Input Capture/Input Transition Counter TPU Function (ITC)

by Sharon Darley

1 Functional Overview

The ITC function can detect rising or falling input transitions. When a transition is detected, the current TCR timer 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 ITC function can count the programmed maximum number of transitions continually, or it can count the programmed number of transitions once, ceasing channel activity until reinitialization. Once the programmed maximum number of transitions is counted, the TPU can request an interrupt from the CPU and generate a link to a sequential block of up to eight channels. (A link causes another channel or group of channels to request service.) The user specifies the starting channel of the block and the number of channels within the block.

2 Detailed Description

Input Capture

Any channel of the TPU can perform an input capture. Performing an input capture means recording the value of the selected TCR of an input transition. If only one input capture is desired, then the value in MAX_COUNT should be set to zero or one.

Input Transition Counter

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 time 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 is equal to or greater than the value in MAX_COUNT, the TPU stores the TCR value into the parameter FINAL_TRANS_TIME and requests an interrupt from the CPU. It also increments the high byte of the memory location pointed to by the parameter BANK_ADDRESS. This feature is intended for use with the PMA/PMM functions in engine control applications. Typically, the ITC parameter BANK_ADDRESS points to the PMM/PMA parameter BANK_SIGNAL. If this feature is not desired, BANK_ADDRESS must point to an unimplemented address of the TPU parameter RAM, such as $0E.

Depending on the state of the host sequence field bits, the TPU can operate in one of four modes. These four modes are described in detail in the following paragraphs.

2.1 Single Shot Without Links

In this mode, the TPU counts the number of transitions specified in MAX_COUNT, requests an interrupt, and increments the high byte of the memory location pointed to by BANK_ADDRESS. Channel activity then ceases until reinitialization.

2.2 Continual Without Links

In this mode, the TPU counts the programmed number of transitions specified in MAX_COUNT, requests an interrupt, and increments the high byte of the memory location pointed to by BANK_ADDRESS. TRANS_COUNT is cleared to zero. The channel continues to count transitions.

2.3 Single Shot With Links

In this mode, the TPU counts the programmed number of transitions specified in MAX_COUNT, requests an interrupt, and increments the high byte of the memory location pointed to by BANK_ADDRESS. In addition, a link service request is generated 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 ignores all subsequent transitions until the channel has been reinitialized.

2.4 Continual With Links

In this mode, each time the TPU counts the programmed number of transitions specified in MAX_COUNT, it requests an interrupt and increments the byte pointed to by BANK_ADDRESS. It also generates 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). Finally, the TPU clears the value in TRANS_COUNT to zero and continues to count transitions.

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. ITC function code size is:

27 µ instructions + 6 entries = **33 long words**

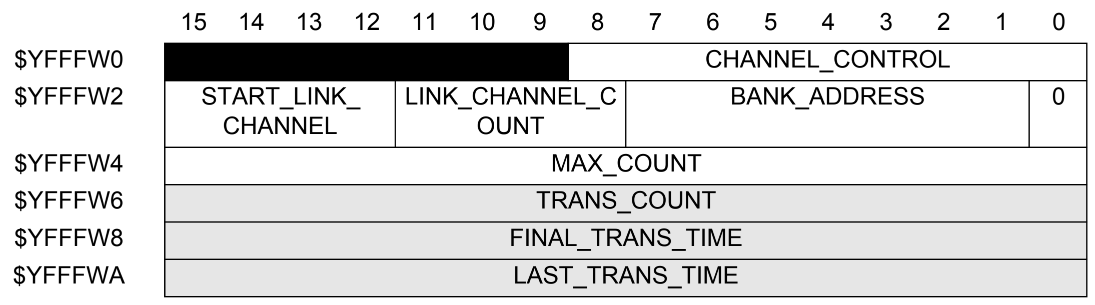
4 Function Parameters

This section provides detailed descriptions of input transition counter 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 ITC 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



W = Channel number

Parameter Write Access:

□	Written by CPU
░	Written by TPU
▒	Written by CPU and TPU
■	Unused parameters

Figure 2 ITC Function Parameter RAM Assignment

4.1 CHANNEL_CONTROL

CHANNEL_CONTROL contains the channel latch controls and configures 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 which type of transitions to detect: low-to-high, high-to-low, any, or no transitions. The TBS field configures a channel pin as an input and specifies which time base (TCR1 or TCR2) to use. The following table defines the allowable data for this parameter.

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

Table 1 ITC 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						**Input Channel**
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.

NOTE

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 BANK_ADDRESS

BANK_ADDRESS contains the address of a TPU parameter RAM location. The high byte of the RAM location pointed to by this parameter is incremented when a programmable number of specified transitions occur. If this increment is not desired, BANK_ADDRESS should point to an unused TPU parameter RAM location such as $0E. This way, the unused memory location will be incremented instead of one that might affect program operation.

