Freescale Semiconductor
Application Note

AN2366
Rev. 1, 08/2004

Using the New Input Transition / Input Capture TPU Function (NITC) with the MPC500 Family

by: Vernon Goler
32-bit Embedded Controller Division

This TPU Programming Note is intended to provide simple C interface routines to the new input transition/input capture TPU function (NITC).[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.

Table of Contents

1 Functional Overview 1
2 Detailed Description 2
2.1 NITC Routines 2
3 New Input Transition / Input Capture Examples 9
3.1 Example 1 9
3.2 Example 2 12
3.3 Function State Timing 14
3.4 Function Code Size 15
4 Notes on Use and Performance of the NITC Function 15
4.1 Performance 15
4.2 Noise Immunity 15
5 Revision History 16

1 Functional Overview

The NITC function can detect rising and/or falling input transitions. When a transition is detected, a value is captured. The value can be either a free-running incrementing counter value, known as the current Timer Count Register (TCR), or a parameter RAM value. The channel continues to detect and count input transitions until it has counted a programmable number of transitions. The NITC function can count the programmed maximum number of transitions once, ceasing channel activity until reinitialized, or the channel may perform this operation continuously. When the programmed number of transitions is counted, the

1. The information in this Programming Note is based on TPUPN02 and TPUPN08. It is intended to compliment the information found in those Programming Notes.

function can send an interrupt request to the host CPU and/or generate a link request message to a sequential block of up to eight TPU channels. A link request is a message from one TPU channel to another TPU channel requesting service for the channel(s) receiving the link. The user specifies the starting channel of the sequential block and the number of channels to receive a link within the block.

2 Detailed Description

Any channel of the TPU can perform an input capture by detecting either a rising edge, falling edge, or both edges of an input transition. Performing an input capture means to record the TCR value, or a parameter RAM value when an input transition occurs. The TPU is initialized differently depending on which value, a TCR value or a parameter RAM value is to be captured when a transition is detected. The function *tpu_nitc_init_tcr_mode* is used to initialize the TPU channel for TCR capture, and the function *tpu_nitc_init_parameter_mode* is used to initialize the TPU channel for parameter RAM capture. Any channel of the TPU can count several input transitions with the maximum number of transitions to be counted specified in the initialization routines or by the function *tpu_nitc_write_max_count.* An initial count of transitions detected can be set to some initial value by the function *tpu_nitc_write_trans_count.* The TPU services each input transition detected by saving a TCR value or a parameter RAM value. The function *tpu_nitc_read_last_trans_time* can be used to read the TCR value or parameter RAM value TPU_INTC_LAST_TRANS_TIME for each transition detected when the transition count is less than the parameter TPU_INTC_MAX_COUNT. The function *tpu_nitc_read_final_trans_time* can be used to read the TCR value or parameter RAM value TPU_INTC_FINAL_TRANS_TIME for the final transition when the transition count is equal to parameter TPU_INTC_MAX_COUNT.

A TPU channel executing the NITC function can be initialized to perform the count operation once, or continuously. In both initialization routines, the TPU_INTC_TRANS_COUNT parameter is cleared to zero. A count operation refers to counting transitions until the maximum programmed number of transitions (as specified by parameter TPU_INTC_MAX_COUNT) is counted. When a count operation is complete, the TPU channel can be configured to generate an interrupt, and/or link to a sequential block of up to eight TPU channels. A TPU channel configured for a single count operation will upon the completion of each count operation cease channel activity until reinitialized. A TPU channel configured for continuous count operation will, upon the completion of each count operation, clear the transition count parameter TPU_INTC_TRANS_COUNT, and start a new count operation.

The parameter capture capability of the *tpu_nitc_init_parameter_mode* is very useful when working with a quadrature encoder that has three outputs. Encoders with three signals have two quadrature signals and an index signal. The quadrature signals are connected to a quadrature decode function like FQD or QDEC and the index signal is connected a channel running NITC. The NITC channel is configured to capture the POSITION_COUNT parameter of the quadrature decode function. The NITC channel must be run on a lower channel number than the quadrature function channel, and assigned the same priority.

