

# New Input Capture/Input Transition Counter TPU Function (NITC)

By Jeff Loeliger and Jeff Wright

## 1 Functional Overview

The NITC function can detect rising and/or falling input transitions. When a transition is detected, the current TCR timer value or a parameter RAM value is captured. The channel continues to detect and count input transitions until it has counted the maximum programmable number stored in the parameter MAX\_COUNT. The NITC function can count the programmed maximum number of transitions continually, or it can count the programmed number of transitions once, then cease channel activity until reinitialized. Once the programmed number of transitions is counted, the function can send an interrupt request to the host CPU and generate a link to a sequential block of up to eight channels. A link is simply a message from one channel to another. The user specifies the starting channel of the block and the number of channels within the block.

The NITC function is very similar to the ITC function. The BANK\_ADDRESS update feature of the ITC function, which is used in conjunction with the PMA/PMM functions has been removed. The NITC PARAM\_ADDR parameter replaces the ITC BANK\_ADDRESS parameter. When NITC is used in parameter mode (HSR %10), it captures the value of a parameter RAM address pointed to by PARAM\_ADDR instead of a TCR value.

## 2 Detailed Description

Any channel of the TPU can perform an input capture. Performing an input capture means to record the TCR value or a parameter RAM value when one input transition occurs. If only one input capture is desired, then the number of transitions to be counted should be set to zero or one in MAX\_COUNT. Any channel of the TPU can count several input captures as specified by the parameter MAX\_COUNT.

The TPU services each input capture by saving the transition TCR value or parameter RAM value to the parameter LAST\_TRANS\_TIME and then incrementing the number of counts stored in TRANS\_COUNT. When the number of counts in TRANS\_COUNT equals the value in MAX\_COUNT, the TPU stores the final transition time into the parameter FINAL\_TRANS\_TIME and then requests an interrupt from the CPU.

Depending on the state of the host sequence field bits, the TPU can operate in one of four modes.

In **single shot mode without links**, the TPU counts the number of transitions specified in MAX\_COUNT and makes an interrupt service request. Then, channel activity ceases until reinitialization.

In **continual mode without links**, the TPU counts the number of transitions specified in MAX\_COUNT and makes an interrupt service request. The function then clears TRANS\_COUNT and continues to count transitions.

In **single shot mode with links**, the TPU counts the number of transitions specified in MAX\_COUNT, makes an interrupt service request, and generates a link service request to a sequential block of up to eight channels. The user specifies the starting channel of the block (in START\_LINK\_CHANNEL) and the number of channels within the block (in LINK\_CHANNEL\_COUNT). The TPU then ignores all subsequent transitions until the channel has been reinitialized.

In **continual mode with links**, the TPU counts the number of transitions specified in MAX\_COUNT, makes an interrupt service request, and generate a link to a sequential block of up to eight channels. The user specifies the starting channel of the block (in START\_LINK\_CHANNEL) and the number of channels within the block (in LINK\_CHANNEL\_COUNT). After these actions have been taken, the TPU clears TRANS\_COUNT and continues to count transitions.

The parameter capture mode (HSR %10) 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 to 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. There is latency associated with using the function in parameter capture mode. In the TCR mode, the value of the TCR is captured in hardware when the input transition is detected. In the parameter RAM mode the input transition causes the channel to request to be serviced by the TPU microengine; when the channel is serviced the parameter RAM value is captured.

### 3 Function Code Size

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

$$27 \mu \text{ instructions} + 8 \text{ entries} = \mathbf{35 \text{ long words}}$$

## 4 Function Parameters

This section provides detailed descriptions of the function parameters stored in channel parameter RAM. **Figure 1** shows TPU parameter RAM address mapping. **Figure 2** shows the parameter RAM assignment used by the function. In the diagrams, Y = M111, where M is the value of the module mapping bit (MM) in the system integration module configuration register (Y = \$7 or \$F).