4.5 MAX_COUNT

MAX_COUNT specifies the number of transitions to be counted before an interrupt is requested and the high byte of the memory location pointed to by BANK_ADDRESS is incremented. In continuous mode, when MAX_COUNT is reached, TRANS_COUNT is reset to zero, and the TPU continues to count. In single-shot mode, when MAX_COUNT is reached, the TPU ignores all further input transitions. The CPU can update this parameter.

NOTE

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.

4.6 TRANS_COUNT

TRANS_COUNT contains the current number of transitions counted. The TPU increments this parameter when a programmed transition is detected. When the ITC function is operating in continuous mode, TRANS_COUNT is reset to zero at the start of every series of transitions counted. The CPU can also update this parameter.

NOTE

In continuous mode, the TPU may overwrite the value written by the host CPU, i.e., the CPU may have written a new value just before the TPU was about to reset this parameter to zero. In this case, the CPU (via software interrogation) must ensure that sufficient time remains to complete the update.

TRANS_COUNT and MAX_COUNT should be accessed coherently by the host 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.

4.7 FINAL_TRANS_TIME

FINAL_TRANS_TIME contains the TCR time of the final transition when 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 of the previous transition. 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 ITC 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] — 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[15:0]	Action Taken
00	Single shot, no links
01	Continual, no links
10	Single shot, links
11	Continual, links

HSRR[0:1] — 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[15:0]	Action
01	Initialization

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[15:0]	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 by the following:

1. Writing CHANNEL_CONTROL, MAX_COUNT, and BANK_ADDRESS to parameter RAM;
2. Writing START_LINK_CHANNEL and LINK_CHANNEL_COUNT to parameter RAM if running in link mode (host sequence bits = 1x);
3. Writing host sequence bits according to the mode of operation;
4. Issuing an HSR %01 for initialization; and
5. Enabling channel servicing by assigning a high, middle, or low priority.

The TPU 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 the CPU changes MAX_COUNT, the TPU uses the new value on the next transition detected. Because TRANS_COUNT can be written by both the CPU and 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 plus one). 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 the TPU. This case occurs when the CPU writes a new value to TRANS_COUNT just as TRANS_COUNT equals the value of MAX_COUNT (in continuous mode only). 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. The TPU resets TRANS_COUNT to zero to initialize a new series of counts (in continuous mode).

7 Performance and Use of Function

7.1 Performance

Like all TPU functions, ITC 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. The more TPU channels are active, the more performance decreases. 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 ITC state timing table below.

Table 2 ITC State Timing

State Number and Name	Max CPU Clock Cycles	RAM Accesses by TPU
S1 Init	18	4
S2 Count_Up (last count)		
HSB = 00	30	6
HSB = 01	36	7
HSB = 10	34[1]	6
HSB = 11	40[1]	7
All modes (not last count)	16	4

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 the ITC 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 Function Examples

8.1 Example A

8.1.1 Description

The following program uses the continuous mode to count input pulses and generate an interrupt each time MAX_COUNT reaches a pre-specified value. In the continuous mode with no links, the ITC function repeatedly counts the number of transitions programmed in MAX_COUNT. Each time TRANS_COUNT reaches the value in MAX_COUNT, TRANS_COUNT resets to zero. If BANK_ADDRESS points to a valid parameter address, then the value in the high byte of that address is incremented by one. If interrupts are enabled, as in this example, an interrupt request is made. BANK_ADDRESS points to an unimplemented RAM location so that it does not affect operation of other channels. For an example of how to use the parameter BANK_ADDRESS, see Example A of *Period Measurement with Additional Transition Detection (PMA)* (TPUPN15A/D) or *Period Measurement with Missing Transition Detection (PMM)* (TPUPN15B/D).

In this program, the ITC function on channel 1 counts input pulses from the PWM function on channel 0. Each time the ITC function counts seven pulses, it resets TRANS_COUNT to zero and requests an interrupt. In order to see this is happening, the interrupt routine increments the pulse HIGH_TIME of the SPWM wave by one. Thus, when the program runs, the output of the SPWM channel will be a square wave with a constantly changing duty cycle.

Channel 0 is set up in PWM mode, channel 1 is set up in ITC mode, and channel 2 is set up in SPWM mode. A physical connection must be made between channels 0 and 1.

