



Rotary Encoder for 16LX-Series

© Fujitsu Microelectronic Europe GmbH, Microcontroller Application Group

Summary

This Application Note gives some ideas and basic information, how to connect a rotary-encoder to Fujitsu-16LX-microcontrollers.

The hardware is based on a 2-bit quadrature encoder, series 62A from Grayhill

(www.grayhill.com).

The software-examples are for a MB90540series, but can be used with minor changes for all other 16bit controllers, too.

History

24.01.2002	HWe	V1.0	New version
11.04.2002	HWe	V1.1	minor changes
25.04.2002	HWe	V1.2	Pin-number INT0 (P90=Pin69), INT1 (P91=Pin70) corrected

Table of Contents:

SUMMARY	1
1. OUTPUT SIGNAL OF THE ENCODER.....	3
2. SOFTWARE: USING “EXTERNAL INTERRUPTS”	4
3. SOFTWARE: USING THE “INPUT-CAPTURE-UNIT”.....	7
4. MAXIMUM FREQUENCY CALCULATION	11
5. CONCLUSION.....	11
6. APPENDIX A: SAMPLE PROJECT: ROT_ENC_EXTIRQ.....	12
7. APPENDIX B: SAMPLE PROJECT: ROT_ENC_ICU.....	15

Warranty and Disclaimer

To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH restricts its warranties and its liability for **all products delivered free of charge** (eg. software include or header files, application examples, target boards, evaluation boards, engineering samples of IC's etc.), its performance and any consequential damages, on the use of the Product in accordance with (i) the terms of the License Agreement and the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials. In addition, to the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH disclaims all warranties and liabilities for the performance of the Product and any consequential damages in cases of unauthorised decompiling and/or reverse engineering and/or disassembling. **Note, all these products are intended and must only be used in an evaluation laboratory environment.**

1. Fujitsu Microelectronics Europe GmbH warrants that the Product will perform substantially in accordance with the accompanying written materials for a period of 90 days from the date of receipt by the customer. Concerning the hardware components of the Product, Fujitsu Microelectronics Europe GmbH warrants that the Product will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.
2. Should a Product turn out to be defect, Fujitsu Microelectronics Europe GmbH's entire liability and the customer's exclusive remedy shall be, at Fujitsu Microelectronics Europe GmbH's sole discretion, either return of the purchase price and the license fee, or replacement of the Product or parts thereof, if the Product is returned to Fujitsu Microelectronics Europe GmbH in original packing and without further defects resulting from the customer's use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to Fujitsu Microelectronics Europe GmbH, or abuse or misapplication attributable to the customer or any other third party not relating to Fujitsu Microelectronics Europe GmbH.
3. To the maximum extent permitted by applicable law Fujitsu Microelectronics Europe GmbH disclaims all other warranties, whether expressed or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the Product is not designated.
4. To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH's and its suppliers' liability is restricted to intention and gross negligence.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES

To the maximum extent permitted by applicable law, in no event shall Fujitsu Microelectronics Europe GmbH and its suppliers be liable for any damages whatsoever (including but without limitation, consequential and/or indirect damages for personal injury, assets of substantial value, loss of profits, interruption of business operation, loss of information, or any other monetary or pecuniary loss) arising from the use of the Product.

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect.

1. Output signal of the encoder

The typical output of a 2-bit quadrature encoder, like the Series 62A from www.grayhill.com, looks like in figure 1-1. Alternately output A and B will change its logic level by each (rotation-) step.

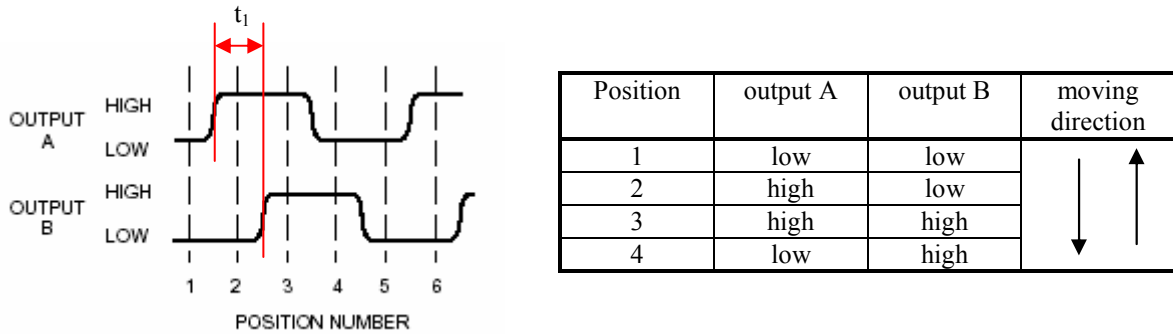


Figure 1-1: output waveform and truth-table

As easily can be seen, each step causes an edge change either of output A or of output B, and ends in a new position. The code repeats every 4 positions: 1 – 2 – 3 – 4 – 1 – 2 – 3 – ...

