Application Note

AN2362/D
Rev. 0, 10/2002

Using the Fast Quadrature Decode TPU Function (FQD) with the MPC500 Family

Jeff Loeliger
TECD

This TPU Programming Note is intended to provide simple C interface routines to the fast quadrature decode TPU function (FQD). [1] The routines are targeted for the MPC500 family of devices, but they should be easy to use with any device that has a TPU.

1 Functional Overview

The fast quadrature decode function (FQD) is a TPU input function that uses two channels to decode a pair of out of phase signals in order to increment or decrement a (position) counter. It is particularly useful for decoding position and direction information from a slotted encoder in motion control systems, thus replacing expensive external solutions (see Figure 1).

[1]The information in this Programming Note is based on TPUPN02. It is intended to compliment the information found in that Programming Note.

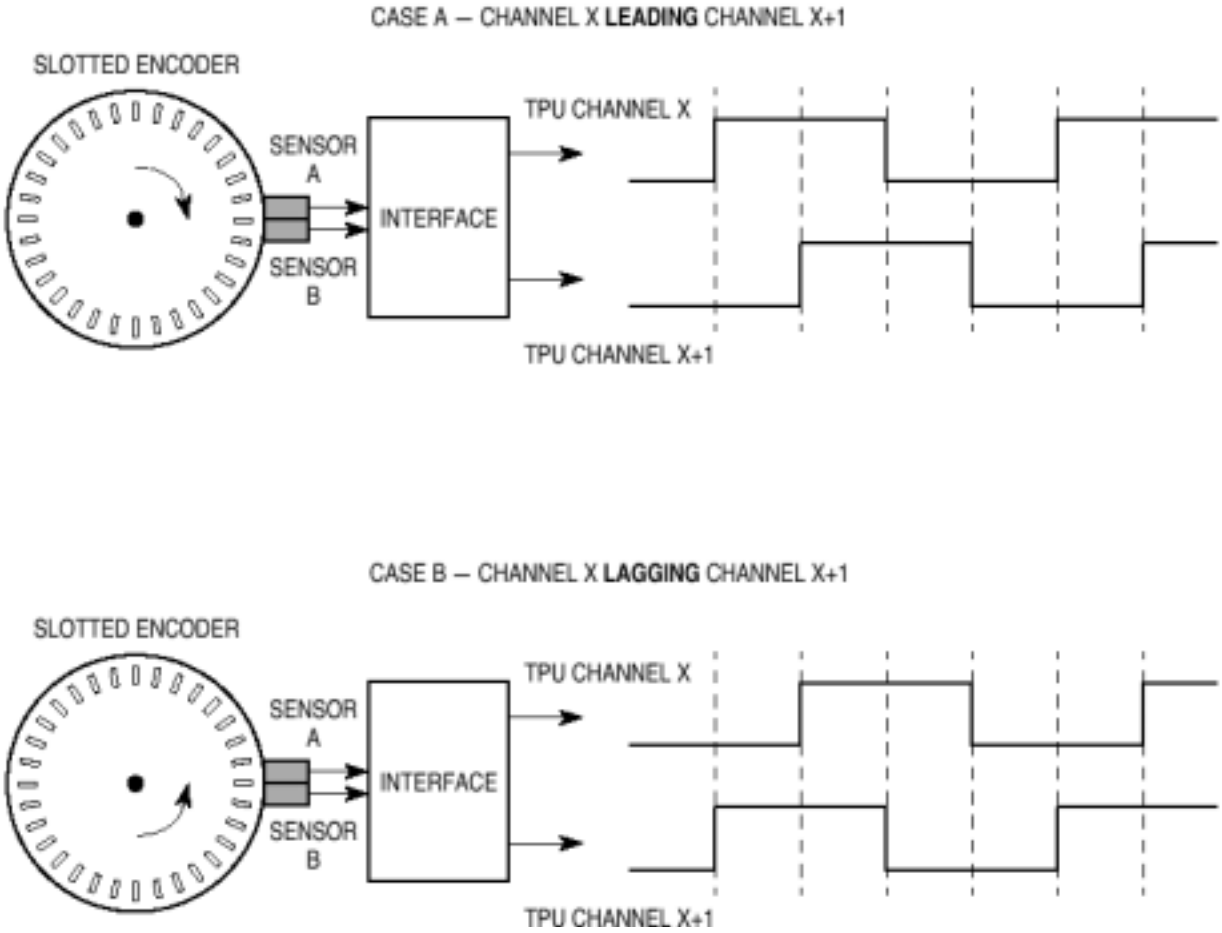

Figure 1. Lead/Lag Timing

2 Detailed Description

The FQD function uses a pair of adjacent TPU channels to decode quadrature signals into a 16 bit counter in parameter RAM that is updated when a valid transition is detected on either one of the two inputs i.e. full '4x' every edge resolution is derived from the encoder signals. The counter is incremented or decremented depending on the lead/lag relationship of the two signals at the time of servicing the transition. The user can read or write the counter at any time using the *tpu_fqd_position* and *tpu_fqd_write_position* functions. The counter is free running, overflowing to 0x0000 or under flowing to 0xFFFF depending on direction.

In systems where the counter may over or under flow, the user should ensure that the CPU reads the counter periodically, with the maximum period between reads being equivalent to 0x8000 counts at maximum signal frequency. Two’s complement arithmetic can then be used by the CPU to maintain position and direction information.

When initialized the FQD function is automatically configured so that the first edge on either channel will result in a counter update.