8.2 Program Code for CPU32-Based Microcontrollers

This program was assembled using the IASM32 assembler available from P&E Microcomputer Systems with the M68332 In-Circuit Debugger. It was run on an M68332EVS and BCC.

```
TPUMCR    equ       $fffe00
TICR      equ       $fffe08
CIER      equ       $fffe0a
CFSR0     equ       $fffe0c
CFSR1     equ       $fffe0e
CFSR2     equ       $fffe10
CFSR3     equ       $fffe12
HSQR0     equ       $fffe14
HSQR1     equ       $fffe16
HSRR0     equ       $fffe18
HSRR1     equ       $fffe1a
CPR0      equ       $fffe1c
CPR1      equ       $fffe1e
CISR      equ       $fffe20
* Channel 0 Parameters
PRAM0_0   equ       $ffff00
PRAM0_1   equ       $ffff02
PRAM0_2   equ       $ffff04
PRAM0_3   equ       $ffff06
PRAM0_4   equ       $ffff08
PRAM0_5   equ       $ffff0a
* Channel 1 parameters
PRAM1_0   equ       $ffff10
PRAM1_1   equ       $ffff12
PRAM1_2   equ       $ffff14
PRAM1_3   equ       $ffff16
PRAM1_4   equ       $ffff18
PRAM1_5   equ       $ffff1a
* Channel 2 parameters
PRAM2_0   equ       $ffff20
PRAM2_1   equ       $ffff22
```

```
PRAM2_2   equu      $ffff24
PRAM2_3   equ       $ffff26
PRAM2_4   equ       $ffff28
PRAM2_5   equ       $ffff2a
```

8.3 Initialization

```
          org       $4000                   ;begin at memory location $4000
          move.w    #$07a9,(CFSR3).l        ;Channel Function Select Field (Function
                                            ;numbers may vary for different mask sets)
          move.w    #$00ff,(CPR1).l         ;Channel Priority Field, high priority
          move.w    #$0004,(HSQR1).l        ;ITC mode = continual with no links
                                            ;SPWM = mode 0
```

8.3.1 PWM Initialization for Channel 0

This PWM wave will have a pulse period of $200 and a pulse hightime of $100. The ITC function on channel 1 will count the rising edges.

```
          move.w    #$0092,(PRAM0_0).l      ;Channel Control, use TCR1
          move.w    #$0100,(PRAM0_2).l      ;pulse hightime = 100
          move.w    #$0200,(PRAM0_3).l      ;pulse period = 200
```

8.3.2 ITC Initialization for Channel 1

Since this program does not need to increment a parameter used by another function each time the number of transitions specified in MAX_COUNT has been counted, BANK_ADDRESS points to an unimplemented memory location.

```
          move.w    #$0007,(PRAM1_0).l      ;Channel control, detect rising edge, use
TCR1
          move.w    #$000e,(PRAM1_1).l      ;BANK_ADDRESS points to unimplemented RAM
          move.w    #$0007,(PRAM1_2).l      ;MAX_COUNT = 7
```

8.3.3 SPWM Initialization for Channel 2 in Mode 0

The SPWM is set up in mode 0. It is initialized with a pulse hightime of $100 and a period of $600. The interrupt routine changes the pulse hightime. REF_ADDR1 points to a reference value to which DELAY and PERIOD are added to form the rising transition time. Here, it points to LASTRISE. LASTRISE contains the TCR time of the previous low to high transition.

```
          move.w    #$92,(PRAM2_0).l        ;Channel Control
          move.w    #$100,(PRAM2_2).l       ;HIGH_TIME = $100
          move.w    #$600,(PRAM2_3).l       ;PERIOD = $600
          move.w    #$0020,(PRAM2_4).l      ;REF_ADDR1=$20 (LASTRISE)
          move.w    #$0000,(PRAM2_5).l      ;DELAY = 0
```

8.3.4 Initialization of Interrupts for Channel 1

First, disable all TPU interrupts by writing zeros to the channel interrupt enable register (CIER). Second, negate all TPU interrupt status flags by first reading the channel interrupt status register (CISR) and then writing it to all zeros. Third, start the interrupt routine at the label INT by storing the address of INT in the appropriate vector address location. For this example, the base vector number $80 is chosen. This number is stored in the TPU interrupt configuration register (TICR). The actual interrupt vector number is calculated by concatenating the base vector with the channel number. Thus, the interrupt vector number is $81, since the program uses channel 1. The vector address (where the starting address of the interrupt routine is stored) is calculated as four times the vector number plus the value in the vector base register. In this case, the program was run on Freescale Business Card Computer (BCC) and the vector base register has already been initialized to $400 by CPU32Bug, so the vector address is $4 ∗ $81 + $400, which equals $604.

