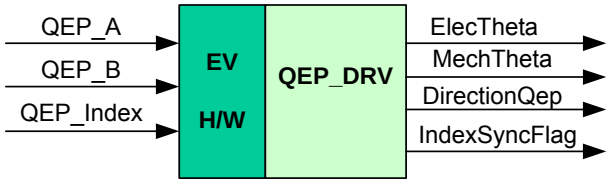
| QEP_THETA_DRV | *Quadrature Encoder Pulse Driver* |
|---|---|

**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:** x281x or x280x

**C Version File Names:** f281xqep.c, f281xqep.h (for x281x)
f280xqep.c, f280xqep.h (for x280x)

**IQmath library files for C:** N/A

| Item | C version | Comments |
|---|---|---|
| Code Size□ (x281x/x280x) | 88/150 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 17 words in the data memory (for x281x) and 21 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 ElecTheta;          // Output: Motor Electrical angle (Q15)
                 int16 MechTheta;          // Output: Motor Mechanical Angle (Q15)
                 Uint16 DirectionQep;      // Output: Motor rotation direction (Q0)
                 Uint16 QepCountIndex;     // Variable: Encoder counter index (Q0)
                 Uint16 RawTheta;          // Variable: Raw angle from Timer 2 (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)
                 int16 CalibratedAngle;    // Parameter: Raw offset between encoder and ph-a (Q0)
                 Uint16 IndexSyncFlag;     // Output: Index sync status (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 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)
                 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 |
| | QEP_Index | QEP_Index signal applied to CAP3 | N/A | 0-3.3 v |
| **Outputs** | ElecTheta | Motor Electrical angle | Q15 (281x)<br>Q24 (280x) | 0000-7FFF<br>00000000-7FFFFFFF<br>(0 – 360 degree) |
| | MechTheta | Motor Mechanical Angle | Q15 (281x)<br>Q24 (280x) | 0000-7FFF<br>00000000-7FFFFFFF<br>(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,<br>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

**void F281X_EV1_QEP_Init(QEP *);**
**void F281X_EV1_QEP_Calc(QEP *);**
**void F281X_EV1_QEP_Isr(QEP *);**

**void F280X_QEP_Init(QEP *);**
**void F280X_QEP_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_Init, F281X_EV1_QEP_Calc, and F281X_EV1_QEP_Isr functions for x281x or F280X_QEP_Init, and F280X_QEP_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);

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()
{

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

}

void interrupt periodic_interrupt_isr()
{

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

}


void interrupt cap3_interrupt_isr()
{

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

}
```

Notice that this example is for x281x only. For x280x devices, there is no capture 3 interrupt.

**Technical Background**

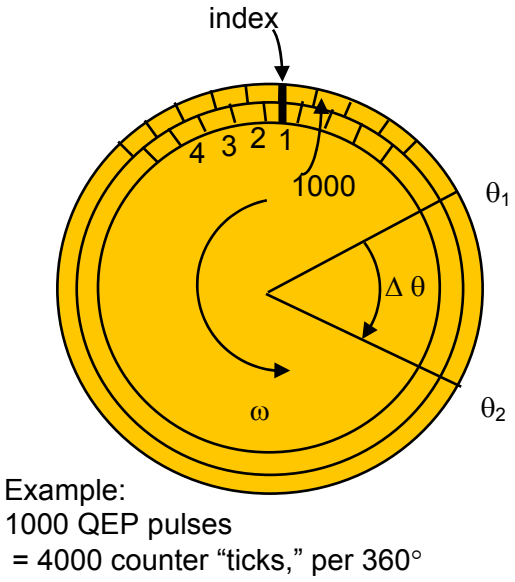



Figure 1. Speed sensor disk

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.

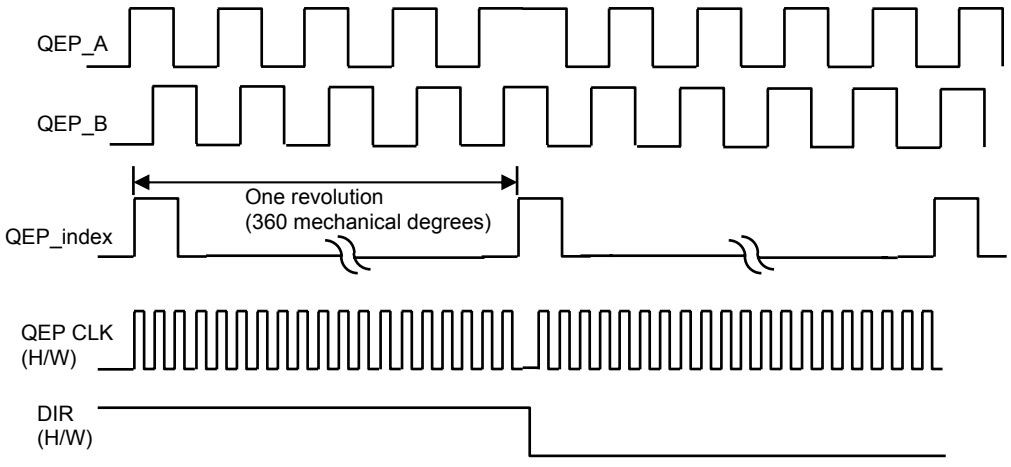



Figure 2. Quadrature encoder pulses, decoded timer clock and direction signal.

These signals are applied to x281x 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 x281x device, respectively. The QEP_index signal is applied to the CAP3 pin. 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 Timer 2 counting direction. For the x280x 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.

In the x281x devices, 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. For the x280x devices, the QDF bit in QEOSTS register is used to check the rotational direction.

The capture module (CAP3) is configured to generate an interrupt on every rising edge of the QEP_index signal. In the corresponding CAP3 interrupt routine the function F281X_EV1_QEP_Isr() is called. This function 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,

$$\text{MechScaler} \times 4000 = 360^0 \,\text{mechanical} = 1 \,\text{per unit(pu) mechanical displacement}$$

$$\Rightarrow \text{MechScaler} = (1/4000)\,\text{pu mech displacement / count}$$

$$= 16777 \,\text{pu mech displacement / count(in Q30)}$$

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,

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