The FQD function differs from QDEC in having two modes of operation: 'Normal' and 'Fast'. Since the two FQD channels (which must always be adjacent) operate differently, the one with the lower channel number shall be referred to as the 'primary' channel and the other the 'secondary' channel.

In operation, the CPU dynamically switches the FQD function between modes depending on the current encoder speed; this is shown in example listing 2. The following sections describe how the two modes operate.

2.1 Normal Mode

In normal mode, both quadrature signals are decoded by the TPU and the counter updated by 1 for each valid transition on either channel - see Figure 2. The counter is incremented or decremented depending on the 'lead/lag' relationship of the two signals at the time of transition service.

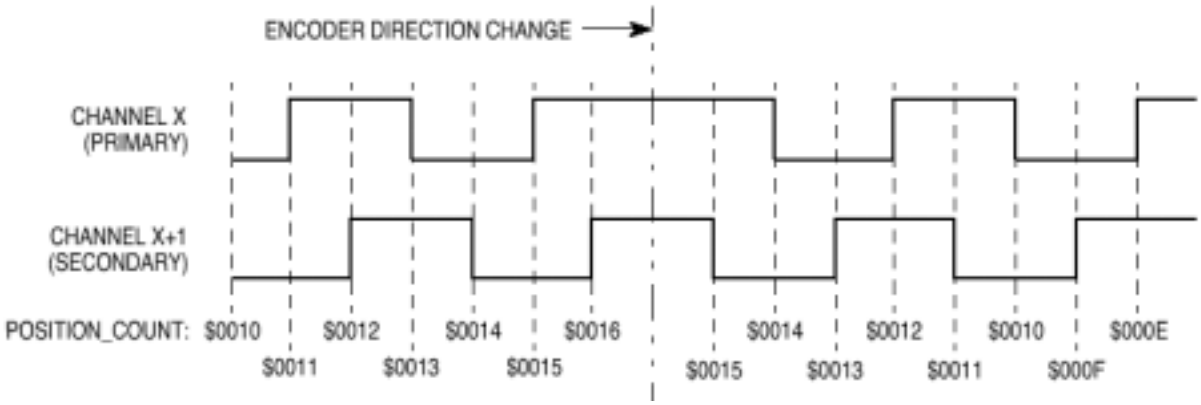

Figure 2. Normal Mode Timing

2.2 Fast Mode

In fast mode, only one of the two quadrature signals (the primary channel) is serviced and the counter is updated by 4 for each rising transition (all falling transitions are ignored) - see Figure 3. The effect of this is to more than quadruple the maximum count rate that the TPU can reliably decode compared with normal mode. No direction decoding is done in fast mode, the counter being updated in the same direction as when the last transition was serviced in normal mode. This is not an issue because when running quickly the signal is not likely to change direction. When the signal starts slowing down to change direction the FQD function should be put back into normal mode.

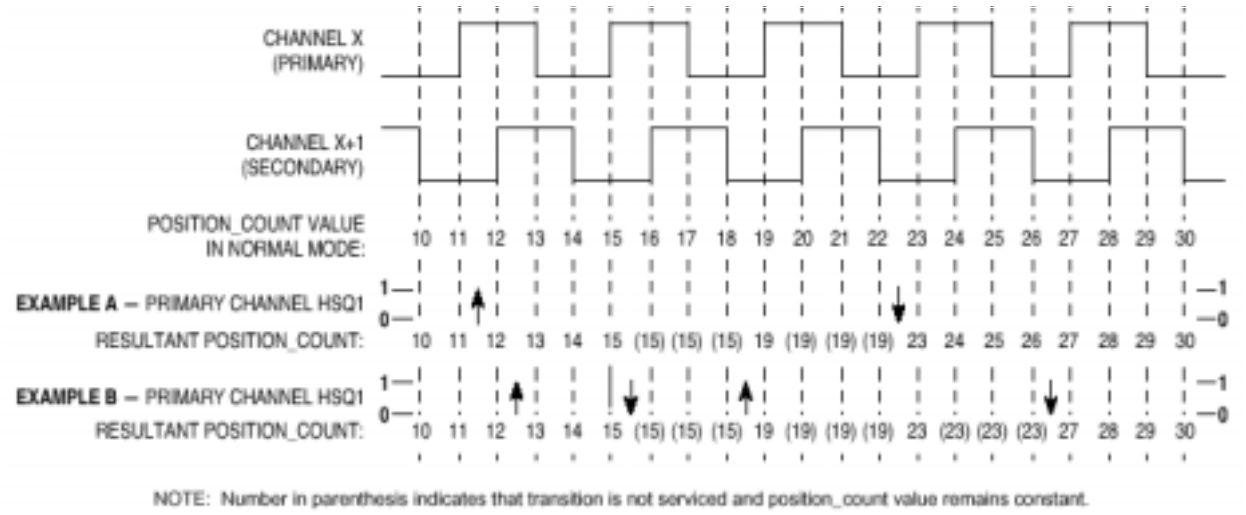

Figure 3. Fast Mode Timing

The mode of operation of the FQD can be changed at any time by the CPU by using the *tpu_fqd_mode* function. Any requested change in operating mode will take effect when the next rising transition on the primary channel is serviced.

No counts are lost when switching in and out of fast mode, although there is a +/- 2 lsb uncertainty in the counter while in fast mode due to the 'by 4' update.