To determine the moving direction, two possibilities are imaginable:

- I. starting with the actual position the next edge-change has to be examined; this means for each actual position only two fixed edge-changes are possible, one for a right-step, another for a left-step
 e.g.: actual position = 3, edge-change = outA: HL → right-step (3 → 4)
 e.g.: actual position = 3, edge-change = outB: HL → left-step (3 → 2)
- II. starting with the actual position a comparison with the new position will result in the moving-direction
 e.g.: actual position = 3, new position = 4 → right-step (3 → 4)
 e.g.: actual position = 3, new position = 2 → left-step (3 → 2)
 e.g.: actual position = 3, new position = 1 → invalid

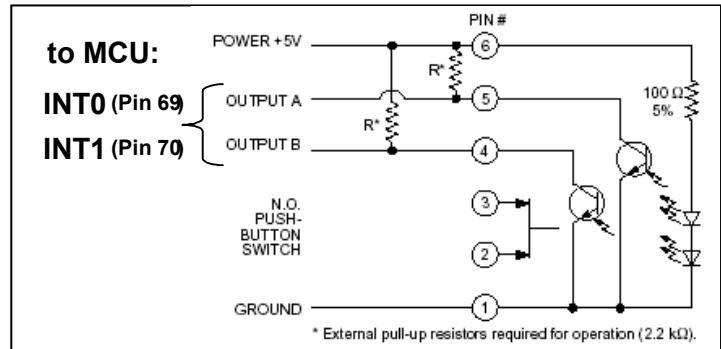
Also the (rotation-) speed can be evaluated, if the time t_1 between two edges is measured.

The second method seems to be more efficient because in any case the new position has to be saved in order to check the next step.

Two sample programs will be discussed in the following by using different resources of the microcontroller.

2. Software: using “External Interrupts”

By the Request Level Setting Register (ELVR) the request event for an external pin can be defined as follows: low-level, high-level, rising- or falling-edge. This means that within the Interrupt-Service-Routine the settings have to be changed in order to catch the next edge of this input that will be reverse.



The following code is an extract from the whole program in order to discuss the most important software parts. The External Interrupts INT0 and INT1 will be used to connect the Rotary-Encoder output A and B. Although the software is designed for MB90540 series, it can easily adapted to other MCUs, too.

Vector.c

In `vector.c` the Interrupt-Routines has to be defined and the related Interrupt-Levels (ICRxx) has to be set.

```
ICR02 = 2;          /* IRQ15, IRQ16 */

__interrupt void IRQ_RotaryEncoder (void);

#pragma intvect IRQ_RotaryEncoder 15 /* external interrupt INT0/INT1
*/
```

Hint 1: One Interrupt-Control is shared by two Interrupt-sources
e.g. ICR02 will set IRQ-level of IRQ15 and IRQ16

Hint 2: some Interrupt-sources will use the same the Interrupt-Handler, like INT0 and INT1
This makes additional software necessary within the Service-Interrupt-Routine in order to find out what source has caused the IRQ.
e.g.: IRQ-Handler 15 will be called by INT0 and INT1, too.
Well, in case of the rotary-encoder, this means an advantage, because both external interrupts will be used, but in other applications it may be disturbing.

Main.c

In `Main.c` the registers for the use of external interrupts are initialised. Two global variables `enc_position` and `enc_error` will contain the actual status of the rotary-encoder. The real checking of the rotary-encoder is interrupt-driven, so the user application can react on these values.

```

void InitEncoder(void)
{
    DDR9_D90 = 0;           // Rotary-Encoder Pin A is Input
    DDR9_D91 = 0;           // Rotary-Encoder Pin B is Input

    enc_old = PDR9 & 0x03; // save actual Rotary-Encoder position
                           // as startposition

    ELVR_LA0 = PDR9_P90;    // Depending on Startposition
                           // extINT has to be prepared
    ELVR_LA1 = PDR9_P91;    // for the next expected Interrupt edge

    ELVR_LB0 = 1;           // always edge-detection is used
    ELVR_LB1 = 1;           // always edge-detection is used

    EIRR = EIRR & 0xfc;    // clear Ext-IRQ-Flags INT0, INT1

    ENIR_EN0 = 1;           // enable ExtINT0
    ENIR_EN1 = 1;           // enable ExtINT0
}

```

The status of the rotary-encoder will be updated each time, when the prepared edge-change (see register ELVR) at INT0 or INT1 (encoder output A or B) is detected.