The interrupt level must be set to a non-zero value in the TICR. The interrupt level chosen determines the priority given to this interrupt by the CPU. Level 7 is the highest priority, and level 1 is the lowest. This example uses level 6. Once an interrupt level has been chosen, bits [10:8] in the CPU status register must be modified to allow recognition of that level of interrupt. These bits must be set to a number that is lower than the interrupt level number to be recognized. Interrupts at the same level or lower than the number in the CPU status register will be masked out and will not be recognized by the CPU. In addition, the interrupt arbitration (IARB) field in the TPU module configuration register (TPUMCR) must be set to a value between $1 and $F. This IARB field determines the interrupt's priority if two or more modules with the same interrupt level request interrupts at the same time. In this example, the IARB field is set to $5.

```
          andi.w    #$fffd,(CIER).l         ;disable TPU interrupts on Channel 1
          move.w    (CISR).l,d0             ;clear TPU interrupt requests for Channel 1
          andi.w    #$fffd,(CISR).l
          move.l    #INT,($0604).l          ;start interrupt routine at label INT
          ori.w     #$0005,(TPUMCR).l       ;set IARB field
          move.w    #$0680,(TICR).l         ;interrupt level 6, base vector = $80
          andi.w    #$f5ff,SR               ;allow interrupts on level 6 and above
```

8.3.5 Service Initialization Request

Initialize channels 0, 1 and 2 and enable interrupts for channel 1.

```
          move.w    #$0026,(HSRR1).l        ;Initialization for ch 0, 1, 2
          ori.w     #$0002,(CIER).l         ;enable interrupts for channel 1
finish
          bra       finish
```

8.3.6 Interrupt Routine

This routine will be called each time TRANS_COUNT reaches MAX_COUNT. It will increment HIGH_TIME of the SPWM wave until it reaches $500. Then, the HIGH_TIME will be reinitialized to $100.

```
INT
          andi.w    #$fffd,(CIER).l         ;disable interrupt in CIER
          move.w    (CISR).l,d0             ;read interrupt flag
          andi.w    #$fffd,(CISR).l         ;clear interrupt
          cmpi.w    #$500,(PRAM2_2).l       ;compare HIGH_TIME to $500
          beq       chng
          addi.w    #$0001,(PRAM2_2).l      ;add $01 to HIGH_TIME of SPWM wave
          bra       done
chng
          move.w    #$0100,(PRAM2_2).l      ;reset HIGH_TIME to #$100
done
          ori.w     #$0002,(CIER).l         ;enable interrupt for channel 1
          RTE                               ;Return from Exception
```

8.4 Program Code for CPU16-Based Microcontrollers

This program was assembled on the IASM16 Assembler available with the ICD16 In-Circuit Debugger from P&E Microcomputer Systems and was run on an MC68HC16Y1EVB.

```
TPUMCR    equ       $fe00
TICR      equ       $fe08
CIER      equ       $fe0a
CFSR0     equ       $fe0c
CFSR1     equ       $fe0e
CFSR2     equ       $fe10
CFSR3     equ       $fe12
HSQR0     equ       $fe14
HSQR1     equ       $fe16
HSRR0     equ       $fe18
HSRR1     equ       $fe1a
```

```
CPR0      equu      $fe1c
CPR1      equ       $fe1e
CISR      equ       $fe20
* Channel 0 parameters
PRAM0_0   equ       $ff00
PRAM0_1   equ       $ff02
PRAM0_2   equ       $ff04
PRAM0_3   equ       $ff06
PRAM0_4   equ       $ff08
PRAM0_5   equ       $ff0a
* Channel 1 parameters
PRAM1_0   equ       $ff10
PRAM1_1   equ       $ff12
PRAM1_2   equ       $ff14
PRAM1_3   equ       $ff16
PRAM1_4   equ       $ff18
PRAM1_5   equ       $ff1a
* Channel 2 parameters
PRAM2_0   equ       $ff20
PRAM2_1   equ       $ff22
PRAM2_2   equ       $ff24
PRAM2_3   equ       $ff26
PRAM2_4   equ       $ff28
PRAM2_5   equ       $ff2a
```

8.4.1 Initialization

The following code is included to set up the reset vector ($00000 – $00006). It may be changed for different systems.

```
ORG       $0000                    ;put the following reset vector information
                                   ;at address $00000 of the memory map
DW        $0000                    ;zk=0, sk=0, pk=0
DW        $0200                    ;pc=200 -- initial program counter
DW        $3000                    ;sp=3000 -- initial stack pointer
DW        $0000                    ;iz=0 -- direct page pointer
org       $0400                     ;begin program at memory location $0400
```

The following code initializes and configures the system; including the software watchdog and system clock. It was written to be used with an EVB.

```
INITSYS:                           ;give initial values for extension regis-
ters
                                   ;and initialize system clock and COP
          LDAB      #$0F
          TBEK                     ;point EK to bank F for register access
          LDAB      #$00
          TBXK                     ;point XK to bank 0
          TBYK                     ;point YK to bank 0
          TBZK                     ;point ZK to bank 0
          TBSK
          LDD       #$0003         ;at reset, the CSBOOT block size is 512K.
          STD       CSBARBT        ;this line sets the block size to 64K since
                                   ;that is what physically comes with the EVB16
          LDAA      #$7F           ;w=0, x=1, y=111111
          STAA      SYNCR          ;set system clock to 16.78 MHz
          CLR       SYPCR          ;turn COP off, since COP is on after reset
          lds       #$f000
**** MAIN PROGRAM ****
          ldab      #$0f
          tbek                     ;use bank $0f for parameter RAM
          ldab      #$00
          tbzk
          ldz       #$0000         ;use IZ for indexed offset
          ldd       #$07a9
          std       CFSR3          ;Channel Function Select Field (Note: func-
tion
          ldd       #$00ff                   ;numbers may vary for different mask sets)
          std       CPR1                     ;Channel Priority Field, high priority
          ldd       #$0004
          std       HSQR1                    ;ITC mode = continual with no links
                                             ;SPWM = mode 0
```