If the application performance requires that the fast mode be used then the CPU should start FQD in normal mode and switch into fast mode when the desired speed (from periodic reads of the position counter) is above a certain threshold an example of this is shown in example listing 2. The function should then be run in fast mode until the speed falls below the threshold when the CPU should switch the function back into normal mode. The speed threshold at which the function should switch in and out of fast mode is dependent on overall TPU system activity and must be evaluated for each application.

2.2.1 Time Stamp

The FQD function also provides a time stamp in normal mode referenced to TCR1 for every valid signal edge and the ability for the host CPU to obtain a current TCR1 value. These features allow position and speed interpolation by the host CPU between quadrature edges at very slow count rates.

2.2.2 Discrete Input / Transition Counter

A single channel programmed to run FQD could be used as a digital input pin with a transition counter.

2.3 FQD C Level API

Rather than controlling the TPU registers directly, the FQD routines in this TPU Programming Note may be used to provide a simple and easy interface. There are 6 routines for controlling the FQD function in 2 files (tpu_fqd.h and tpu_fqd.c). The tpu_fqd.h file should be included in any files that use the routines. This file contains the function prototypes and useful # defines. The tpu_fqd.c file contains of the following routines, which are described in detail in the sections below:

- General TPU Functions (defined in mpc500_util.h)
  - void tpu_enable(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority);
  - void tpu_disable (struct TPU3_tag *tpu, UINT8 channel);
- Initialization Functions
  - void tpu_fqd_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, INT16 init_position);
  - void tpu_fqd_init_trans_count(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority);
- Change Operation Mode Function
  - void tpu_fqd_mode(struct TPU3_tag *tpu, UINT8 channel, UINT8 mode);
- Return Value Functions
  - UINT8 tpu_fqd_current_mode(struct TPU3_tag *tpu, UINT8 channel);
  - INT16 tpu_fqd_position(struct TPU3_tag *tpu, UINT8 channel);
  - void tpu_fqd_data(struct TPU3_tag *tpu, UINT8 channel, INT16 *tcr1, INT16 *edge, INT16 *primary_pin, INT16 *secondary_pin)

2.3.1 void tpu_enable

This can also be used to change the priority. Both FQD channels must have the same priority. Both channels should be disabled before changing priority.

2.3.2 void tpu_disable

Both FQD channels must be disabled together.

2.3.3 void tpu_fqd_init

This function is used to initialize a pair of channels to run the FQD function. This function has 4 parameters:

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h
- channel - This is the channel number of the primary FQD channel. The next channel is used as the secondary.
- priority - This is the priority to assign to both channels. This parameter should be assigned a value of TPU_PRIORITY_HIGH, TPU_PRIORITY_MIDDLE or TPU_PRIORITY_LOW. The TPU priorities are defined in mpc500_utils.h.
- init_position - This is the starting position count.

Care should be taken when initializing TPU channels. The TPU's behavior may be unpredictable if a channel is reconfigured while it is running. The channels should be stopped before they are configured. This is done by setting the channel's priority to disabled. If the channel is currently being serviced when the priority is set to disable it will continue to service the channel until the state ends. To make sure the channel is not being service you need to wait for the longest state execution time after disabling the channel. All channels are disabled out of reset so the channels can be configured immediately from reset.

The *tpu_fqd_init* function attempts to wait between the disabling of the channels before it starts configuring them, however the actual execution speed of the code will be depend on the specific system. If you are not

configuring the channels from reset, then ideally it is best to have the functions disabled before calling this function. TPU channels can be disabled by using the *tpu_disable* function in the mpc500_utils.c file. For example, disabling channels 5 & 6 is done like this:

```
tpu_disable(tpu, 5);

tpu_disable(tpu,6);
```

2.3.4 void tpu_fqd_init_trans_count

This function is used to initialize a single channel to run in discrete input/transition count feature of the FQD function. When configured in this mode, the pin state will be updated when each transition is serviced to contain a value representing the latest pin level and the position count will count the number of transitions on the pin (positive and negative). The *tpu_fqd_data* function can be used to get the pin state. This function has 3 parameters:

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h
- channel - This is the channel number of the primary FQD channel.
- priority - This is the priority to assign to both channels. This parameter should be assigned a value of: TPU_PRIORITY_HIGH, TPU_PRIORITY_MIDDLE or TPU_PRIORITY_LOW. The TPU priorities are defined in mpc500_utils.h.

As described in *tpu_fqd_init*, it is best if the channel is disabled and not running before this initialization routine is called. When using a channel in this mode only the *tpu_fqd_position, tpu_fqd_get_position* and *tpu_fqd_data* functions should be used. The *tpu_fqd_data* function will return the current state of both of pins, however because only one channel is used in this mode, the state of the second channel will not return useful data.

2.3.5 void tpu_fqd_mode

This function is used to switch between normal and fast mode on the FQD channels. This function has 3 parameters:

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h
- channel - This is the channel number of the primary FQD channel.
- mode - This defines which mode to use; the value should be: TPU_FQD_NORMAL_MODE or TPU_FQD_FAST_MODE. The TPU modes are defined in tpu_fqd.h.

Normally a program will switch between the normal and fast modes as the speed of the input signal changes. An example of using this feature is shown in example program 2.

2.3.6 UINT8 tpu_fqd_current_mode

This function returns the current mode of the FQD channels. This function has 2 parameters:

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h
- channel - This is the channel number of the primary FQD channel. The next channel is used as the secondary.

The returned value will be: TPU_FQD_NORMAL_MODE or TPU_FQD_FAST_MODE. The TPU modes are defined in tpu_fqd.h.

