 4*no_lines_encoder	Q30	00000000- 7FFFFFFF
	PolePairs	Number of pole pairs	Q0	1,2,3,...
	CalibratedAngle	Raw offset between encoder and phase a	Q0	8000-7FFF
Internal	QepCountIndex	Encoder counter index	Q0	8000-7FFF
	RawTheta	Raw angle from Timer 2	Q0	8000-7FFF

*MechScaler in Q30 is defined by a 32-bit word-length

Special Constants and Data types

QEP

The module definition is created as a data type. This makes it convenient to instance an interface to the QEP driver. To create multiple instances of the module simply declare variables of type QEP.

QEP_handle

User defined Data type of pointer to QEP module

QEP_DEFAULTS

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

Methods

```
QEP_INIT_MACRO(QEP *);
QEP_MACRO(QEP *);
```

This default definition of the object implements three methods – the initialization, the runtime compute, and interrupt macros for QEP generation. This is implemented by means of a macro pointer, and the initializer sets this to QEP_INIT_MACRO, and QEP_MACRO macros for x2833x. The argument to this macro is the address of the QEP object.

Module Usage

Instantiation

The following example instances one QEP object
QEP qep1;

Initialization

To Instance pre-initialized objects
QEP qep1 = QEP_DEFAULTS;

Invoking the computation macro

QEP_INIT_MACRO (qep1);
QEP_MACRO (qep1);

The index event handler resets the QEP counter, and synchronizes the software/hardware counters to the index pulse. Also it sets the QEP IndexSyncFlag variable to reflect that an index sync has occurred.

The index handler is invoked in an interrupt service routine. Of course the system framework must ensure that the index signal is connected to the correct pin and the appropriate interrupt is enabled and so on.

Example

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

```
main()
{
    QEP_INIT_MACRO(qep1);           // Call init macro for qep1
}

void interrupt periodic_interrupt_isr()
{
    QEP_MACRO(qep1);               // Call compute macro for qep1
}
```

Technical Background

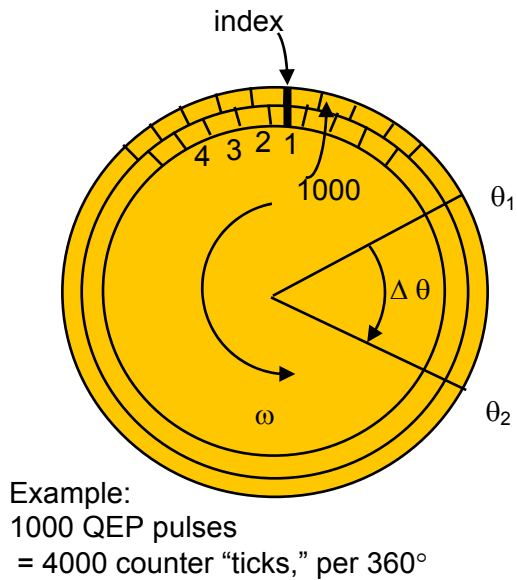


Figure 1. Speed sensor disk

Figure 1 shows a typical speed sensor disk mounted on a motor shaft for motor speed, position and direction sensing applications. When the motor rotates, the sensor generates two quadrature pulses and one index pulse. These signals are shown in Figure 2 as QEP_A, QEP_B and QEP_index.

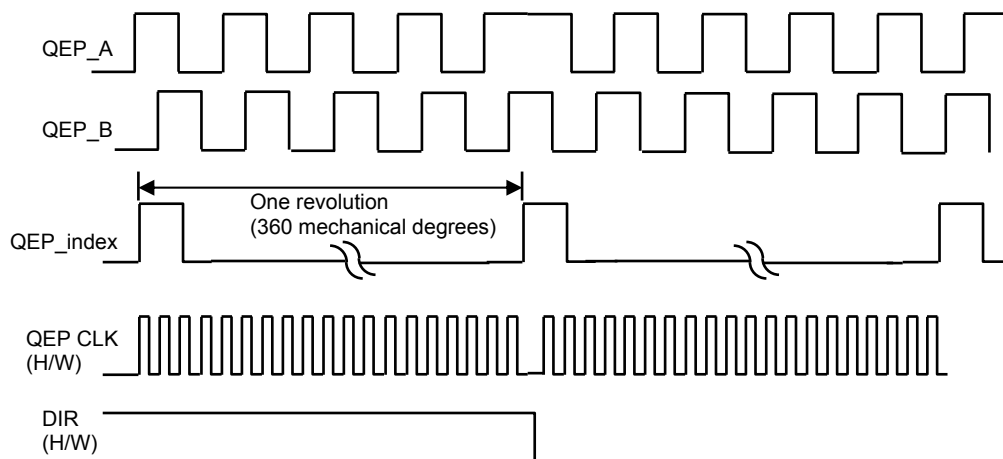


Figure 2. Quadrature encoder pulses, decoded timer clock and direction signal.

For the x280x and 2833x devices, QEP_A and QEP_B signals are applied to the EQEP1A and EQEP1B pins, respectively. The QEP_index signal is applied to the EQEP1I pin. And the position counter is obtained by QPOSCNT register.

Now the number of pulses generated by the speed sensor is proportional to the angular displacement of the motor shaft. In Figure 1, a complete 360° rotation of motor shaft generates 1000 pulses of each of the signals QEP_A and QEP_B. The QEP circuit in 281x counts both edges of the two QEP pulses. Therefore, the frequency of the counter clock, QEP_CLK, is four times that of each input sequence. This means, for 1000 pulses for each of QEP_A and QEP_B, the number of counter clock cycles will be 4000. Since the counter value is proportional to the number of QEP pulses, therefore, it is also proportional to the angular displacement of the motor shaft.

For the x280x and 2833x devices, the QDF bit in QEOSTS register is used to check the rotational direction. The index pulse resets the timer counter T2CNT and sets the index synchronization flag *IndexSyncFlag* to 00F0. Thus the counter T2CNT gets reset and starts counting the QEP_CLK pulses every time a QEP_index high pulse is generated.

To determine the rotor position at any instant of time, the counter value(T2CNT) is read and saved in the variable *RawTheta*. This value indicates the clock pulse count at that instant of time. Therefore, *RawTheta* is a measure of the rotor mechanical displacement in terms of the number of clock pulses. From this value of *RawTheta*, the corresponding per unit mechanical displacement of the rotor, *MechTheta*, is calculated as follows:

Since the maximum number of clock pulses in one revolution is 4000, i.e., maximum count value is 4000, then a coefficient, *MechScaler*, can be defined as,

$$\begin{aligned} \text{MechScaler} \times 4000 &= 360^\circ \text{ mechanical} = 1 \text{ per unit(pu) mechanical displacement} \\ \Rightarrow \text{MechScaler} &= (1/4000) \text{ pu mech displacement / count} \\ &= 16777 \text{ pu mech displacement / count (in Q30)} \end{aligned}$$

Then, the pu mechanical displacement, for a count value of *RawTheta*, is given by,

$$\text{MechTheta} = \text{MechScaler} \times \text{RawTheta}$$

If the number of pole pair is *polepairs*, then the pu electrical displacement is given by,

$$\text{ElecTheta} = \text{PolePairs} \times \text{MechTheta}$$