8.4.2 PWM Initialization for Channel 0

This PWM wave will have a pulse period of $200 and a pulse hightime of $100. The ITC function on channel 1 will count the rising edges.

```
          ldd       #$0092
          std       PRAM0_0                  ;Channel Control, use TCR1
          ldd       #$0100
          std       PRAM0_2                  ;pulse hightime = 100
          ldd       #$0200
          std       PRAM0_3                  ;pulse period = 200
```

8.4.3 ITC Initialization for Channel 1

Since this program does not need to increment a parameter in another memory location each time the number of transitions specified in MAX_COUNT has been counted, BANK_ADDRESS points to an unimplemented memory location. For an example in which BANK_ADDRESS is used, see Example A in *Period Measurement with Additional Transition Detection (PMA)* (TPUPN15A/D) or *Period Measurement with Missing Transition Detection (PMM)* (TPUPN15B/D).

```
          ldd       #$0007
          std       PRAM1_0                  ;Channel control, detect rising edge, use
TCR1
          ldd       #$000e
          std       PRAM1_1                  ;BANK_ADDRESS points to unimplemented RAM
          ldd       #$0007
          std       PRAM1_2                  ;MAX_COUNT = 7
```

8.4.4 SPWM Initialization for Channel 2 in Mode 0

The SPWM is set up in mode 0. It is initialized with a pulse hightime of $100 and a period of $600. The interrupt routine changes the pulse hightime. REF_ADDR1 points to a reference value to which DELAY and PERIOD are added to form the rising transition time. Here, it points to LASTRISE. LASTRISE contains the TCR time of the previous low to high transition.

```
          ldd       #$92
          std       PRAM2_0                  ;Channel Control
          ldd       #$100
          std       PRAM2_2                  ;HIGH_TIME = $100
          ldd       #$600
          std       PRAM2_3                  ;PERIOD = $600
          ldd       #$0020
          std       PRAM2_4                  ;REF_ADDR1=$20 (LASTRISE)
          ldd       #$0000
          std       PRAM2_5                  ;DELAY = 0
```

8.4.5 Initialization of Interrupts for Channel 1

First, disable all TPU interrupts by writing zeros to the channel interrupt enable register (CIER). Second, negate all TPU interrupt status flags by reading the channel interrupt status register (CISR) and then writing it to all zeros. Third, start the interrupt routine at the label INT by storing the address of INT in the appropriate vector address location. For this example, the base vector number $80 is chosen. This number is stored in the TPU interrupt configuration register (TICR). The actual interrupt vector number is calculated by concatenating the base vector with the channel number. Thus, the interrupt vector number is $81, since this program uses channel 1. The vector address (where the starting address of the interrupt routine is stored) is calculated as two times the vector number. In this case, the vector address is $2 * $81, which is equal to $102.

The interrupt level must be set to a non-zero value in the TICR. The interrupt level chosen determines the priority given to this interrupt by the CPU. Level 7 is the highest priority, and level 1 is the lowest. This example uses level 6. Once an interrupt level has been chosen, bits [6:4] in the CPU status register must be modified to allow recognition of that level of interrupt. These bits must be set to a number that is lower than the interrupt level number. Interrupts at the same level or lower than the number in the CPU status register will be masked out and will not be recognized by the CPU. In addition, the interrupt arbitration (IARB) field in the TPU module configuration register (TPUMCR) must be set to a value between $1 and $F. This IARB field determines the interrupt's priority if two or more modules with the same interrupt level request interrupts at the same time. In this example, the IARB field is set to $5.

```
          ldd       CIER
          andd      #$fffd
          std       CIER                     ;disable TPU interrupts on Channel 1
          ldd       CISR                     ;clear TPU interrupt requests for Channel 1
          andd      #$fffd
          std       CISR
          ldd       #INT
          std       $0102,z                  ;start interrupt routine at label INT
          ldd       TPUMCR
          ord       #$0005
          std       TPUMCR                   ;set IARB field
          ldd       #$0680
          std       TICR                     ;interrupt level 6, base vector = $80
          andp      #$ff5f                   ;allow interrupts on level 6 and above
```

8.4.6 Service Initialization Request

Initialize channels 0, 1 and 2 and enable interrupts for channel 1.

```
          ldd       #$0026
          std       HSRR1                    ;Initialization for ch 0, 1, 2
          ldd       CIER
          ord       #$0002
          std       CIER                     ;enable interrupts for channel 1
finish
          bra       finish
```

8.4.7 Interrupt Routine

This routine will be called each time TRANS_COUNT reaches MAX_COUNT. It will increment HIGH_TIME of the SPWM wave until it reaches $500. Then, it will reinitialize HIGH_TIME to $100.

```
INT
          ldd       CIER
          andd      #$fffd
          std       CIER                     ;disable interrupt in CIER
          ldd       CISR                     ;read interrupt flag
          andd      #$fffd
          std       CISR                     ;clear interrupt
          ldd       PRAM2_2
          cmpa      #$05                     ;compare HIGH_TIME to $500
          bne       skip
          tstb
          beq       chng
skip
          addd      #$0001
          std       PRAM2_2                  ;add $01 to HIGH_TIME of SPWM wave
          bra       done
chng
          ldd       #$0100
          std       PRAM2_2                  ;reset HIGH_TIME to #$100
```

```
done
          ldd       CIER
          ord       #$0002
          std       CIER                    ;enable interrupt for channel 1
          RTI                               ;Return from Exception
```

8.5 Example B

8.5.1 Description

The following program uses single shot with links mode to count input pulses and generate a link when MAX_COUNT reaches a pre-specified value. In single-shot mode with links, the ITC function counts the number of transitions programmed in MAX_COUNT once. When TRANS_COUNT reaches the value in MAX_COUNT, a link is generated to the channel specified by START_LINK_CHANNEL, and the value in the high byte of the parameter pointed to by BANK_ADDRESS is incremented by one. In this example, BANK_ADDRESS points to an unimplemented RAM location so that it does not affect operation of other channels.

In this program, the ITC function on channel 1 counts input pulses from the PWM function on channel 0. When the ITC function counts seven pulses, it generates a link to channel 2, which is set up to run the SPWM function. This simply means that channel 1 issues a service request to channel 2. In order to see when the link is generated, the SPWM square wave is programmed to be out of phase with the PWM square wave. The rising edge of the SPWM wave will begin at the falling edge of the PWM wave.

Channel 0 is set up to run the PWM function, channel 1 is set up to run the ITC function, and channel 2 is set up to run the SPWM function. Use the same equates as for example A.

8.6 Program Code for CPU32-Based Microcontrollers

This program was assembled using the IASM32 assembler available from P&E Microcomputer Systems with the M68332 In-Circuit Debugger. It was run on an M68332EVS and BCC.

8.6.1 Initialization

```
          org                               $4000;begin at memory location $4000
          move.w    #$07a9,(CFSR3).l        ;Channel Function Select Field (channel
                                            ;numbers may vary for different mask sets)
          move.w    #$00ff,(CPR1).l         ;Channel Priority Field, high priority
          move.w    #$0008,(HSQR1).l        ;ITC mode = single with links
                                            ;SPWM = mode 0
```

8.6.2 PWM Initialization for Channel 0

This PWM wave will have a pulse period of $1000 and a pulse hightime of $500. The ITC function on channel 1 will count the rising edges.

```
          move.w    #$0092,(PRAM0_0).l      ;Channel Control, use TCR1
          move.w    #$0500,(PRAM0_2).l      ;pulse hightime = 500
          move.w    #$1000,(PRAM0_3).l      ;pulse period = 1000
```

8.6.3 ITC Initialization for Channel 1

In this example, the ITC function only links to channel 2. Thus, START_LINK_CHANNEL = 2, and LINK_CHANNEL_COUNT = 1. As required, LINK_CHANNEL_COUNT is a value greater than zero and less than or equal to eight.

Since this program does not need to increment a parameter in another memory location when the number of transitions specified in MAX_COUNT has been counted, BANK_ADDRESS points to an unimplemented memory location.

```
          move.w    #$0007,(PRAM1_0).l      ;Channel control, detect rising edge, use
```

```
TCR1
          move.w    #$210e,(PRAM1_1).l     ;START_LINK_CHANNEL = 2,
                                           ;LINK_CHANNEL_COUNT = 1,
                                           ;BANK_ADDRESS points to unimplemented RAM
          move.w    #$0007,(PRAM1_2).l     ;MAX_COUNT = 7
```