Channel Number	Base Address	Parameter Address							
		0	1	2	3	4	5	6	7
0	\$YFFF##	00	02	04	06	08	0A	—	—
1	\$YFFF##	10	12	14	16	18	1A	—	—
2	\$YFFF##	20	22	24	26	28	2A	—	—
3	\$YFFF##	30	32	34	36	38	3A	—	—
4	\$YFFF##	40	42	44	46	48	4A	—	—
5	\$YFFF##	50	52	54	56	58	5A	—	—
6	\$YFFF##	60	62	64	66	68	6A	—	—
7	\$YFFF##	70	72	74	76	78	7A	—	—
8	\$YFFF##	80	82	84	86	88	8A	—	—
9	\$YFFF##	90	92	94	96	98	9A	—	—
10	\$YFFF##	A0	A2	A4	A6	A8	AA	—	—
11	\$YFFF##	B0	B2	B4	B6	B8	BA	—	—
12	\$YFFF##	C0	C2	C4	C6	C8	CA	—	—
13	\$YFFF##	D0	D2	D4	D6	D8	DA	—	—
14	\$YFFF##	E0	E2	E4	E6	E8	EA	EC	EE
15	\$YFFF##	F0	F2	F4	F6	F8	FA	FC	FE

— = Not Implemented (reads as \$00)

**Figure 1 TPU Channel Parameter RAM CPU Address Map**

**Figure 2** shows all of the host interface areas for the NITC function, as well as the parameters, addresses, reference times, and reference sources. This segment lists and defines the parameters for all modes of the NITC time function.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$YFFFW0									CHANNEL_CONTROL							
\$YFFFW2	START_LINK_CHANNEL				LINK_CHANNEL_COUNT				PARAM_ADDR				0			
\$YFFFW4	MAX_COUNT															
\$YFFFW6	TRANS_COUNT															
\$YFFFW8	FINAL_TRANS_TIME															
\$YFFFWA	LAST_TRANS_TIME															

Y= Channel number

Parameter Write Access

	Written by CPU
	Written by TPU
	Written by CPU and TPU
	Unused parameters

**Figure 2 NITC Function Parameter RAM Assignment**

**4.1 CHANNEL\_CONTROL**

CHANNEL\_CONTROL contains configuration for the PSC, PAC, and TBS fields. The channel executing this function is configured as input, and CHANNEL\_CONTROL must be written by the CPU before initialization. The PSC field is “don’t care” for input channels. The PAC field specifies the pin logic response for transition detection or an input channel. The TBS field configures a channel pin as input and configures the time base for capture events. Only the PAC field (bits [4:2]) are used in parameter RAM mode (HSR %10).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT USED								TBS			PAC			PSC	

**NITC CHANNEL\_CONTROL Options**

TBS				PAC			PSC		ACTION
8	7	6	5	4	3	2	1	0	
							1	1	—
				0	0	0			Do Not Detect Transition
				0	0	1			Detect Rising Edge
				0	1	0			Detect Falling Edge
				0	1	1			Detect Either Edge
				1	x	x			Do Not Change PAC
0	0	x	x						<b>Input Channel</b>
0	0	0	x						Capture TCR1
0	0	1	x						Capture TCR2

**4.2 START\_LINK\_CHANNEL**

START\_LINK\_CHANNEL contains the first channel of the link block. This parameter is written by the host CPU at initialization.

#### 4.3 LINK\_CHANNEL\_COUNT

LINK\_CHANNEL\_COUNT determines the number of channels in the link block. This parameter is written by the host CPU at initialization and can be changed at any time. If this parameter is used, it must be greater than zero and less than or equal to eight:  $0 < \text{count} \leq 8$ . No check is performed by the TPU. If this number is out of range, the results are unpredictable.

Example: If START\_LINK\_CHANNEL = \$F and LINK\_CHANNEL\_COUNT = 4, a link is generated, in order of appearance, to channels 15, 0, 1, and 2.

#### 4.4 PARAM\_ADDRESS

This is the address of the parameter that is captured instead of a TCR value when the function is operating in parameter RAM mode. This address should always point to an even address.