Then an Interrupt-Service-Routine will be called, where the new position of the encoder is checked. By comparison of the last and the new position the moving-direction can be tracked as described in chapter 1.

In dependency of the new position the Request Level Setting Register (ELVR) has to be set again for the next expected edge-changes.

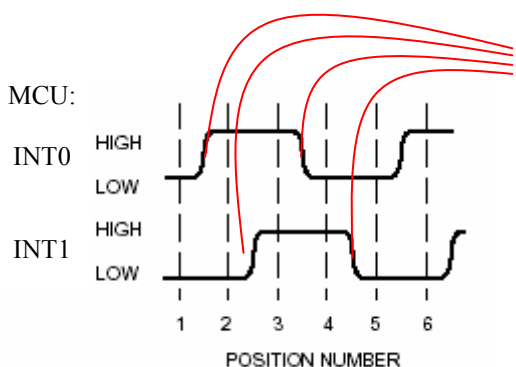


Figure 2-1 : every edge-transition will cause an IRQ

```

__interrupt
void IRQ_RotaryEncoder(void)
{
    /* comparison of
       last and new state:      */

    enc = (PDR9 & 0x03); // read encoder
                        // switch (enc_old)
    {
        case ENC_state00:
            switch (enc)
            {
                case ENC_state01:
                    enc_position++;
                    ELVR_LA0 = 1; break;
                case ENC_state10:
                    enc_position--;
                    ELVR_LA1 = 1; break;
                case ENC_state00: enc_error++;
                    ELVR_LA1 = 0; ELVR_LA0 = 0;
                    break;
                case ENC_state11: enc_error++;
                    ELVR_LA1 = 1; ELVR_LA0 = 1;
                    break;
            }
            break;

        case ENC_state01: ... break;
        case ENC_state11: ... break;
        case ENC_state10: ... break;
    }

    enc_old = enc; // save new position
    EIRR = EIRR & 0xfc; // clear IRQ
}

```

```

__interrupt void IRQ_RotaryEncoder(void) //
{
  enc = PDR9 & 0x03; // read-encoder (INT0, INT1)
  switch (enc_old)
  {
    case ENC_state00:
      switch (enc)
      {
        case ENC_state01: enc_position++; ELVR_LA0 = 1; break;
        case ENC_state10: enc_position--; ELVR_LA1 = 1; break;
        case ENC_state00: enc_error++; ELVR_LA1 = 0; ELVR_LA0 = 0; break;
        case ENC_state11: enc_error++; ELVR_LA1 = 1; ELVR_LA0 = 1; break;
      }
      break;

    case ENC_state01:
      switch (enc)
      {
        case ENC_state11: enc_position++; ELVR_LA1 = 1; break;
        case ENC_state00: enc_position--; ELVR_LA0 = 0; break;
        case ENC_state01: enc_error++; ELVR_LA1 = 0; ELVR_LA0 = 1; break;
        case ENC_state10: enc_error++; ELVR_LA1 = 1; ELVR_LA0 = 0; break;
      }
      break;

    case ENC_state11:
      switch (enc)
      {
        case ENC_state10: enc_position++; ELVR_LA0 = 0; break;
        case ENC_state01: enc_position--; ELVR_LA1 = 0; break;
        case ENC_state11: enc_error++; ELVR_LA1 = 1; ELVR_LA0 = 1; break;
        case ENC_state00: enc_error++; ELVR_LA1 = 0; ELVR_LA0 = 0; break;
      }
      break;

    case ENC_state10:
      switch (enc)
      {
        case ENC_state00: enc_position++; ELVR_LA1 = 0; break;
        case ENC_state11: enc_position--; ELVR_LA0 = 1; break;
        case ENC_state10: enc_error++; ELVR_LA1 = 1; ELVR_LA0 = 0; break;
        case ENC_state01: enc_error++; ELVR_LA1 = 0; ELVR_LA0 = 1; break;
      }
      break;
  }

  enc_old = enc; // save new position
  EIRR &= 0xf3; // clear ExternalInterruptRequest-Flags INT0, INT1
}

