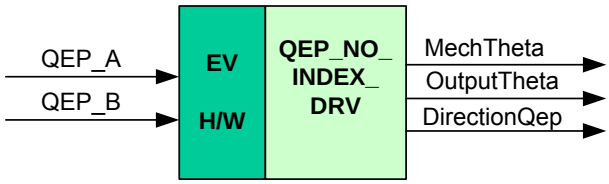
QEP_NO_INDEX_DRV	*Quadrature Encoder Without Index Pulse Driver*

Description

This module determines the motor and output positions. In this case, the motor is coupled with a gearbox. For example, for a gearbox ratio of 256:1, that means that motor turns 256 revolutions to get 1 output revolution. This module also generates a direction (of rotation) signal from the shaft position encoder A and B pulses without the index pulse.



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: x281x or x280x

C Version File Names:

f281xqep_no_index.c, f281xqep_no_index.h (for x281x)
f280xqep_no_index.c, f280xqep_no_index.h (for x280x)

IQmath library files for C: N/A

Item	C version	Comments
Code Size□ (x281x/x280x)	126/160 words	
Data RAM	0 words•	
xDAIS ready	No	
XDAIS component	No	IALG layer not implemented
Multiple instances	Yes	
Reentrancy	Yes	

• Each pre-initialized QEP structure consumes 20 words in the data memory (for x281x) and 22 words in the data memory (for x280x)

□ Code size mentioned here is the size of the ***init()***, ***calc()***, and ***isr()*** functions (for x281x) and the size of the ***init()*** and ***calc()*** functions (for x280x)

C Interface

Object Definition

The structure of QEP object is defined by following structure definition

x281x series

```
typedef struct { int16 MechTheta;            // Output: Motor Mechanical Angle (Q15)
                 int16 OutputTheta;          // Output: Output Mechanical Angle (Q15)
                 Uint16 DirectionQep;        // Output: Motor rotation direction (Q0)
                 Uint16 QepCountIndex;   // Parameter: Encoder cnt index = 4*pulse_per_rev (Q0)
                 Uint16 RawTheta;           // Variable: Raw angle from Timer 2 (Q0)
                 Uint32 OutputRawTheta;  // Variable: Raw angle for output position (Q0)
                 Uint32 MechScaler;         // Parameter: 0.9999/total count for motor (Q30)
                 Uint32 OutputMechScaler;  // Parameter: 0.9999/total count for output (Q30)
                 Uint16 LineEncoder;          // Parameter: Number of line encoder (Q0)
                 Uint16 PreScaler;            // Parameter: ratio of motor rev. to output rev. (Q0)
                 Uint16 Counter;              // Variable: Counter for revolution of motor (Q0)
                 void (*init)();              // Pointer to the init function
                 void (*calc)();              // Pointer to the calc function
                 void (*isr)();               // Pointer to the isr function
               }  QEP;
```

x280x series

```
typedef struct { int32 MechTheta;            // Output: Motor Mechanical Angle (Q24)
                 int32 OutputTheta;          // Output: Output Mechanical Angle (Q24)
                 Uint16 DirectionQep;        // Output: Motor rotation direction (Q0)
                 Uint16 QepPeriod;           // Output: Capture period of Qep signal (Q0)
                 Uint16 QepCountIndex;    // Variable: Encoder cnt index = 4*pulse_per_rev (Q0)
                 Uint32 RawTheta;           // Variable: Raw angle from EQep Postiion counter (Q0)
                 Uint32 OutputRawTheta;  // Variable: Raw angle for output position (Q0)
                 Uint32 MechScaler;         // Parameter: 0.9999/total count for motor (Q30)
                 Uint32 OutputMechScaler; // Parameter: 0.9999/total count for output (Q30)
                 Uint16 LineEncoder;         // Parameter: Number of line encoder (Q0)
                 Uint16 PreScaler;           // Parameter: ratio of motor rev. to output rev. (Q0)
                 Uint16 Counter;             // Variable: Counter for revolution of motor (Q0)
                 void (*init)();             // Pointer to the init function
                 void (*calc)();             // Pointer to the calc function
               }  QEP;


typedef QEP *QEP_handle;
```

Item	Name	Description	Format	Range(Hex)
Inputs	QEP_A	QEP_A signal applied to CAP1	N/A	0-3.3 v
	QEP_B	QEP_A signal applied to CAP2	N/A	0-3.3 v
Outputs	MechTheta	Motor Mechanical angle	Q15 (281x) Q24 (280x)	0000-7FFF 00000000-7FFFFFFF (0 – 360 degree)
	OutputTheta	Output Mechanical Angle	Q15 (281x) Q24 (280x)	0000-7FFF 00000000-7FFFFFFF (0 – 360 degree)
	DirectionQep	Motor rotation direction	Q0	0 or 1
QEP parameter	MechScaler*	MechScaler = 1/total count, total count = 4*no_lines_encoder	Q30	00000000-7FFFFFFF
	OutputMechScaler*	OutputMechScaler = 1/total output count, total out put count = 4*no_lines_encoder*256	Q30	00000000-7FFFFFFF
	LineEncoder	Number of line encoder	Q0	00000000-7FFFFFFF
	PreScaler	A ratio of motor revolution to output revolution	Q0	0000-7FFF
Internal	Counter	Counter for revolution of motor	Q0	0000-7FFF

*MechScaler and OutputMechScaler 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

void F281X_EV1_QEP_NO_INDEX_Init(QEP *);
void F281X_EV1_QEP_NO_INDEX_Calc(QEP *);
void F281X_EV1_QEP_NO_INDEX_Isr(QEP *);

void F280X_QEP_NO_INDEX_Init(QEP *);
void F280X_QEP_NO_INDEX_Calc(QEP *);