8.6.4 SPWM Initialization for Channel 2 in Mode 0

The SPWM is set up in mode 0 so that it can receive links from another channel. It is initialized with a pulse hightime of $500 and a period of $1000. REF_ADDR1 points to a reference value to which DELAY and PERIOD are added to form the rising transition time. Here, it points to FINAL_TRANS_TIME on the ITC channel. FINAL_TRANS_TIME contains the TCR time of the final transition when MAX_COUNT is reached.

```
          move.w    #$92,(PRAM2_0).l       ;Channel Control
          move.w    #$500,(PRAM2_2).l      ;HIGH_TIME = $500
          move.w    #$1000,(PRAM2_3).l     ;PERIOD = $1000
          move.w    #$0018,(PRAM2_4).l     ;REF_ADDR1 = $18
          move.w    #$0500,(PRAM2_5).l     ;DELAY = $500
```

8.6.5 Service Initialization Request

```
          move.w    #$0026,(HSRR1).l       ;Initialization for ch 0, 1, 2
finish
          bra                              finish
```

8.7 Program Code for CPU16-Based Microcontrollers

This program was assembled on the IASM16 Assembler available with the ICD16 In-Circuit Debugger from P&E Microcomputer Systems and was run on an MC68HC16Y1EVB.

8.7.1 Initialization

The following code is included to set up the reset vector ($00000 – $00006). It may be changed for different systems.

```
          ORG       $0000                  ;put the following reset vector information
                                           ;at address $00000 of the memory map
          DW        $0000                  ;zk=0, sk=0, pk=0
          DW        $0200                  ;pc=200 -- initial program counter
          DW        $3000                  ;sp=3000 -- initial stack pointer
          DW        $0000                  ;iz=0 -- direct page pointer
          org       $0400                  ;begin program at memory location $0400
```

The following code initializes and configures the system including the Software Watchdog and System Clock. It was written to be used with an EVB.

```
INITSYS:                                   ;give initial values for extension regis-
ters
                                           ;and initialize system clock and COP
          LDAB      #$0F
          TBEK                             ;point EK to bank F for register access
          LDAB      #$00
          TBXK                             ;point XK to bank 0
          TBYK                             ;point YK to bank 0
          TBZK                             ;point ZK to bank 0
          TBSK
          LDD       #$0003                 ;at reset, the CSBOOT block size is 512K.
          STD       CSBARBT                ;this line sets the block size to 64K since
                                           ;that is what physically comes with the EVB16
          LDAA      #$7F                   ;w=0, x=1, y=111111
          STAA      SYNCR                  ;set system clock to 16.78 MHz
          CLR       SYPCR                  ;turn COP off, since COP is on after reset
          lds       #$f000
**** MAIN PROGRAM ****
```

```
          ldab      #$0f                    ;use bank $0f for parameter RAM
          tbek
          ldd       #$07a9
          std       CFSR3                   ;Channel Function Select Field (Note: func-
tion
          ldd       #$00ff                  ;numbers may vary for different mask sets)
          std       CPR1                    ;Channel Priority Field, high priority
          ldd       #$0008
          std       HSQR1                   ;ITC mode = single with links, SPWM = mode 0
```

8.7.2 PWM Initialization for Channel 0

This PWM wave will have a pulse period of $1000 and a pulse hightime of $500. The ITC function on channel 1 will count the rising edges.

```
          ldd       #$0092
          std       PRAM0_0                 ;Channel Control, use TCR1
          ldd       #$0500
          std       PRAM0_2                 ;pulse hightime = 500
          ldd       #$1000
          std       PRAM0_3                 ;pulse period = 1000
```

8.7.3 ITC Initialization for Channel 1

In this example, the ITC function only links to channel 2. Thus, START_LINK_CHANNEL = 2, and LINK_CHANNEL_COUNT = 1. As required, LINK_CHANNEL_COUNT is a value greater than zero and less than or equal to eight.

Since this program does not need to increment a parameter in another memory location when the number of transitions specified in MAX_COUNT has been counted, BANK_ADDRESS points to an unimplemented memory location.

```
          ldd       #$0007
          std       PRAM1_0                 ;Channel control, detect rising edge, use
TCR1
          ldd       #$210e
          std       PRAM1_1                 ;START_LINK_CHANNEL = 2,
                                            ;LINK_CHANNEL_COUNT = 1,
                                            ;BANK_ADDRESS points to unimplemented RAM
          ldd       #$0007
          std       PRAM1_2                 ;MAX_COUNT = 7
```

8.7.4 SPWM Initialization for Channel 2 in Mode 0

The SPWM is set up in mode 0 so that it can receive links from another channel. It is initialized with a pulse hightime of $500 and a period of $1000. REF_ADDR1 points to a reference value to which DELAY and PERIOD are added to form the rising transition time. Here, it points to FINAL_TRANS_TIME on the ITC channel. FINAL_TRANS_TIME contains the TCR time of the final transition when MAX_COUNT is reached. This waveform will be delayed from the PWM waveform. Its rising edge will occur at the falling edge of PWM.

```
          ldd       #$92
          std       PRAM2_0                 ;Channel Control
          ldd       #$500
          std       PRAM2_2                 ;HIGH_TIME = $500
          ldd       #$1000
          std       PRAM2_3                 ;PERIOD = $1000
          ldd       #$0018
          std       PRAM2_4                 ;REF_ADDR1=$18
          ldd       #$0500
          std       PRAM2_5                 ;DELAY = $500
```