```

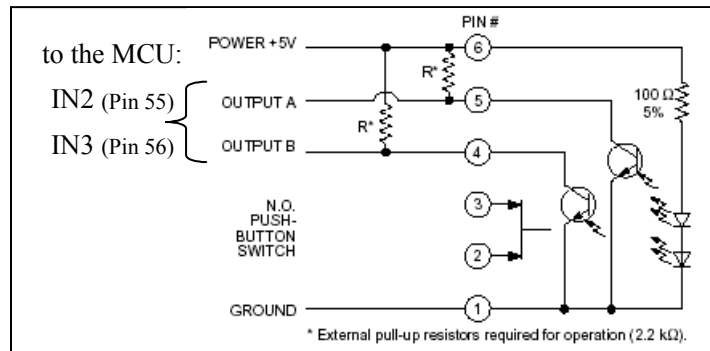
Figure 2-2: the whole Interrupt-Service-Routine for position-evaluation by using External Interrupts

If also the speed (= time between two edges) shall be measured, any available timer (Reload-Timer, free-running I/O-Timer, Timebase-timer, etc.) can be used. When the Interrupt-Service-Routine is entered the time counter value has to be compared with the last one in order to calculate the speed. An example of speed-measurement is given in the next chapter when the use of the rotary-encoder together with the Input-Capture-Unit is explained.

3. Software: using the “Input-Capture-Unit”

In contrast with the external Interrupts the Input-Capture-Unit of the 16LX-series has the advantage of *both edge detection*.

This makes the software much easier. With the Free-Running-Timer (16Bit - I/O-Timer), that is related to the Input-Capture-Unit¹, time- (= speed) measurement is very simple, too.



The following code is an extract from the whole program in order to discuss the most important software parts. Interrupt Channel 2/3 will be used to connect the Rotary-Encoder output A and B. Although the software is designed for MB90540series, it can easily adapted to other MCUs, too.

Vector.c

In `vector.c` the Interrupt-Routines has to be defined and the related Interrupt-Levels (ICRxx) has to be set.

```
ICR04 = 4;          /* IRQ19 (Free Running Timer)      IRQ20 */
ICR09 = 2;          /* IRQ29 (input Capture 2 and 3)  IRQ30 */

__interrupt void IRQ_FreeRunningTimer (void);
__interrupt void IRQ_RotaryEncoder (void);

#pragma intvect IRQ_FreeRunningTimer 19 /* I/O timer          */
#pragma intvect IRQ_RotaryEncoder 30 /* input capture CH.2/3 */
```

Hint 1: One Interrupt-Control is shared by two Interrupt-sources
e.g. ICR04 will set IRQ-level of IRQ19 and IRQ20

Hint 2: some Interrupt-sources will use the same the Interrupt-Handler.

This makes additional software necessary within the Service-Interrupt-Routine in order to find out what source has caused the IRQ.

e.g.: IRQ-Handler 30 will be called by Input-Capture channel 2 and by channel 3, too.

Well, in case of the rotary-encoder, this means an advantage, because both channels will be used, but in other applications it may be disturbing.

Main.c

In `Main.c` the Input-Capture-Unit and the Free-Running-Timer are initialised. Two global variables `enc_position` and `speed` will contain the actual status of the rotary-encoder. The real checking of the rotary-encoder is interrupt-driven, so the user application can react on these values.

Some basic-settings are necessary to select the ICU and the input-port used for the encoder:

```
void InitEncoder(void)
{
    DDR7_D72 = 0;           // ICU Ch2: Rotary-Encoder Pin A is Input
    DDR7_D73 = 0;           // ICU Ch3: Rotary-Encoder Pin B is Input
    enc_old = (PDR7 & 0x0c); // save actual Rotary-Encoder position
    ICS23 = 0x3f;          // enable ICU, both edge detection
}
```

The status of the rotary-encoder will be updated each time, when an edge-change at Input-Capture-Channel 2 or Channel 3 (encoder output A or B) is detected.

Then an Interrupt-Service-Routine will be called, where the new position of the encoder is checked. By comparison of the last and the new position the moving-direction can be tracked as described in chapter 1.