This default definition of the object implements three methods – the initialization, the runtime compute, and interrupt functions for QEP generation. This is implemented by means of a function pointer, and the initializer sets this to F281X_EV1_QEP_NO_INDEX_Init, F281X_EV1_QEP_NO_INDEX_Calc, and F281X_EV1_ QEP_NO_INDEX_Isr functions for x281x or F280X_QEP_NO_INDEX_Init and F280X_QEP_NO_INDEX_Calc functions for x280x. The argument to this function 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 function
qep1.init(&qep1);
qep1.calc(&qep1);
qep1.isr(&qep1);

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

```
main()
{

        qep1.init(&qep1);                   // Call init function for qep1

}

void interrupt periodic_interrupt_isr()
{

        qep1.calc(&qep1);                   // Call compute function for qep1

}

void interrupt T2P_interrupt_isr()
{

        qep1.isr(&qep1);                    // Call isr function for qep1

}
```

Technical Background

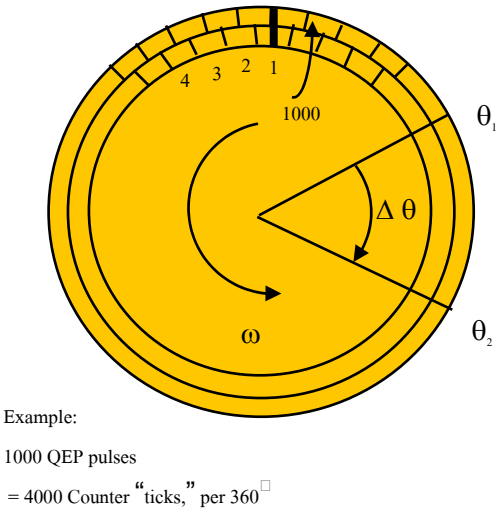


Figure 1. Speed sensor disk

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 without the index pulse. These signals are shown in Figure 2 as QEP_A and QEP_B.

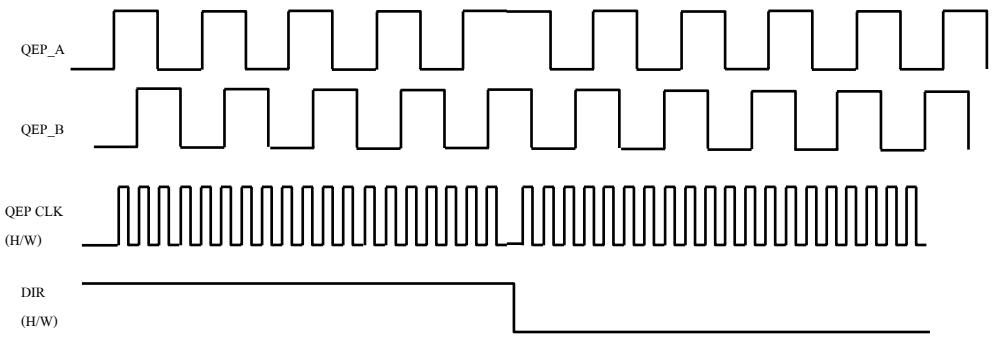


Figure 2. Quadrature encoder pulses, decoded timer clock and direction signal.

These signals are applied to 281x CAP/QEP interface circuit to determine the motor speed, position and direction of rotation. QEP_A and QEP_B signals are applied to the QEP1 and QEP2 pins of 281x device respectively. The QEP interface circuit in 281x, when enabled (CAPCONx[13,14]), count these QEP pulses and generates two signals internal to the device. These two signals are shown in Figure 2 as QEP_CLK and DIR. QEP_CLK signal is used as the clock input to GP Timer2. DIR signal controls the GP Timer2 counting direction.

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.

The counting direction of GP Timer2 is reflected by the status bit, BIT14, in GPTCON register. Therefore, in the s/w, BIT14 of GPTCON is checked to determine the direction of rotation of the motor.

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 motor 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 motor revolution is 4000 (ENCODER_MAX=4000), i.e., maximum count value is 4000, then a coefficient, *MechScaler*, can be defined as,

$$\begin{aligned}&\text{MechScaler} \times 4000 = 360^0 \,\text{mechanical} = 1 \,\text{per unit(pu) mechanical displacement} \\ &\Rightarrow \text{MechScaler} = (1/4000)\,\text{pu mech displacement / count} \\ &= 268435\text{pu mech displacement / count(in Q30)}\end{aligned}$$

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

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

In addition, an T2P interrupt is configured with a period of ENCODER_MAX = 4000 for counting the motor revolution. The objective is to compute the output counts. In this case, the motor turns 256 revolutions to get 1 output revolution through a gearbox attached with the motor.

Similarly, the *OutputMechScaler* can be defined as,

$$\text{OutputMechScaler} \times 4000 \times 256 = 360^0 \text{ output mechanical}$$
$$= 1 \text{ per unit (pu) output mechanical displacement}$$
$$\Rightarrow \text{OutputMechScaler} = \left(1/(256 \times 4000)\right) \text{pu output mech displacement / count}$$
$$= 1049 \text{pu output mech displacement / count (in Q30)}$$

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

$$\text{OutputRawTheta} = \text{Counter} \times \text{ENCODER\_MAX} + \text{RawTheta}$$

Finally, the output position is computed as

$$\text{OutputTheta} = \text{OutputMechScaler} \times \text{OutputRawTheta}$$