2.1 NITC Routines

Rather than controlling the TPU registers directly, the NITC routines in this TPU programming note may be used to provide a simple and easy interface to the user’s application. There are 8 routines for controlling the NITC function in 2 files (tpu_nitc.h and tpu_nitc.c). The tpu_nitc.h file should be included in any files

that use the routines. This file contains the function prototypes and useful #defines. Each of the following routines in tpu_nitc.c will be examined in detail in the sections below:

- void tpu_nitc_init_tcr_mode(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, UINT8 detect_edge, INT16 max_count, UINT8 single_continuous_operation, UINT8 tcr, UINT8 nolink_link, UINT8 start_link_channel, UINT8 link_channel_count, UINT8 nointerrupt_interrupt);
- void tpu_nitc_init_parameter_mode(struct TPU3_tag *tpu, UINT8 channel, UINT8 priority, UINT8 detect_edge, INT16 max_count, UINT8 single_continuous_operation, UINT8 parameter_address, UINT8 nolink_link, UINT8 start_link_channel, UINT8 link_channel_count, UINT8 nointerrupt_interrupt);
- void tpu_nitc_write_max_count(struct TPU3_tag *tpu, UINT8 channel, INT16 max_count);
- void tpu_nitc_write_trans_count(struct TPU3_tag *tpu, INT16 trans_count);
- INT16 tpu_nitc_read_final_trans_time(struct TPU3_tag *tpu, UINT8 channel);
- INT16 tpu_nitc_read_last_trans_time(struct TPU3_tag *tpu, UINT8 channel);
- INT16 tpu_nitc_read_max_count(struct TPU3_tag *tpu, UINT8 channel);
- INT16 tpu_nitc_read_trans_count(struct TPU3_tag *tpu, UINT8 channel);

2.1.1 void tpu_nitc_init_tcr_mode

This function is used to initialize a TPU channel to run the NITC function in TCR capture mode. This function has 11 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 channel that will perform the NITC function.
- priority - This is the channel priority. This parameter should be assigned a value of:
  - TPU_PRIORITY_HIGH
  - TPU_PRIORITY_MIDDLE
  - TPU_PRIORITY_LOW

  The TPU priorities are defined in mpc500_utils.h.
- detect_edge - This is the transition edge that will be detected. The edge detected is either rising, falling, or both. This parameter should be assigned a value of:
  - TPU_NITC_RISING
  - TPU_NITC_FALLING
  - TPU_NITC_RISING_FALLING

  Parameters specific to the NITC function are defined in tpu_nitc.h.
- max_count – This defines the maximum number of transitions to be detected. When max_count transitions have been detected, an interrupt may be generated, and/or a message sent to other channels via a link mechanism.

- Single_continuous_operation – This defines whether the channel performs one count operation and then ceases operation until reinitialized, or performs the count operation continuously. This parameter should be assigned a value of:
  - TPU_NITC_SINGLE
  - TPU_NITC_CONTINUOUS
- tcr – Determines which TPU timebase TCR1 or TCR2 to capture when the specified transition is detected. This parameter should be assigned a value of:
  - TPU_NITC_TCR1
  - TPU_NITC_TCR2
- nolink_link – This determines whether a message is sent to other channels by a link mechanism when the maximum number of transitions have been detected. This parameter should be assigned a value of:
  - TPU_NITC_NOLINK
  - TPU_NITC_LINK
- start_link_channel – This is the channel where linking begins. If linking is not enabled, then this parameter is a don’t care and can be assigned any of the start link channel values shown below. This parameter should be assigned a value of:
  - TPU_NITC_START_LINK_CHANNEL_0
  - TPU_NITC_START_LINK_CHANNEL_1
  - TPU_NITC_START_LINK_CHANNEL_2
  - TPU_NITC_START_LINK_CHANNEL_3
  - TPU_NITC_START_LINK_CHANNEL_4
  - TPU_NITC_START_LINK_CHANNEL_5
  - TPU_NITC_START_LINK_CHANNEL_6
  - TPU_NITC_START_LINK_CHANNEL_7
  - TPU_NITC_START_LINK_CHANNEL_8
  - TPU_NITC_START_LINK_CHANNEL_9
  - TPU_NITC_START_LINK_CHANNEL_10
  - TPU_NITC_START_LINK_CHANNEL_11
  - TPU_NITC_START_LINK_CHANNEL_12
  - TPU_NITC_START_LINK_CHANNEL_13
  - TPU_NITC_START_LINK_CHANNEL_14
  - TPU_NITC_START_LINK_CHANNEL_15
- link_channel_count - This is the number of sequential channels to link to starting from the start link channel. The maximum number of channels that can be linked is eight channels. If linking is not enabled, then this parameter is a don’t care and can be assigned any of the start link channel values shown below. This parameter should be assigned a value of:

  - TPU_NITC_LINK_ONE
  - TPU_NITC_LINK_TWO
  - TPU_NITC_LINK_THREE
  - TPU_NITC_LINK_FOUR
  - TPU_NITC_LINK_FIVE
  - TPU_NITC_LINK_SIX
  - TPU_NITC_LINK_SEVEN
  - TPU_NITC_LINK_EIGHT
- nointerrupt_interrupt – This determines whether an interrupt is generated when the programmable maximum number of transitions are detected. This parameter should be assigned a value of:
  - TPU_NITC_NOINTERRUPT
  - TPU_NITC_INTERRUPT

NOTE

Care should be taken when initializing TPU channels. The TPU's behavior may become unpredictable if a channel is reinitialized while it is running. This unpredictability can occur because there is no way to stop a TPU channel that is executing code. Therefore, the channel must complete the execution of code before it is reinitialized. To ensure that the channel is stopped before it is configured, the channel's priority should be set to disabled. If the channel is currently being serviced when the priority is set to disabled, it will continue to service the channel until the state ends. To ensure that the channel is not being serviced, the user should wait for the longest state execution time after disabling the channel. All channels are disabled out of reset so that they can be configured immediately from reset.

The *tpu_nitc_init_tcr_mode* function attempts to wait between the disabling of the channel before it starts configuring it; however, the actual execution speed of the code will be depend on the specific system. If the user is not configuring the channels from reset, then ideally it is best to have the channel 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);
```

2.1.2 void tpu_nitc_init_parameter_mode

This function is used to initialize a TPU channel to run the NITC function in parameter capture mode. This function has 11 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 channel that will perform the NITC function.

- priority - This is the channel priority. This parameter should be assigned a value of:
  - TPU_PRIORITY_HIGH
  - TPU_PRIORITY_MIDDLE
  - TPU_PRIORITY_LOW

  The TPU priorities are defined in mpc500_utils.h.
- detect_edge - This is the transition edge that will be detected. The edge detected is either rising, falling, or both. This parameter should be assigned a value of:
  - TPU_NITC_RISING
  - TPU_NITC_FALLING
  - TPU_NITC_RISING_FALLING

  Parameters specific to the NITC function are defined in tpu_nitc.h.
- max_count – This defines the maximum number of transitions to be detected. When max_count transitions have been detected, an interrupt may be generated and/or a message sent to other channels via a link mechanism.
- Single_continuous_operation – This defines whether the channel performs one count operation and then ceases operation until reinitialized, or performs the count operation continuously. This parameter should be assigned a value of:
  - TPU_NITC_SINGLE
  - TPU_NITC_CONTINUOUS
- parameter_address – This defines the address of the TPU parameter to be captured. The value pointed to by this parameter is captured into one of the two following parameter locations:
  - TPU_NITC_FINAL_TRANS_TIME
  - TPU_NITC_LAST_TRANS_TIME

  depending on whether max_count transitions have been detected or fewer than max_count transitions have been detected. If less than the max_count number of transitions have been detected, the value pointed to by parameter_address is placed in parameter location

  TPU_NITC_LAST_TRANS_TIME. If the max_count number of transitions have been detected, the value pointed to by the parameter_address is placed in parameter location TPU_NITC_FINAL_TRANS_TIME. The parameter_address value needs to be aligned to a half word (2-byte) boundary.
- nolink_link – This determines whether a message is sent to other channels by a link mechanism when the maximum number of transitions have been detected. This parameter should be assigned a value of:
  - TPU_NITC_NOLINK
  - TPU_NITC_LINK
- start_link_channel – This is the channel where linking begins. If linking is not enabled, then this parameter is a don’t care and can be assigned any of the start link channel values shown below. This parameter should be assigned a value of:

  - TPU_NITC_START_LINK_CHANNEL_0
  - TPU_NITC_START_LINK_CHANNEL_1
  - TPU_NITC_START_LINK_CHANNEL_2
  - TPU_NITC_START_LINK_CHANNEL_3
  - TPU_NITC_START_LINK_CHANNEL_4
  - TPU_NITC_START_LINK_CHANNEL_5
  - TPU_NITC_START_LINK_CHANNEL_6
  - TPU_NITC_START_LINK_CHANNEL_7
  - TPU_NITC_START_LINK_CHANNEL_8
  - TPU_NITC_START_LINK_CHANNEL_9
  - TPU_NITC_START_LINK_CHANNEL_10
  - TPU_NITC_START_LINK_CHANNEL_11
  - TPU_NITC_START_LINK_CHANNEL_12
  - TPU_NITC_START_LINK_CHANNEL_13
  - TPU_NITC_START_LINK_CHANNEL_14
  - TPU_NITC_START_LINK_CHANNEL_15
- link_channel_count - This is the number of sequential channels to link to starting from the start link channel. The maximum number of channels that can be linked is eight channels. If linking is not enabled, then this paramter is a don’t care and can be assigned any of the start link channel values shown below. This parameter should be assigned a value of:
  - TPU_NITC_LINK_ONE
  - TPU_NITC_LINK_TWO
  - TPU_NITC_LINK_THREE
  - TPU_NITC_LINK_FOUR
  - TPU_NITC_LINK_FIVE
  - TPU_NITC_LINK_SIX
  - TPU_NITC_LINK_SEVEN
  - TPU_NITC_LINK_EIGHT
- nointerrupt_interrupt – This determines whether an interrupt is generated when the programmable maximum number of transitions are detected. This parameter should be assigned a value of:
  - TPU_NITC_NOINTERRUPT
  - TPU_NITC_INTERRUPT

As described in *tpu_nitc_init_tcr_mode,* it is best if the channel is disabled and not running before the initialization routine is called.

2.1.3 void tpu_nitc_write_max_count

This function is used to write the max_count value. 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 channel that will perform the NITC function.
- max_count – This defines the maximum number of transitions to be detected. When max_count transitions have been detected an interrupt may be generated, and/or a message sent to other channels via a link. When max_count transitions have been detected one count operation is complete.

2.1.4 void tpu_nitc_write_trans_count

This function is used to write the trans_count value. 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 channel that will perform the NITC function.
- trans_count – This is the current count of transitions that have been detected.

The trans_count value can be initialized to some intermediate count value less than max_count after initialization is run. During initialization the trans_count value is always cleared to zero.

2.1.5 INT16 tpu_nitc_read_final_trans_time

This function returns the final transition time in TCR mode, or the final value pointed to in parameter mode. 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 channel that will perform the NITC function.

The returned value is the final transition time or the final value pointed to by the parameter address.

2.1.6 INT16 tpu_nitc_read_last_trans_time

This function returns the last transition time in TCR mode, or the last value pointed to in parameter mode. 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 channel that will perform the NITC function.

The returned value is the last transition time or the last value pointed to by the parameter address.

2.1.7 INT16 tpu_nitc_read_max_count

This function returns the max_count value. 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 channel that will perform the NITC function.

The returned value is the max count value.

2.1.8 INT16 tpu_nitc_read_trans_count

This function returns the trans_count value. 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 channel that will perform the NITC function.

The returned value is the transition count value which is the current number of transitions that have been counted. This value will always be less than or equal to max_count.

3 New Input Transition / Input Capture Examples

The following examples show configuration new input transition/ input capture. Each example is a C program that shows how to configure and use the NITC interface routines.

3.1 Example 1

3.1.1 Description

This is a simple program to initialize a TPU channel to perform the NITC function in parameter capture mode. The TPU channel is configured to detect rising edge transitions. TPU parameter address 0x32 is the TPU address that the NITC channel will capture data from when a rising edge transition is detected. The TPU channel is configured to perform the count operation once and then stop. When the maximum number of transitions corresponding to MAX_COUNT have been counted the TPU channel will link to six sequential channels starting with channel number five. The interrupt status flag on channel 0 is polled to determine when MAX_COUNT, TRANS_COUNT, FINAL_TRANS_TIME, and LAST_TRANS_TIME parameters are read and stored in local variables.

3.1.2 Program

```
/*************************************************************************/
/* FILE NAME: tpu_nitc_ex1.c              COPYRIGHT (c) FREESCALE 2002     */
/* VERSION: 1.0                                    All Rights Reserved     */
/*                                                                         */
```

```
/* DESCRIPTION: This routine is used to initialize TPU channel 0 to run   */
/* the NITC function in parameter capture mode.                           */
/*                                                                        */
/* This 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: Vernon Goler                        */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                  */
/* ---   -----------  ---------    --------------------                   */
/* 1.0   V. Goler           28/Aug/02    Initial version of function.     */
/*************************************************************************/

#include "mpc555.h"     /* Define all of the MPC555 registers, this needs to  */
                        /* be 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_nitc.h"   /* TPU NITC functions                              */

void main ()
{

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

INT16 max_count_nitc;           /* max count value                     */
INT16 trans_count_nitc;         /* number of transistions counted      */
INT16 final_param_capture_nitc; /* value captured, max count reached   */
INT16 last_param_capture_nitc;  /* value captured, < max counts        */

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

```
        /* Initialize for parameter capture mode with:                  */
        /*      - Input signal on TPU A channel 0                       */
        /*      - Priority is high                                      */
        /*      - Count to 10,000       decimal                         */
        /*      - Count rising edges                                    */
        /*      - Count up to 10,000 once                               */
        /*      - The value pointed to by TPU address 0x32 is the value captured*/
        /*      - Link to 6 channels starting with chan 5 when max count reached*/
        /*      - Interrupts are not enabled                            */
        tpu_nitc_init_parameter_mode(tpua, 0, TPU_PRIORITY_HIGH, \
        TPU_NITC_RISING, 10000, TPU_NITC_SINGLE, 0x32, TPU_NITC_LINK, \
        TPU_NITC_START_LINK_CHANNEL_5, TPU_NITC_LINK_SIX, TPU_NITC_NOINTERRUPT);

        /* wait for max_count transitions to be counted                 */
        while((tpu_check_interrupt(tpua, 0)) != 1)
                ;

        /* clear interrupt status flag for channel 0                    */
        tpu_clear_interrupt(tpua, 0);

        /* Read all parameters                                          */
        max_count_nitc = tpu_nitc_read_max_count(tpua, 0);
        trans_count_nitc = tpu_nitc_read_trans_count(tpua, 0);
        final_param_capture_nitc = tpu_nitc_read_final_trans_time(tpua, 0);
        last_param_capture_nitc = tpu_nitc_read_last_trans_time(tpua, 0);

        while(1);
}
```

3.2 Example 2

3.2.1 Description

This is a simple program to initialize a TPU channel to perform the NITC function in tcr capture mode. The TPU channel is configured to detect falling edge transitions. The value of TCR1 is captured when each falling edge transition is detected. The TPU channel is configured to perform the count operation once and then stop. When the maximum number of transitions corresponding to MAX_COUNT have been counted, the TPU channel will link to 5 sequential channels starting with channel number one. The interrupt status flag on channel 0 is polled to determine when MAX_COUNT transitions have been counted. WHen the interrupt status flag is set, MAX_COUNT, TRANS_COUNT, FINAL_TRANS_TIME, and LAST_TRANS_TIME parameters are read and stored in local variables.

3.2.2 Program

```
/*************************************************************************/
/* FILE NAME: tpu_nitc_ex2.c              COPYRIGHT (c) FREESCALE 2002   */
/* VERSION: 1.0                                    All Rights Reserved    */
/*                                                                        */
/* DESCRIPTION: This routine is used to initialize TPU channel 0 to run   */
/* the NITC function in TCR capture mode.                                 */
/*                                                                        */
/* This 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: Vernon Goler                        */
/* REV      AUTHOR      DATE       DESCRIPTION OF CHANGE                  */
/* ---   -----------  ---------    ---------------------                  */
/* 1.0   V. Goler    28/Aug/02    Initial version of function.          */
/*************************************************************************/

#include "mpc555.h"     /* Define all of the MPC555 registers, this needs to  */
                        /* be 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_nitc.h"   /* TPU NITC functions   */

void main ()
{

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

INT16 max_count_nitc;   /* max count value            */
INT16 trans_count_nitc; /* number of transistions counted      */
INT16 final_tcr1_capture_nitc;/* value captured, max count reached      */
INT16 last_tcr1_capture_nitc;      /* value captured, < max count reached    */

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



        /* Initialize for parameter capture mode with:*/
        /*      - Input signal on TPU A channel 0              */
        /*      - Priority is high                                      */
        /*      - Count to 59 decimal                                   */
        /*      - Count falling edges                                   */
        /*      - Count up to 59    once                                */
        /*      - TCR1 is captured                                      */
        /*      - Link to 5 channels starting with chan 1 when max count reached*/
        /*      - Interrupts are not enabled                            */
        tpu_nitc_init_tcr_mode(tpua, 0, TPU_PRIORITY_HIGH, \
        TPU_NITC_FALLING, 59, TPU_NITC_SINGLE, TPU_NITC_TCR1, TPU_NITC_LINK, \
        TPU_NITC_START_LINK_CHANNEL_1, TPU_NITC_LINK_FIVE, TPU_NITC_NOINTERRUPT);

        /* wait for max_count transitions to be counted*/
        while((tpu_check_interrupt(tpua, 0)) != 1)
```

```
                ;


        /* clear interrupt status flag for channel 0 */
        tpu_clear_interrupt(tpua, 0);


        /* Read all parameters                                          */
        max_count_nitc = tpu_nitc_read_max_count(tpua, 0);
        trans_count_nitc = tpu_nitc_read_trans_count(tpua, 0);
        final_tcr1_capture_nitc = tpu_nitc_read_final_trans_time(tpua, 0);
        last_tcr1_capture_nitc = tpu_nitc_read_last_trans_time(tpua, 0);


        while(1);
}
```

3.3 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 NITC 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
S0 INIT_NITC_TCR_MODE	8	2
S1 INIT_NITC_PARAMETER_MODE	6	2
S2 Count _Up (last count)		
Single, no links	32	5
Continuous, no links	30	6
Single, links	32	6
Continuous, links	32[1]	7
All modes (not last count)	24	5

NOTES:
1 Assumes no channels linked. Add two clocks for each channel linked.

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

Table 2. NITC Function State Usage

NITC Function	State Uses
tpu_nitc_init_tcr_mode	S0
tpu_nitc_init_parameter_mode	S1
tpu_nitc_write_max_count	None
tpu_nitc_write_trans_count	None
tpu_nitc_read_final_trans_time	None
tpu_nitc_read_last_trans_time	None
tpu_nitc_read_max_count	None
Tpu_nitc_read_trans_count	None

3.4 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. NITC function code size is:

27 µ instructions + 8 entries = 35 long words

4 Notes on Use and Performance of the NITC Function

4.1 Performance

Like all TPU functions, the performance limit of the NITC 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.

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 New Input Capture/ Input Transition Counter state timing table to perform an analysis on any proposed TPU application that appears to approach the performance limits of the TPU.

4.2 Noise Immunity

Features in the hardware of the TPU and the microcode of the NITC 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. If greater noise immunity is desired additional external protection can be added, such as a schmitt trigger buffer and an additional analog or digital filter stage.

5 Revision History

Table 3 provides revision history details for this document.

Table 3. Revision History

Revision Number	Date	Description
0	10/2002	Initial release
1	08/2004	• Replaced NITC example 2 with the correct code (see 3.2.2). • Updated template to Freescale's brand strategy. • Replaced the name "Motorola" with "Freescale." • Removed '/D' from the document order number to reflect updated ordering process.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.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 480-768-2130

Europe, Middle East, and Africa:
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-0047, Japan
0120-191014 or +81-3-3440-3569

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
852-26668334

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

AN2366
Rev. 1
08/2004