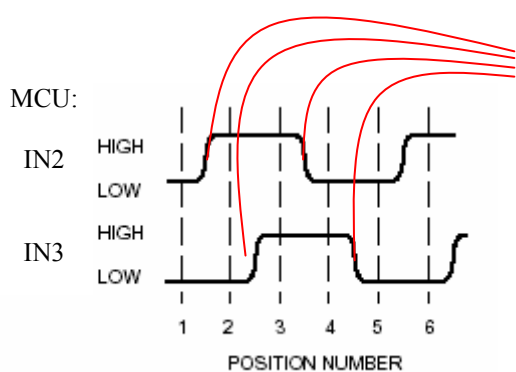


Figure 3-1 : every edge-transition will cause IRQ

```
__interrupt
void IRQ_RotaryEncoder(void)
{
    /* comparison of
       last and new state:      */

    enc = (PDR7 & 0x0c); // read encoder
                          switch (enc_old)
    {
        case ENC_state00: // last position
                          switch (enc) // new position
        {
            case ENC_state01:
                enc_position++; break;

            case ENC_state10:
                enc_position--; break;

            default: enc_error++;
        }
        break;

        case ENC_state01: ... break;
        case ENC_state11: ... break;
        case ENC_state10: ... break;
    }

    enc_old = enc; // save new position
    ICS23 = 0x3f; // clear ICU IRQ-Flags
} //Return from IRQ
```

Speed-Measurement:

For speed measurement the Free-Running-Timer (16Bit I/O-Timer) has to set up.

The range and resolution of the speed, that wants to be evaluated, depends on the application.

```
unsigned int FRT_overrun; // realtime-overflow-counter
unsigned int save_overrun; // save overflow-counter with entrance to
ISR
unsigned long time; // calculated speed

void InitFreeRunningTimer(void)
{
    TCCS = 0x25; //IRQ enable, 1us (makes it easy for further
calculations)
}
```

But always should be kept in mind, that the Free-Running-Timer will overflow, when the time between two steps will exceed the 16bit range of the counter. Therefore the count clock should be selected as slow as possible, in order to increase performance by less interrupt-calls caused by the overflow.

Nevertheless the overflow-IRQs has to be counted for a correct time evaluation later on.

```

__interrupt void IRQ_FreeRunningTimer(void)
{
    if (FRT_overrun != 0xffff)
        FRT_overrun++;           // count overflows
    TCCS = 0x21;                // clear Interrupt-Flag IFV;
}

```

With each interrupt the value of the Free-Running-Timer is stored automatically to the Input-Capture-Register related to the Interrupt source (channel 2 or 3). Because both sources are triggered alternately, the difference between both Input-Capture-Registers has to build for time calculation, with relation to the channel that caused the interrupt. But also it has to be taken care of the overruns.

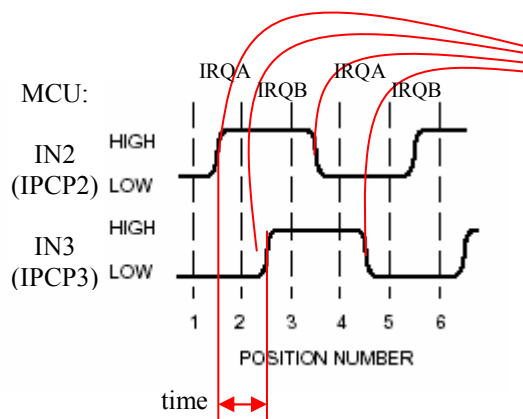


Figure 3-2 : every edge-transition will cause an IRQ, but the channel has to be determined for right time-calculation

In order to get real speed-values (e.g. m/s), more calculations based on the time-value may be necessary, but this depends on the final application.

```

__interrupt
void IRQ_RotaryEncoder(void)
{
    /* calculation of time between
    last IRQ and yet:
    time = |(IPCP2-IPCP3)| + overrun
    */

    if (ICS23 & 0x40) // check for IRQA
        if (IPCP2 >= IPCP3)
            time = (IPCP2 - IPCP3)
                + (save_overrun * 0x10000);
        else
            time = (save_overrun * 0x10000)
                - IPCP3 + IPCP2;
    else
        if (IPCP3 >= IPCP2)
            time = (IPCP3 - IPCP2)
                + (save_overrun * 0x10000);
        else
            time = (save_overrun * 0x10000)
                - IPCP2 + IPCP3;

    /* comparison of
    last and new state: */

    switch (enc_old)
    {
        case ENC_state00: ... break
        case ENC_state01: ... break;
        case ENC_state11: ... break;
        case ENC_state10: ... break;
    }
    enc_old = enc; // save new position
    ICS23 = 0x3f; // clear ICU IRQ-Flags
} //Return from IRQ

```

¹Hint:

Because the Free-Running-Timer (16Bit I/O-Timer) may be used for both Output- and Input-Capture-Unit, interference has to be prevented by software.

```

__interrupt void IRQ_RotaryEncoder(void)
{
    __DI();                // store new encoder position
    enc = (PDR7 & 0x0c);   // read encoder (IN2, IN3)
    save_overflow = FRT_overflow; FRT_overflow = 0; // and time-overflows
    __EI();

    // time calculation: check what edge caused the IRQ
    // and build difference with last one

    if (ICS23 & 0x40) // check for IRQA
        if (IPCP2 >= IPCP3)
            time = (IPCP2 - IPCP3) + (save_overflow * 0x10000);
        else
            time = (save_overflow * 0x10000) - IPCP3 + IPCP2;
    else
        if (IPCP3 >= IPCP2)
            time = (IPCP3 - IPCP2) + (save_overflow * 0x10000);
        else
            time = (save_overflow * 0x10000) - IPCP2 + IPCP3;

    switch (enc_old)
    {
        case ENC_state00:
            switch (enc)
            { case ENC_state01: enc_position++; break;
              case ENC_state10: enc_position--; break;
              default: enc_error++; break; }
            break;

        case ENC_state10:
            switch (enc)
            { case ENC_state00: enc_position++; break;
              case ENC_state11: enc_position--; break;
              default: enc_error++; break; }
            break;

        case ENC_state11:
            switch (enc)
            { case ENC_state01: enc_position--; break;
              case ENC_state10: enc_position++; break;
              default: enc_error++; break; }
            break;

        case ENC_state01:
            switch (enc)
            { case ENC_state00: enc_position--; break;
              case ENC_state11: enc_position++; break;
              default: enc_error++; break; }
            break;

        default : while(1); // never come here
    }
    enc_old = enc;        // save new Rotary position as old one for next compare
    ICS23 = 0x3f;       // clear ICU IRQ-Flags
}

```

Figure 3-3: whole Interrupt-Service-Routine for position- and speed-evaluation by using the Input-Capture-Unit

4. Maximum Frequency calculation

The evaluation of the encoder-signal is completely done within the Interrupt-Service-Routine. So the theoretically time that is need for the evaluation of one moving-step, that means one interrupt call, is calculated by the IRQ-call/return itself and the time needed within the Interrupt-Service-Routine:

$$\text{total_time} = \text{time}_{(\text{Jump to Interrupt})} + \text{time}_{(\text{Interrupt-Service-Routine})} + \text{time}_{(\text{return from Interrupt})}$$

In best case, when no other interrupts are active, “Jump to / Return from” interrupt will delay about $5\mu\text{s}$ ¹. The Interrupt-Service-Routine used in this example will need around $15\mu\text{s}$ ¹. With the formula given above the minimum time results in $25\mu\text{s}/\text{step}$ ¹, that means $f_{\text{max}} = 40\text{kHz}$ ¹.

This value depends very much on the application, and may increase by software optimisation but will decrease if further calculations are necessary.

¹based on 16MHz internal CPU-clock

5. Conclusion

The External-Interrupt-Unit as well as the Input-Capture-Unit of the Fujitsu F²MC-16LX-Series is very well suitable in use with a 2-bit quadrature encoder. At least it depends on the application, what unit to prefer.

The given software examples are not optimised, because this also will differ from application to application. Especially the time-calculation may be changed. Depending on the resolution a `short` or `int` variable should be used instead of `long`. Further it may be important to get the moving direction back (left/right or forward/backwards) instead of the `enc_position`.

So this software may only be a template for creating own applications.