#### 4.5 MAX\_COUNT

MAX\_COUNT specifies the number of transitions to be counted before an interrupt is requested. In continuous mode, when MAX\_COUNT is reached, TRANS\_COUNT is reset to zero, and the TPU continues to count. In single mode, when MAX\_COUNT is reached, the TPU ignores all further input transitions. This parameter can be updated by the CPU.

#### 4.6 TRANS\_COUNT

TRANS\_COUNT and MAX\_COUNT should be accessed coherently by the CPU, which may change MAX\_COUNT and TRANS\_COUNT at any time. (Altering TRANS\_COUNT by the CPU is not recommended unless the system designer can ascertain that sufficient time remains before the TPU updates TRANS\_COUNT.) Refer to **6 Function Configuration** for MAX\_COUNT and TRANS\_COUNT alteration.

TRANS\_COUNT contains the current number of transitions counted. The TPU increments this parameter when a programmed transition is detected. When the NITC function is operating in continuous mode, TRANS\_COUNT is reset to zero at the start of every series of transitions counted. This parameter can also be updated by the CPU.

The TPU can overwrite the value written by the host CPU when operating in continuous mode. In this case, the CPU must ensure, by means of software interrogation, that sufficient time remains to complete the update before the TPU clears this parameter to zero.

TRANS\_COUNT and MAX\_COUNT should be accessed coherently by the host CPU. The CPU can change MAX\_COUNT and TRANS\_COUNT at any time, but altering TRANS\_COUNT is not recommended unless the system designer can ascertain that sufficient time remains before the TPU updates TRANS\_COUNT. Refer to **6 Function Configuration** for MAX\_COUNT and TRANS\_COUNT alteration.

#### 4.7 FINAL\_TRANS\_TIME

FINAL\_TRANS\_TIME contains the TCR value or parameter RAM value when the final transition of MAX\_COUNT is reached. This parameter is updated by the TPU when the number of transitions counted is equal to or greater than MAX\_COUNT. An interrupt is requested when FINAL\_TRANS\_TIME is updated.

#### 4.8 LAST\_TRANS\_TIME

LAST\_TRANS\_TIME contains the TCR value or parameter RAM when the last transition is detected. The TPU updates this parameter whenever the specified transition occurs and the number of transitions counted is less than MAX\_COUNT.

## 5 Host Interface to Function

This section provides information concerning the TPU host interface to the function. **Figure 3** is a TPU address map. Detailed TPU register diagrams follow the figure. In the diagrams, Y = M111, where M is the value of the module mapping bit (MM) in the system integration module configuration register (Y = \$7 or \$F).

Address	15	8	7	0
\$YFFE00	TPU MODULE CONFIGURATION REGISTER (TPUMCR)			
\$YFFE02	TEST CONFIGURATION REGISTER (TCR)			
\$YFFE04	DEVELOPMENT SUPPORT CONTROL REGISTER (DSCR)			
\$YFFE06	DEVELOPMENT SUPPORT STATUS REGISTER (DSSR)			
\$YFFE08	TPU INTERRUPT CONFIGURATION REGISTER (TICR)			
\$YFFE0A	CHANNEL INTERRUPT ENABLE REGISTER (CIER)			
\$YFFE0C	CHANNEL FUNCTION SELECTION REGISTER 0 (CFSR0)			
\$YFFE0E	CHANNEL FUNCTION SELECTION REGISTER 1 (CFSR1)			
\$YFFE10	CHANNEL FUNCTION SELECTION REGISTER 2 (CFSR2)			
\$YFFE12	CHANNEL FUNCTION SELECTION REGISTER 3 (CFSR3)			
\$YFFE14	HOST SEQUENCE REGISTER 0 (HSQR0)			
\$YFFE16	HOST SEQUENCE REGISTER 1 (HSQR1)			
\$YFFE18	HOST SERVICE REQUEST REGISTER 0 (HSRR0)			
\$YFFE1A	HOST SERVICE REQUEST REGISTER 1 (HSRR1)			
\$YFFE1C	CHANNEL PRIORITY REGISTER 0 (CPR0)			
\$YFFE1E	CHANNEL PRIORITY REGISTER 1 (CPR1)			
\$YFFE20	CHANNEL INTERRUPT STATUS REGISTER (CISR)			
\$YFFE22	LINK REGISTER (LR)			
\$YFFE24	SERVICE GRANT LATCH REGISTER (SGLR)			
\$YFFE26	DECODED CHANNEL NUMBER REGISTER (DCNR)			