2.3.7 INT16 tpu_fqd_position

This function returns the current position count of the FQD channels. This is the 16 bit counter that is the primary output of the FQD function. This function has 2 parameters:

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h
- channel - This is the channel number of the primary FQD channel.

The return value is the current position count value.

2.3.8 void tpu_fqd_data

This function returns the data parameters associated with the FQD channels. This is function can be used to get the current state of the pins or the data could be used to interpolate the position on very slow input signals.This function has 6 parameters:

- *tpu - This is a pointer to the TPU3 module to use. It is of type TPU3_tag which is defined in m_tpu3.h
- channel - This is the channel number of the primary FQD channel.
- *tcr1 - The current value of the TCR1 timebase.
- *edge - The last edge time.
- *primary_pin - The current state of the primary channel, this will be TPU_FQD_PIN_HIGH or TPU_FQD_PIN_LOW.
- *secondary_pin - The current state of the primary channel, this will be TPU_FQD_PIN_HIGH or TPU_FQD_PIN_LOW.

WARNING

In order for this function to return the current TCR1 value, it must request a TPU host service and wait for the service to be completed. If the TPU channel was accidentally disabled or the TPU is stuck in an endless loop then this function WILL NEVER RETURN.

The value 0x8000 is used to represent a pin high level, and 0x0000 to represent a pin low level.

3 Fast Quadrature Decode Examples

The following examples show configuration of the fast quadrature decode function for both quadrature decode and as an input pin with transition counter. Each example is a C program that shows how to configure and use the FQD interface routines.

3.1 Example 1

3.1.1 Description

This is a simple program to get a basic quadrature input signal system running. It configure channels 0 and 1 on TPU A to run FQD. The initial position should be 0x0000. The channel should be scheduled as high priority.

The program then runs in an infinite loop reading back the current position based on the quadrature signal on TPU A channels 0 and 1.

3.1.2 Program

```
/*****************************************************************************/
/* FILE NAME: tpu_fqd_exmaple1.c              COPYRIGHT (c)  2002 */
/* VERSION: 1.0                                   All Rights Reserved     */
/*                                                                           */
/* DESCRIPTION: This sample program show a simple example of a program    */
/* that updates of the global "position" variable from an external         */
/* quadrature input signal. The signal is connected to TPU A on channels  */
/* 0 and 1.                                                                  */
/* The program is targeted for the MPC555 but should work on any MPC500   */
/* device with a TPU. For other devices the setup routines will also need */
/* to be changed.                                                            */
/*===========================================================================*/
/* HISTORY           ORIGINAL AUTHOR: Jeff Loeliger                       */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                 */
/* ---   -----------  ---------    --------------------                  */
/* 1.0   J. Loeliger  03/Aug/02    Initial version of function.          */
/*****************************************************************************/

#include "mpc555.h"     /* Define all of the MPC555 registers, this needs to */
                        /* changed if other MPC500 devices are used.         */
#include "mpc500.c"     /* Configuration routines for MPC555 EVB, will need */
                        /* to be changed if other hardware is used.         */
#include "mpc500_util.h"    /* Utility routines for using MPC500 devices */
#indlue  "tpu_fqd.h"    /* TPU FQD functions */
#define ENCODER1 tpua,0 /* ENCODER1 is connected to TPU A channels 0 & 1 */

INT16 position;     /* global position value for quadrature input signal */

void main ()
{
```

```
    struct TPU3_tag *tpua = &TPU_A;   /* pointer for TPU routines */

    setup_mpc5xx(40);       /*Setup device and programm PLL to 40MHz*/

   /* Initialize quadrature input function with:   */
   /*    -Input signal on TPU A channel 0 and 1    */
   /*    -Initial count value of 0x0000              */
   /*    -Schedule as high priority in the TPU     */
   tpu_fqd_init( ENCODER1, TPU_PRIORITY_HIGH, 0x0000);

   while (1) {
        position = tpu_fqd_position(ENCODER1);
   }
}
```

3.2 Example 2

3.2.1 Description

The example program uses the mode feature to shift in and out of fast mode depending on the speed of the input signal. If the function is in normal mode and the number of counts exceeds FQD_MAX_DELTA_COUNT then the function shifts into fast mode. If the function is in fast mode and the number of counts is less than FQD_MIN_DELTA_COUNT then the function shifts into normal mode.

It configures channels 4 and 5 on TPU A to run FQD. The initial position should be 0x1000. The channel should be scheduled as high priority.

3.2.2 Program

```
/*************************************************************************/
/* FILE NAME: tpu_fqd_example2.c             COPYRIGHT (c)  2002 */
/* VERSION: 1.0                                  All Rights Reserved     */
/*                                                                       */
/* DESCRIPTION: This sample program show a simple example of a program    */
/* that updates of the global "position" variable from an external         */
/* quadrature input signal. The signal is connected to TPU A on channels  */
/* 4 and 5.                                                              */
/* The program switches between fast and normal. If the count during a    */
/* certain period is more than a maximum count and the function is in     */
/* normal mode then it shifts to fast mode. If the count during a certain */
/* period is less than a minumum count and the function is in fast mode   */
/* then it shifts to normal mode.                                        */
/* The program is targeted for the MPC555 but should work on any MPC500   */
```