6. Appendix A: sample project: rot_enc_extIRQ

MAIN.C:

```
/*-----  
  MAIN.C (rot_enc_extIRQ)  
  - description  
  - See README.TXT for project description and disclaimer.  
/*-----*/  
  
#include "mb90540.h"  
  
/*-----*/  
/* defines, constants and variables */  
/*-----*/  
  
#define ENC_state00 0x00  
#define ENC_state01 0x01  
#define ENC_statel1 0x03  
#define ENC_statel0 0x02  
  
  signed int  enc_position;  
  unsigned int  enc_error;  
  unsigned char enc, enc_old;  
  
/*****  
* void encoder_init(void)  
*  
* DESCRIPTION: Interrup Request Detection Factor - Initialization  
*****/  
void InitEncoder(void)  
{  
  DDR9_D90 = 0;          // Rotary-Encoder Pin A is Input  
  DDR9_D91 = 0;          // Rotary-Encoder Pin B is Input  
  
  enc_old = PDR9 & 0x03; // save (start with) actual Rotary-Encoder position  
  
  ELVR_LA0 = PDR9_P90;   // Depending on Startposition extINT has to be  
  prepared  
  ELVR_LA1 = PDR9_P91;   // for the next expected Interrupt edge  
  
  ELVR_LB0 = 1;          // always edge-detection is used  
  ELVR_LB1 = 1;          // always edge-detection is used  
  
  EIRR = EIRR & 0xfc;   // clear ExternalInterruptRequest-Flags INT0, INT1  
  
  ENIR_EN0 = 1;          // enable ExtINT0  
  ENIR_EN1 = 1;          // enable ExtINT0  
}
```

```

__interrupt void IRQ_RotaryEncoder(void)//
{
  enc = PDR9 & 0x03; // read-encoder (INT0, INT1)
  switch (enc_old)
  {
    case ENC_state00:
      switch (enc)
      {
        case ENC_state01: enc_position++; ELVR_LA0 = 1; break;
        case ENC_state10: enc_position--; ELVR_LA1 = 1; break;
        case ENC_state00: enc_error++; ELVR_LA1 = 0; ELVR_LA0 = 0; break;
        case ENC_state11: enc_error++; ELVR_LA1 = 1; ELVR_LA0 = 1; break;
      }
      break;

    case ENC_state01:
      switch (enc)
      {
        case ENC_state11: enc_position++; ELVR_LA1 = 1; break;
        case ENC_state00: enc_position--; ELVR_LA0 = 0; break;
        case ENC_state01: enc_error++; ELVR_LA1 = 0; ELVR_LA0 = 1; break;
        case ENC_state10: enc_error++; ELVR_LA1 = 1; ELVR_LA0 = 0; break;
      }
      break;

    case ENC_state11:
      switch (enc)
      {
        case ENC_state10: enc_position++; ELVR_LA0 = 0; break;
        case ENC_state01: enc_position--; ELVR_LA1 = 0; break;
        case ENC_state11: enc_error++; ELVR_LA1 = 1; ELVR_LA0 = 1; break;
        case ENC_state00: enc_error++; ELVR_LA1 = 0; ELVR_LA0 = 0; break;
      }
      break;

    case ENC_state10:
      switch (enc)
      {
        case ENC_state00: enc_position++; ELVR_LA1 = 0; break;
        case ENC_state11: enc_position--; ELVR_LA0 = 1; break;
        case ENC_state10: enc_error++; ELVR_LA1 = 1; ELVR_LA0 = 0; break;
        case ENC_state01: enc_error++; ELVR_LA1 = 0; ELVR_LA0 = 1; break;
      }
      break;
  }

  enc_old = enc; // save new position

  EIRR = EIRR & 0xfc; // clear ExternalInterruptRequest-Flags INT0, INT1
}

```

```

/*-----*/
/*-----*/
/*  M A I N / M A I N / M A I N / M A I N / M A I N / M A I N */
/*-----*/
/*-----*/

void main(void)
{
    InitIrqLevels();

    __set_il(7);          /* Set ILM to 7 */
                        /* allow all interrupts levels */
    __EI();              /* enable interrupts at all */

    enc_position = 0;    /* Impuls-counter for rotary-encoder */
    enc_error    = 0;    /* enc_error will be increased, if rotary movement is
too fast */

    InitEncoder();

    while(1)
    {
        __asm("\tNOP"); /* nothing to do,
                        Encoder is checked in InterruptServiceRoutine */
    }
}

```

VECTOR.C: only the changes are listed (please refer to template.prj)

```

/*-----*/
VECTOR.C (rot_enc_extIRQ)
- description
- See README.TXT for project description and disclaimer.
/*-----*/

#include "mb90540.h"

void InitIrqLevels(void)
{
/*  ICRxx          shared IRQs for ICR */

ICR02 = 2;        /*  IRQ15 IRQ16 */

__interrupt void IRQ_RotaryEncoder (void);

#pragma intvect IRQ_RotaryEncoder 15 /* external interrupt INT0/INT1 */