**Figure 3 TPU Address Map**

### CIER — Channel Interrupt Enable Register

**\$YFFE0A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15	CH 14	CH 13	CH 12	CH 11	CH 10	CH 9	CH 8	CH 7	CH 6	CH 5	CH 4	CH 3	CH 2	CH 1	CH 0

CH	Interrupt Enable
0	Channel interrupts disabled
1	Channel interrupts enabled

### CFSR[0:3] — Channel Function Select Registers

**\$YFFE0C – \$YFFE12**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CFS (CH 15, 11, 7, 3)			CFS (CH 14, 10, 6, 2)			CFS (CH 13, 9, 5, 1)			CFS (CH 12, 8, 4, 0)						

CFS[4:0] — NITC Function Number (Assigned during microcode assembly)

**HSQR[0:1] — Host Sequence Registers**

**\$YFFE14 – \$YFFE16**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15, 7		CH 14, 6		CH 13, 5		CH 12, 4		CH 11, 3		CH 10, 2		CH 9, 1		CH 8, 0	

CH	Mode
00	Single Shot, No Links
01	Continual, No Links
10	Single Shot, Links
11	Continual, Links

**HSRR[1:0] — Host Service Request Registers**

**\$YFFE18 – \$YFFE1A**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15, 7		CH 14, 6		CH 13, 5		CH 12, 4		CH 11, 3		CH 10, 2		CH 9, 1		CH 8, 0	

CH	Initialization
00	No Host Service (Reset Condition)
01	Initialize TCR mode
10	Initialize parameter mode
11	Not Used

**CPR[1:0] — Channel Priority Registers**

**\$YFFE1C – \$YFFE1E**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15, 7		CH 14, 6		CH 13, 5		CH 12, 4		CH 11, 3		CH 10, 2		CH 9, 1		CH 8, 0	

CH	Channel Priority
00	Disabled
01	Low
10	Middle
11	High

**CISR — Channel Interrupt Status Register**

**\$YFFE20**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15	CH 14	CH 13	CH 12	CH 11	CH 10	CH 9	CH 8	CH 7	CH 6	CH 5	CH 4	CH 3	CH 2	CH 1	CH 0

CH	Interrupt Status
0	Channel interrupt not asserted
1	Channel interrupt asserted

## 6 Function Configuration

The CPU initializes the function as follows.

1. Writes CHANNEL\_CONTROL and MAX\_COUNT values to parameter RAM.
2. If running in link mode (host sequence bits = 1x), writes START\_LINK\_CHANNEL and LINK\_CHANNEL\_COUNT to parameter RAM.
3. If running in parameter mode (host service request = %10), writes PARAM\_ADDRESS to parameter RAM.
4. Writes host sequence bits according to the mode of operation.
5. Issues an HSR %01 or HSR %10 for initialization.
6. Enables channel servicing by assigning high, middle, or low priority.

The TPU then executes initialization and starts counting the transition type specified by the PAC field in CHANNEL\_CONTROL. The CPU should monitor the HSR register until the TPU clears the service request to 00 before changing any parameters or issuing a new service request to this channel.

### 6.1 MAX\_COUNT and TRANS\_COUNT Alteration

If MAX\_COUNT is changed by the CPU, the TPU uses the new value on the next transition detected. Because TRANS\_COUNT can be written both by the CPU and by the TPU, one of the following cases can occur if the CPU alters the TRANS\_COUNT value:

- A. The TPU uses the new value of (TRANS\_COUNT + 1). This case is the most probable and happens when:
  1. The CPU writes a new value to TRANS\_COUNT;
  2. The TPU increments TRANS\_COUNT; and
  3. The TPU reads TRANS\_COUNT and MAX\_COUNT to compare them.
- B. The TPU uses the new value of TRANS\_COUNT. This case happens when:
  1. The TPU increments TRANS\_COUNT;
  2. The CPU writes a new value to TRANS\_COUNT; and
  3. The TPU reads TRANS\_COUNT and MAX\_COUNT to compare them.
- C. The new value of TRANS\_COUNT is overwritten by TPU. This case occurs when the CPU writes a new value to TRANS\_COUNT just as TRANS\_COUNT equals the value of MAX\_COUNT (if operating in continuous mode). This case, which should be handled according to the specific application, happens when:
  1. The TPU increments TRANS\_COUNT;
  2. The TPU reads TRANS\_COUNT and MAX\_COUNT to compare them, and TRANS\_COUNT ≥ MAX\_COUNT;
  3. The CPU writes a new value to TRANS\_COUNT; and
  4. If operating in continuous mode, the TPU resets TRANS\_COUNT to zero to initialize a new series of counts.

## 7 Performance and Use of Function

### 7.1 Performance

Like all TPU functions, NITC function performance in an application is to some extent dependent upon the service time (latency) of other active TPU channels. This is due to the operational nature of the scheduler. When more TPU channels are active, performance decreases. However, worst-case latency in any TPU application can be closely estimated. To analyze the performance of an application that appears to approach the limits of the TPU, use the guidelines given in the TPU reference manual and the information in the NITC state timing table below.



**Table 1 NITC State Timing**

State Number and Name	Max. CPU Clock Cycles	RAM Accesses by TPU
S0 Init	8	2
S1 Inita	6	2
S2 Count_Up (last count)		
HSQ = 00	32	5
HSQ = 01	30	6
HSQ = 10	32	6
HSQ = 11	32 <sup>1</sup>	7
All modes (not last count)	24	5

**NOTES**

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

**7.2 Changing Mode**

The host sequence bits are used to select NITC function operating mode. Change host sequence bit values only when the function is stopped or disabled (channel priority bits =%00). Disabling the channel before changing mode avoids conditions that cause indeterminate operation.

**8 Examples**

The following examples show configuration of the new input capture/input transition function. Each example includes a description of the example, a diagram of the initial parameter RAM content and the initial control bit settings.

**8.1 Example A**

**8.1.1 Description**

Configure channel 1 to run NITC in single shot mode without links. Set up channel control to detect either edge and to capture TCR1.

**8.1.2 Initialization**

Disable channel 1 by clearing the priority bits. Select NITC function by programming the function select register of each channel. Configure parameter RAM as shown below. Write HSQ =%00 to channel 1. Issue HSR =%01 to initialize in TCR mode. Write the priority bits to a non-zero value.

**Table 2 Channel 1 Parameter RAM**

	15	8	0	
\$YFFF10	x x x x x x x x	0 0 0 0 0 1 1 1	1 1	CHANNEL_CONTROL
\$YFFF12	x x x x x x x x	x x x x x x x x	0	LINK & PARAM_ADDR
\$YFFF14	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0	MAX_COUNT
\$YFFF16	x x x x x x x x	x x x x x x x x	x x	TRANS_COUNT
\$YFFF18	x x x x x x x x	x x x x x x x x	x x	FINAL_TRANS_TIME
\$YFFF1A	x x x x x x x x	x x x x x x x x	x x	LAST_TRANS_TIME

CHANNEL\_CONTROL = \$000F

MAX\_COUNT = \$0000

The function now runs. It captures any transition on channel 1, stores the value of TCR1 in FINAL\_TRANS\_TIME and LAST\_TRANS\_TIME, and issues an interrupt request to the host CPU.

**8.2 Example B**

**8.2.1 Description**

Configure channel 1 to run NITC in continuous mode and link to channels 5 and 6. Set up channel control to detect falling edges and to capture TCR2. In continuous mode count five falling edges.

**8.2.2 Initialization**

Disable channel 1 by clearing the priority bits. Select NITC function by programming the function select register of each channel. Configure parameter RAM as shown below. Write HSQ = %11 to channel 1. Issue HSR = %01 to initialize in TCR mode. Write the priority bits to a non-zero value.

**Table 3 Channel 1 Parameter RAM**

	15							8							0							
\$YFFF10	x	x	x	x	x	x	x	0	0	1	0	0	1	0	1	1	CHANNEL_CONTROL					
\$YFFF12	0	1	0	1	0	0	1	1	x	x	x	x	x	x	x	0	LINK & PARAM_ADDR					
\$YFFF14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	MAX_COUNT					
\$YFFF16	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	TRANS_COUNT					
\$YFFF18	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	FINAL_TRANS_TIME					
\$YFFF1A	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	LAST_TRANS_TIME					

CHANNEL\_CONTROL = \$004B  
 MAX\_COUNT = \$0005

The function now runs. It captures falling transitions on channel 1, stores the value of TCR2 in LAST\_TRANS\_TIME, and increments TRANS\_COUNT. Every fifth transition causes the value of TCR2 to be stored in FINAL\_TRANS\_TIME and LAST\_TRANS\_TIME, links to be sent to channels 5 and 6, TRANS\_COUNT to be cleared to zero, and an interrupt request to be sent to the host CPU.

**8.3 Example C**

**8.3.1 Description**

Configure channel 1 to run NITC in single shot mode without links. Set up channel control to detect rising edges and to capture the value in parameter RAM location \$22.

**8.3.2 Initialization**

Disable channel 1 by clearing the priority bits. Select NITC function by programming the function select register of each channel. Configure parameter RAM as shown below. Write HSR = %00 to channel 1. Issue HSR = %10 to initialize in parameter RAM mode. Write the priority bits to a non-zero value.

**Table 4 Channel 1 Parameter RAM**

	15								8								0			
\$YFFF10	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	1	1	1	CHANNEL_CONTROL	
\$YFFF12	x	x	x	x	x	x	x	x	0	0	1	0	0	0	0	1	0	LINK & PARAM_ADDR		
\$YFFF14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MAX_COUNT		
\$YFFF16	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	TRANS_COUNT		
\$YFFF18	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	FINAL_TRANS_TIME		
\$YFFF1A	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	LAST_TRANS_TIME		

CHANNEL\_CONTROL = \$0007  
 PARAM\_ADDR = \$0022  
 MAX\_COUNT = \$0000

The function now runs, captures a rising transition on channel 1, and stores the value of parameter RAM location \$22 in FINAL\_TRANS\_TIME and LAST\_TRANS\_TIME. An interrupt request is made when these actions are complete.

## 9 Function Algorithm

At each transition detected, the TPU increments TRANS\_COUNT and updates LAST\_TRANS\_TIME to the value of the TCR or parameter RAM location. If TRANS\_COUNT is greater than or equal to MAX\_COUNT, then 1) FINAL\_TRANS\_TIME is updated to contain the time of the last transition detected, 2) an interrupt is asserted causing an interrupt to be generated (if interrupt enable bit is set), 3) if the time function is in link mode, links to a sequential block of channels are generated as specified by the START\_LINK\_CHANNEL and LINK\_CHANNEL\_COUNT parameters, and 4) if continuous mode, clears TRANS\_COUNT.

The NITC function consists of the following states:

### 9.1 State 0:Init

Initialization is entered as a result of HSR %01. The channel executing the time function is configured. TRANS\_COUNT is initialized to zero. The transition type and time base are selected as per the CHANNEL\_CONTROL parameter. TCR or parameter mode is determined by flag0.

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 01xxxx  
Match Enable: Don't Care

```

Configure channel latches via CHANNEL_CONTROL
Clear flag0
TRANS_COUNT = 0
Negate MRL, TDL, and LSR
    
```

### 9.2 State 1:Inita

Initialization is entered as a result of HSR %10. The channel executing the time function is configured. TRANS\_COUNT is initialized to zero. The transition type and time base are selected as per the CHANNEL\_CONTROL parameter. TCR or parameter mode is determined by flag0.

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 10xxxx  
Match Enable: Don't Care

```

Configure channel latches via CHANNEL_CONTROL
Set flag0
TRANS_COUNT = 0
Negate MRL, TDL, and LSR
    
```

### 9.3 State 2:Count\_Up

- FINAL\_TRANS\_TIME is updated to contain the time of the last transition, and an interrupt request is asserted.
- If the time function is in link mode, links are generated to a sequential block of channels as specified by START\_LINK\_CHANNEL and LINK\_CHANNEL\_COUNT.
- If the time function operates in continuous mode, the function re-executes state 1, and TRANS\_COUNT is reinitialized to zero. In single (noncontinuous) mode, the function terminates by ignoring all further transitions.

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 001xxx  
Match Enable: Don't Care

```

TRANS_COUNT = TRANS_COUNT + 1
If TRANS_COUNT ≥ MAX_COUNT then {
    FINAL_TRANS_TIME = time of last transition.
    If host sequence bit 1 = 1 then {
    
```

```

Link to channels START_LINK_CHANNEL
to [START_LINK_CHANNEL + LINK_CHANNEL_COUNT - 1]
}
If host sequence bit 0 = 0 then {
    negate MRL, TDL, LSL
    interrupt request
    ignore further transitions
}
Else {
    configure channel latches via CHANNEL_CONTROL
    TRANS_COUNT = 0
    Negate MRL, TDL, LSL
}
}
Else{
    LAST_TRANS_TIME = time of last transition
    Negate TDL
}

```

The following table shows the NITC transitions listing the service request sources and channel conditions from current state to next state. **Figure 4** illustrates the flow of NITC states.

**Table 5 NITC State Transition Table**

Current State	HSR	M/TSR	LSR	Pin	Flag0	Flag1	Next State
Any State	01	—	—	—	—	—	S0 Init
Any State	10	—	—	—	—	—	S1 Init
S0 Init	00	1	—	—	—	—	S2 Count_Up
S1 Init	00	1	—	—	—	—	S2 Count_Up
S2 Count_Up	00	1	—	—	—	—	S2 Count_Up
Unimplemented Conditions	00 11	0 —	1 —	— —	— —	— —	— —

**NOTES:**

1. Conditions not specified are “don't care.”
2. LSR = Link service request  
 HSR = Host service request  
 M/TSR = Either a match or transition (input capture) service request occurred (M/TSR = 1) or neither occurred (M/TSR = 0).

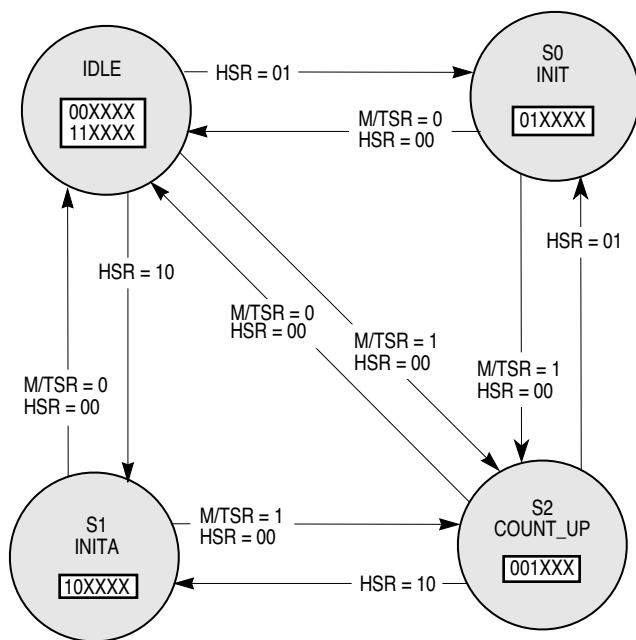


Figure 4 NITC State Flow Diagram



NOTES

## How to Reach Us:

### Home Page:

[www.freescale.com](http://www.freescale.com)

### E-mail:

[support@freescale.com](mailto: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](mailto: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](mailto: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](mailto: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](mailto: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](mailto:LDCForFreescaleSemiconductor@hibbertgroup.com)

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.