Example 2

```
/* device with a TPU. For other devices the setup routines will also need */
/* to be changed.                                                          */
/*=========================================================================*/
/* HISTORY           ORIGINAL AUTHOR: Jeff Loeliger                        */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                  */
/* ---   -----------  ---------    --------------------                   */
/* 1.0   J. Loeliger  03/Aug/02    Initial version of function.           */
/*************************************************************************/

#include "mpc555.h"     /* Define all of the MPC555 registers, this needs to */
                        /* changed if other MPC500 devices are used.         */
#include "mpc500.c"     /* Configuration routines for MPC555 EVB, will need */
                        /* to be changed if other hardware is used.         */
#include "mpc500_util.h"    /* Utility routines for using MPC500 devices */
#include "tpu_fqd.h"    /* TPU FQD functions */

#define FQD_INIT_COUNT 0x1000
#define FQD_MIN_DELTA_COUNT 0x0100
#define FQD_MAX_DELTA_COUNT 0x7000

#define ENCODER1 tpua,4 /* ENCODER1 is connected to TPU A channels 0 & 1 */

INT16 position;     /* global position value for quadrature input signal */

void main ()
{
    INT32 delta;
    INT32 new_position;
    INT32 delay;
    UINT8  mode;

    struct TPU3_tag *tpua = &TPU_A;   /* pointer for TPU routines */

    setup_mpc500(40);       /*Setup device and program PLL to 40MHz*/

    /* Initialize quadrature input function */
    tpu_fqd_init(ENCODER1 , FQD_INIT_COUNT, TPU_PRIORITY_HIGH);

    position = tpu_fqd_position(ENCODER1);

    while (1) {
```

```
        new_position = tpu_fqd_position(ENCODER1);
        delta = new_position - position;
        position = new_position;

        mode = tpu_fqd_current_mode(ENCODER1);

        if ((delta>FQD_MAX_DELTA_COUNT) && (mode==TPU_FQD_NORMAL_MODE))
                tpu_fqd_mode(ENCODER1 , TPU_FQD_FAST_MODE);

        if ((delta<FQD_MIN_DELTA_COUNT) && (mode==TPU_FQD_FAST_MODE))
                tpu_fqd_mode(ENCODER1 , TPU_FQD_NORMAL_MODE);

        /* delay to simulate other tasks or calculations */
        for(delay=0; delay<10000 ; delay++);
    }
}
```

3.3 Example 3

3.3.1 Description

This program shows how to use the FQD function in the discrete input/transition count mode. A single channel is initialized and the number of input transition is monitored. The program also shows how to read back the other information from the FQD function.

Channel 12 is configured for discrete input/transition count mode and scheduled as low priority.

3.3.2 Program

```
/*********************************************************************/
/* FILE NAME: tpu_fqd_example3.c             COPYRIGHT (c)  2002 */
/* VERSION: 1.0                                  All Rights Reserved     */
/*                                                                       */
/* DESCRIPTION: This sample program show a simple example of a program    */
/* that uses the discrete input/transition counter feature of the FQD     */
/* function. The signal is connected to channel 12 on TPU B. The number of*/
/* transitions and other data is continually read from the channel.      */
/* The program is targeted for the MPC555 but should work on any MPC500   */
/* device with a TPU. For other devices the setup routines will also need */
/* to be changed.                                                        */
/*=====================================================================*/
/* HISTORY           ORIGINAL AUTHOR: Jeff Loeliger                      */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                 */
```

```
/* ---   -----------  ---------    --------------------                 */
/* 1.0   J. Loeliger  03/Aug/02    Initial version of function.         */
/************************************************************************/

#include "mpc555.h"     /* Define all of the MPC555 registers, this needs to */
                        /* changed if other MPC500 devices are used.         */
#include "mpc500.c"     /* Configuration routines for MPC555 EVB, will need */
                        /* to be changed if other hardware is used.         */
#include "mpc500_util.h"    /* Utility routines for using MPC500 devices */
#include "tpu_fqd.h"    /* TPU FQD functions */

INT16 count;     /* global transistion counter */

void main ()
{
     struct TPU3_tag *tpub = &TPU_B;   /* pointer for TPU routines */
     INT16 tcr1;          /* current TCR1 value */
     INT16 edge;          /* last edge time */
     INT16 primary_pin;   /* current pin state */
     INT16 junk;          /* this variable is not used and will return junk */

     setup_mpc5xx(40);       /*Setup device and program PLL to 40MHz*/

    /* Initialize transition count feature with:   */
    /*    -Input signal on TPU B channel 12         */
    /*    -Schedule as low priority in the TPU      */
    tpu_fqd_init_trans_count(tpub, 12, TPU_PRIORITY_LOW);

    while (1) {
         count = tpu_fqd_position (tpub, 12);

         tpu_fqd_data(tpub, 12, &tcr1, &edge, &primary_pin, &junk);
    }
}
```

3.4 Function State Timing

When calculating the worst case latency for the TPU the execution time of each state of the TPU is need.

The state timings for the FQD function are shown in Table 1. The states used by the C interface functions are shown in Table 2. States S3 and S4 are entered when there is a transition on the input pins.