```

7. Appendix B: sample project: rot_enc_ICU

MAIN.C:

```
/*-----  
  MAIN.C  
  - description  
  - See README.TXT for project description and disclaimer.  
/*-----*/  
  
#include "mb90540.h"  
  
/*-----*/  
/* defines, constants and variables */  
/*-----*/  
  
#define ENC_state00 0x00  
#define ENC_state01 0x04  
#define ENC_state11 0x0C  
#define ENC_state10 0x08  
  
  signed int  enc_position;  
  unsigned int  enc_error;  
  unsigned char enc, enc_old;  
  
  unsigned int FRT_overrun;  
  unsigned int save_overrun, old_IPCP;  
  unsigned long time;  
  
/*-----*/  
/* Inilisation of the FreeRunningTimer for speed-measurement */  
/*-----*/  
void InitFreeRunningTimer(void)  
{  
  TCCS = 0x25;          // IRQ enable, 1us  
}  
  
/*-----*/  
/* count overflow of FreeRunningTimer in order to calculate correct time*/  
/*-----*/  
__interrupt void IRQ_FreeRunningTimer(void)  
{  
  if (FRT_overrun != 0xffff)  
    FRT_overrun++;  
  TCCS = 0x21;          // clear IFV  
}
```

```

/*-----*/
/* Inilisation of the InputCaptureUnit */
/*-----*/
void InitEncoder(void)
{
    DDR7_D72 = 0;           // ICU Ch2: Rotary-Encoder Pin A is Input
    DDR7_D73 = 0;           // ICU Ch3: Rotary-Encoder Pin B is Input
    enc_old = (PDR7 & 0x0c); // save actual Rotary-Encoder position
    ICS23    = 0x3f;        // enable ICU, both edge detection
}

/*-----*/
/* get new Rotary-Encoder-position when IRQ of InputCaptureUnit occurred */
/*-----*/
__interrupt void IRQ_RotaryEncoder(void)
{
    __DI();                // store new encoder position
    enc = (PDR7 & 0x0c);    // read encoder (IN2, IN3)
    save_overrun = FRT_overrun; FRT_overrun = 0; // and time-overflows
    __EI();

    // time calculation: check what edge caused the IRQ
    // and build difference with last one

    if (ICS23 & 0x40) // check for IRQA
        if (IPCP2 >= IPCP3)
            time = (IPCP2 - IPCP3) + (save_overrun * 0x10000);
        else
            time = (save_overrun * 0x10000) - IPCP3 + IPCP2;
    else
        if (IPCP3 >= IPCP2)
            time = (IPCP3 - IPCP2) + (save_overrun * 0x10000);
        else
            time = (save_overrun * 0x10000) - IPCP2 + IPCP3;

    switch (enc_old)
    {
        case ENC_state00:
            switch (enc)
            { case ENC_state01: enc_position++; break;
              case ENC_state10: enc_position--; break;
              default: enc_error++; break; }
            break;

        case ENC_state10:
            switch (enc)
            { case ENC_state00: enc_position++; break;
              case ENC_state11: enc_position--; break;
              default: enc_error++; break; }
            break;
    }
}

```

```

    case ENC_statel1:
        switch (enc)
        { case ENC_state01: enc_position--; break;
          case ENC_statel0: enc_position++; break;
          default: enc_error++; break; }
        break;

    case ENC_state01:
        switch (enc)
        { case ENC_state00: enc_position--; break;
          case ENC_statel1: enc_position++; break;
          default: enc_error++; break; }
        break;

    default : while(1); // never come here
}
enc_old = enc;          // save new Rotary position as old one for next compare
ICS23  = 0x3f;         // clear ICU IRQ-Flags
}

/*-----*/
/*-----*/
/*  M A I N / M A I N / M A I N / M A I N / M A I N / M A I N  */
/*-----*/
/*-----*/

void main(void)
{
    InitIrqLevels();

    __set_il(7);        /* Set ILM to 7 */
                        /* allow all interrupts levels */
    __EI();             /* enable interrupts at all */

    enc_position = 0;   // Impuls-counter for rotary-encoder
    enc_error    = 0;   // enc_error will be increased,
                        // if rotary movement is to fast
    FRT_overrun  = 0;   // for time-measurement

    InitFreeRunningTimer();
    InitEncoder();

    while(1)
    {
        __asm("\tNOP"); /* nothing to do,
                        Encoder is checked in InterruptServiceRoutine */
    }
}

```

VECTOR.C: only the changes are listed (please refer to template.prj)

```
/*-----  
VECTOR.C (rot_enc_ICU)  
See README.TXT for project description and disclaimer.  
/*-----*/  
  
#include "mb90540.h"  
  
void InitIrqLevels(void)  
{  
/* ICRxx shared IRQs for ICR */  
  
ICR04 = 4;      /* IRQ19 IRQ20 */  
ICR09 = 2;      /* IRQ29 IRQ30 */  
  
__interrupt void IRQ_FreeRunningTimer (void);  
__interrupt void IRQ_RotaryEncoder (void);  
  
#pragma intvect IRQ_FreeRunningTimer 19 /* I/O timer */  
#pragma intvect IRQ_RotaryEncoder 30 /* input capture CH.2/3 */
```