8.7.5 Service Initialization Request

```
          ldd       #$0026
          std       HSRR1                    ;Initialization for ch 0, 1, 2
finish
          bra       finish
```

9 Function Algorithm

At each transition detected, the TPU increments TRANS_COUNT and updates LAST_TRANS_TIME to the value of the TCR. 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) the parameter addressed by BANK_ADDRESS is incremented, 3) an interrupt is asserted causing an interrupt to be generated (if interrupt enable bit is set), 4) if the time function is in link mode, links to a sequential block of channels are generated as specified by START_LINK_CHANNEL and LINK_CHANNEL_COUNT parameters, and 5) if the function is in continuous mode, the TPU clears TRANS_COUNT and continues to count.

The ITC algorithm is described in the following paragraphs. The following description is provided as a guide only, to aid understanding of the function. The exact sequence of operations in microcode may be different from that shown, in order to optimize speed and code size. TPU microcode source listings for all functions in the TPU function library can be downloaded from the Freescale Freeware bulletin board. Refer to *Using the TPU Function Library and TPU Emulation Mode* (TPUPN00/D) for detailed instructions on downloading and compiling microcode.

9.1 State 1: *Init*

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 01xxxx

Match Enable: Don't Care A.2.1 State 1 Init

Summary:

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.

Algorithm:

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

9.2 State 2: *Count_Up*

Condition: HSR1, HSR0, M/TSR, LSR, Pin, Flag0 = 001xxx

Match Enable: Don't Care

Summary:

This state is entered as a result of a transition. TRANS_COUNT is incremented and if TRANS_COUNT is less than MAX_COUNT, LAST_TRANS_TIME is updated to contain the time of the last transition. If TRANS_COUNT is greater than or equal to MAX_COUNT, then:

- FINAL_TRANS_TIME is updated to contain the time of the last transition, and an interrupt request is asserted.
- The byte referenced by BANK_ADDRESS is incremented. If this increment is not desired, BANK_ADDRESS should contain a nonexistent address (such as $0E).

- 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.

Algorithm:

```
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]
    }
    (BANK_ADDR) = (BANK_ADDR) + 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 ITC transitions listing the service request sources and channel conditions from current state to next state. **Figure 4** illustrates the flow of ITC states.

Table 3 ITC State Transition Table

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

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

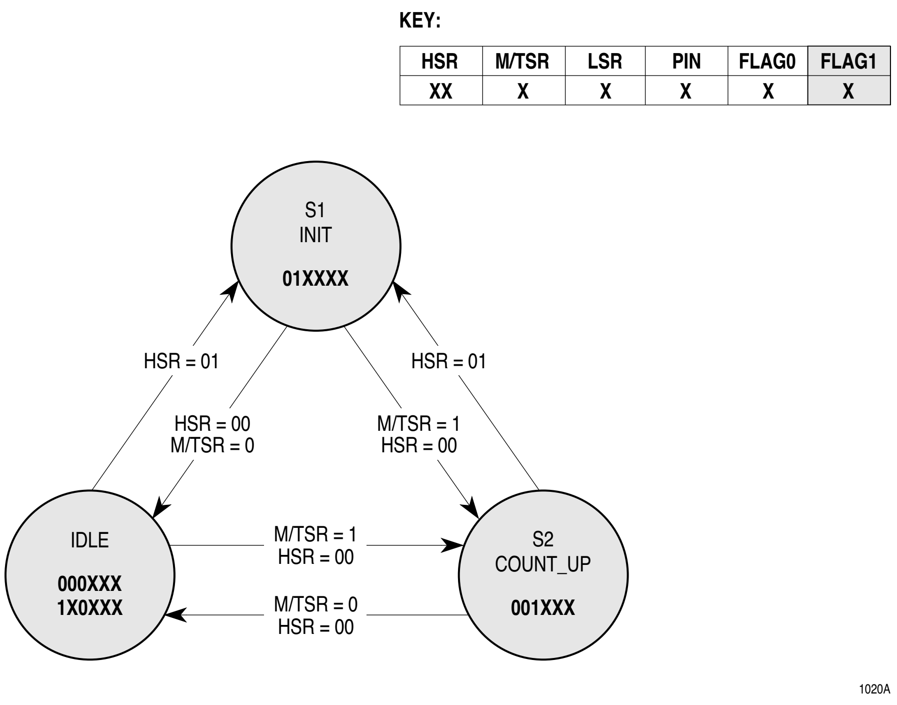


Figure 4 ITC State Flowchart

freescale™ semiconductor