Table 1. Fast Quadrature Decode Function—State Timing

State Number & Name	Max CPU Clock Cycles	RAM accesses by TPU
S1 INIT_FQD	12	3
S2 READ_TCR1_FQD	2	1
S3 EDGE_NORM_FQD i) Remain in Normal mode ii) Switch to Fast mode	 36 40	 8 8
S4 EDGE_FAST_FQD i) Remain in Fast mode ii) Switch to Normal mode	 16 26	 4 5

NOTE: Execution times do not include the time slot transition time (TST= 10 or 14 CPU clocks)

Table 2. Fast Quadrature Decode Function—State Uses

FQD Function	State Uses
tpu_fqd_init	S1
tpu_fqd_init_trans_count	S1
tpu_fqd_mode	None
tpu_fqd_current_mode	None
tpu_fqd_position	None
tpu_fqd_write_position	None
tpu_fqd_data	S2

3.5 Function Code Size

Total TPU function code size determines what combination of functions can fit into a given ROM or emulation DPTRAM memory microcode space. FQD function code size is:

38 µ instructions + 8 entries = 46 long words

4 Notes on Performance and Use of the FQD Function

4.1 Performance

Like all TPU functions, the performance limit of the FQD function in a given application is dependent on the service time (latency) of other active TPU channels. This is due to the operational nature of the scheduler. Where a pair of FQD channels are being used in normal mode and no other TPU channels are active, the minimum time between count edges on the two channels is 50 CPU clock cycles. This is equivalent to a count rate of approximately 780KHz with a system clock speed of 40MHz. In fast mode, the minimum time between rising edges on the primary channel is 30 CPU clock cycles. Since the counter is updated by four on each primary rising edge, this is equivalent to a count rate of approximately 5.2MHz with a system clock speed of 40MHz.

When more TPU channels are active, this performance will be degraded; e.g. if two sets of encoder signals are decoded using four channels then the maximum count rate in each mode would be limited to approximately 380KHz and 2.6MHz respectively. Use of other functions such as PWM will also have a degrading effect on this performance.

However the scheduler assures that the worst case latencies in any TPU application can be closely estimated, so it is recommended that the guidelines given in the TPU reference manual are used along with the figures given in the Fast Quadrature Decode state timing table to perform an analysis on any proposed TPU application that appears to approach the performance limits of the TPU.

4.2 Accuracy

Since the TPU is essentially a software machine that takes time to respond to an input transition, there will always be a +/- 1 lsb uncertainty in the value of the position count in normal mode while the input signals are active. In fast mode, this uncertainty is increased to +/- 4 lsb due to the 'by 4' update of the position count.

These 'uncertainties' only apply while the external system (e.g. a motor) is active, after the system has been brought to a stop with FQD in normal mode, and the last transition has been serviced, the position count will be accurate.

4.3 Noise Immunity

Features in the hardware of the TPU and the microcode of the FQD function protect, to a large extent, the counter from erroneous updates due to noise. All TPU input channels incorporate a digital filter which rejects pulses of less than a programmable duration.

In addition to this protection, when servicing a transition in normal mode, the FQD function always checks the new pinstate against the previous pinstate from the last service and if they are equal then no action is taken. This protects against a noise pulse that is long enough to get through the digital filter, but not long enough to last from the actual transition time to the time that the TPU services the channel.

In fast mode where only rising edges are serviced, no microcode noise immunity is provided. Despite these precautions, there may be situations where severe noise will result in erroneous updates of the counter e.g. noise on both channels simultaneously. Under these conditions it is recommended that additional external protection is added, such as schmitt trigger buffers and an additional analog or digital filter stage.

4.4 Using FQD with 3-Channel Encoders

Many shaft encoders supply three signals: two quadrature signals plus an index signal that generates a pulse once per revolution. This pulse usually has a fixed relationship to other system parameters and is used for alignment during startup.

These 3 signal encoders can be decoded when FQD is used in conjunction with another TPU function called “New Input Transition Counter (NITC).” FQD decodes the quadrature signals and the index pulse should be fed to the NITC channel. NITC allows any location in parameter RAM to be captured on a specified edge and the value presented to the CPU. In this case NITC would be configured to 'capture' the POSITION_COUNT parameter of FQD.

4.5 Listing 1

```
/*****************************************************************************/
/* FILE NAME: tpu_fqd.h                          COPYRIGHT (c) A 2002 */
/* VERSION: 1.0                                      All Rights Reserved     */
/*                                                                           */
/* DESCRIPTION: This file defines the interface to the TPU FQD functions  */
/* and provides useful #defines.                                             */
/*                                                                           */
/*===========================================================================*/
/* HISTORY           ORIGINAL AUTHOR: Jeff Loeliger                          */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                   */
/* ---   -----------  ---------    ---------------------                   */
/* 1.0   J. Loeliger  03/Aug/02    Initial version of function.            */
/*****************************************************************************/
#ifndef _TPU_FQD_H
#define _TPU_FQD_H

#include "m_common.h"
#include "m_tpu3.h"

/* Define HSR values */
#define TPU_FQD_INIT 0x3
#define TPU_FQD_READ_TCR1 0x2

/* Define HSQ values */
#define TPU_FQD_PRIMARY_CHANNEL 0x0
#define TPU_FQD_SECONDARY_CHANNEL 0x1
#define TPU_FQD_NORMAL_MODE 0x0
#define TPU_FQD_FAST_MODE 0x2

/* Define pin state */
#define TPU_FQD_PIN_HIGH 0x8000
#define TPU_FQD_PIN_LOW  0x0000

/* Define parameter RAM locations */
#define TPU_FQD_EDGE_TIME          0
#define TPU_FQD_POSITION_COUNT     1
#define TPU_FQD_TCR1_VALUE         2
#define TPU_FQD_CHAN_PINSTATE      3
#define TPU_FQD_CORR_PINSTATE_ADDR 4
```

```
#define TPU_FQD_EDGE_TIME_LSB_ADDR 5

/* TPU FQD function prototypes */
void tpu_fqd_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, \
                INT16 init_position);
void tpu_fqd_init_trans_count(struct TPU3_tag *tpu, UINT8 channel, \
                UINT8 priority);

void tpu_fqd_mode(struct TPU3_tag *tpu, UINT8 channel, UINT8 mode );
UINT8 tpu_fqd_current_mode(struct TPU3_tag *tpu, UINT8 channel);

INT16 tpu_fqd_position(struct TPU3_tag *tpu, UINT8 channel);

void tpu_fqd_data(struct TPU3_tag *tpu, UINT8 channel, INT16 *tcr1, \
                INT16 *edge, INT16 *primary_pin, INT16 *secondary_pin);

#endif /* ifndef _TPU_FQD_H  */

/*********************************************************************
```

4.6 Listing 2

```
/*********************************************************************/
/* FILE NAME: tpu_fqd.c                      COPYRIGHT (c)  2002 */
/* VERSION: 1.0                                  All Rights Reserved     */
/*                                                                       */
/* DESCRIPTION: This file contains the TPU FQD functions. These functions */
/* allow you to completely control TPU channels running the FQD function. */
/* They provide a simple interface requiring the minimum amount of        */
/* configuration by the user.                                            */
/*=======================================================================*/
/* HISTORY           ORIGINAL AUTHOR: Jeff Loeliger                      */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                 */
/* ---   -----------  ---------    --------------------                  */
/* 1.0   J. Loeliger  03/Aug/02    Initial version of function.          */
/*********************************************************************/
#include "tpu_fqd.h"
#include "mpc500_util.h"
```

Listing 2

```
/*************************************************************************
FUNCTION      : tpu_fqd_init
PURPOSE       : To initialize a pair of channels to run the FQD function.
INPUTS NOTES  : This function has 4 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                        type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the channel number of the primary FQD
                           channel. The next channel is used as the secondary.
                 priority - This is the priority to assign to both channels.
                            This parameter should be assigned a value of:
                            TPU_PRIORITY_HIGH, TPU_PRIORITY_MIDDLE or
                            TPU_PRIORITY_LOW.
                 init_position - This is the starting position.
RETURNS NOTES : none
WARNING       : The channels must be stopped before it is reconfigured. The
                function disables the channels but if they were currently
                being serviced it would continue. The delay for assigning the
                pram pointer may to enough but depends on system loading.
*************************************************************************/
void tpu_fqd_init(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, \
                INT16 init_position)
{
    struct TPU_param_tag *pram;
    UINT8 channel2;

    /* if primary channel is 15 then secondary channel should be 0 */
    channel2 = (channel + 1) & 0xF;

    /* diable channels so they can be configured safely */
    tpu_disable( tpu, channel);
    tpu_disable( tpu, channel2);

    /* select FQD function for both channels */
    tpu_func( tpu, channel, TPU_FUNCTION_FQD);
    tpu_func( tpu, channel2, TPU_FUNCTION_FQD);

    /* Initialize parameter RAM                                          */
    /* -setup initial count in POSITION_COUNT                            */
    /* -initialize CORR_PINSTATE_ADDR and EDGE_TIME_LSB_ADDR in both channels*/
    tpu->PARM.R[channel][TPU_FQD_POSITION_COUNT] = init_position;
    tpu->PARM.R[channel][TPU_FQD_CORR_PINSTATE_ADDR] = (INT16) (((channel2) << 4) + 6);
```

Listing 2

```
    tpu->PARM.R[channel][TPU_FQD_EDGE_TIME_LSB_ADDR] = (INT16) ((channel << 4) + 1);
    tpu->PARM.R[channel2][TPU_FQD_CORR_PINSTATE_ADDR] = (INT16) ((channel << 4) + 6);
    tpu->PARM.R[channel2][TPU_FQD_EDGE_TIME_LSB_ADDR] = (INT16) ((channel << 4) + 1);

    /* Configure the first channel as the primary channel and the following */
    /* channel as the secondary channel.                                    */
    tpu_hsq(tpu, channel, TPU_FQD_PRIMARY_CHANNEL | TPU_FQD_NORMAL_MODE);
    tpu_hsq(tpu, channel2, TPU_FQD_SECONDARY_CHANNEL | TPU_FQD_NORMAL_MODE);

    /* Initialize both channels */
    tpu_hsr(tpu, channel, TPU_FQD_INIT);
    tpu_hsr(tpu, channel2, TPU_FQD_INIT);

    /* Enable channels by assigning a priority to them. */
    /* Both channels MUST have the same priority.       */
    tpu_enable(tpu, channel, priority);
    tpu_enable(tpu, channel2, priority);
}

/*****************************************************************************
FUNCTION      : tpu_fqd_init_trans_count
PURPOSE       : To initialize one channel as a discrete input/transition
                counter.
INPUTS NOTES  : This function has 3 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                        type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the channel number of the primary FQD
                           channel. The next channel is used as the secondary.
                 priority - This is the priority to assign to both channels
                            This parameter should be assigned a value of:
                            TPU_PRIORITY_HIGH, TPU_PRIORITY_MIDDLE or
                            TPU_PRIORITY_LOW.
RETURNS NOTES : none
WARNING       : The channel must be stopped before it is reconfigured. The
                function disables the channel but if it is currently
                being serviced it would continue. The delay for assigning the
                pram pointer may to enough but depends on system loading.
*****************************************************************************/
void tpu_fqd_init_trans_count(struct TPU3_tag *tpu, UINT8 channel, \
                UINT8 priority)
{
```

Listing 2

```
    struct TPU_param_tag *pram;

    /* diable channel so it can be configured safely */
    tpu_disable( tpu, channel);

    /* select FQD function for both channels */
    tpu_func( tpu, channel, TPU_FUNCTION_FQD);

    /* Initialize parameter RAM                              */
    /* -clear POSITION_COUNT to count transitions            */
    /* -initialize CORR_PINSTATE_ADDR and EDGE_TIME_LSB_ADDR */
    tpu->PARM.R[channel][TPU_FQD_POSITION_COUNT] = 0;
    tpu->PARM.R[channel][TPU_FQD_CORR_PINSTATE_ADDR] = (INT16) ((channel << 4) + 6);
    tpu->PARM.R[channel][TPU_FQD_EDGE_TIME_LSB_ADDR] = (INT16) ((channel << 4) + 1);

    /* Configure the channel as a primary channel */
    tpu_hsq(tpu, channel, TPU_FQD_PRIMARY_CHANNEL | TPU_FQD_NORMAL_MODE);

    /* Initialize the channel */
    tpu_hsr(tpu, channel, TPU_FQD_INIT);

    /* Enable the channel by assigning a priority to it. */
    tpu_enable(tpu, channel, priority);
}

/*****************************************************************************
FUNCTION      : tpu_fqd_mode
PURPOSE       : To change between fast and normal modes.
INPUTS NOTES  : This function has 3 parameters:
                *tpu - This is a pointer to the TPU3 module to use. It is of
                       type TPU3_tag which is defined in m_tpu3.h
                channel - This is the channel number of the primary FQD
                          channel. The next channel is used as the secondary.
                mode - The defines which mode to use, the value should be:
                TPU_FQD_NORMAL_MODE or TPU_FQD_FAST_MODE.
RETURNS NOTES : none
*****************************************************************************/
void tpu_fqd_mode(struct TPU3_tag *tpu, UINT8 channel, UINT8 mode )
{
    tpu_hsq(tpu, channel, TPU_FQD_PRIMARY_CHANNEL | mode);
}
```

Listing 2

```
/*************************************************************************
FUNCTION      : tpu_fqd_current_mode
PURPOSE       : To determine the current operating mode of the FQD function.
INPUTS NOTES  : This function has 2 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                        type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the channel number of the primary FQD
                           channel. The next channel is used as the secondary.
RETURNS NOTES : The operating mode which will be TPU_FQD_NORMAL_MODE or
                TPU_FQD_FAST_MODE.
*************************************************************************/
UINT8 tpu_fqd_current_mode(struct TPU3_tag *tpu, UINT8 channel)
{
    return(tpu_get_hsq(tpu, channel));
}

/*************************************************************************
FUNCTION      : tpu_fqd_position
PURPOSE       : This function returns the current postion count for the
                quqdrature input signal.
INPUTS NOTES  : This function has 2 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                        type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the channel number of the primary FQD
                           channel. The next channel is used as the secondary.
RETURNS NOTES : The current position count.
*************************************************************************/
INT16 tpu_fqd_position(struct TPU3_tag *tpu, UINT8 channel)
{
     return (tpu->PARM.R[channel][TPU_FQD_POSITION_COUNT]);
}
/*************************************************************************
FUNCTION      : tpu_fqd_data
PURPOSE       : To get the current TCR1 value and the time of the last edge.
INPUTS NOTES  : This function has 6 parameters:
                 *tpu - This is a pointer to the TPU3 module to use. It is of
                        type TPU3_tag which is defined in m_tpu3.h
                 channel - This is the channel number of the primary FQD
                           channel. The next channel is used as the secondary.
                 *tcr1 - The current Value of the TCR1 timebase.
```

Listing 2

```
                *edge - The last edge time.
                *primary_pin - The current state of the primary channel, this
                               will be TPU_FQD_PIN_HIGH or TPU_FQD_PIN_LOW.
                *secondary_pin - The current state of the primary channel, this
                                 will be TPU_FQD_PIN_HIGH or TPU_FQD_PIN_LOW.
RETURNS NOTES : none
**WARNING**   : If the channel is disabled or the TPU is not responding this
                function will NEVER EXIT!
*************************************************************************/
void tpu_fqd_data(struct TPU3_tag *tpu, UINT8 channel, INT16 *tcr1, \
                INT16 *edge, INT16 *primary_pin, INT16 *secondary_pin)
{
    UINT8 channel2;

    /* if primary channel is 15 then secondary channel should be 0 */
    channel2 = (channel + 1) & 0xF;

     /* wait until the TPU channel has no pending HSRs */
     /* WARNING if the TPU does on service this request then this loop will */
     /* NEVER EXIT!                                                         */
     tpu_ready(tpu, channel);

     /* issue request to update TCR1 value */
     tpu_hsr(tpu, channel, TPU_FQD_READ_TCR1);

     /* read parameters */
     *edge = tpu->PARM.R[channel][TPU_FQD_EDGE_TIME];
     *primary_pin=tpu->PARM.R[channel][TPU_FQD_CHAN_PINSTATE];
     *secondary_pin=tpu->PARM.R[channel2][TPU_FQD_CHAN_PINSTATE];

     /* wait until TCR1 value has been updated */
     /* WARNING if the TPU does on service this request then this loop will */
     /* NEVER EXIT!                                                         */
     tpu_ready(tpu, channel);

     *tcr1 = tpu->PARM.R[channel][TPU_FQD_TCR1_VALUE];
}
/*********************************************************************
```

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor