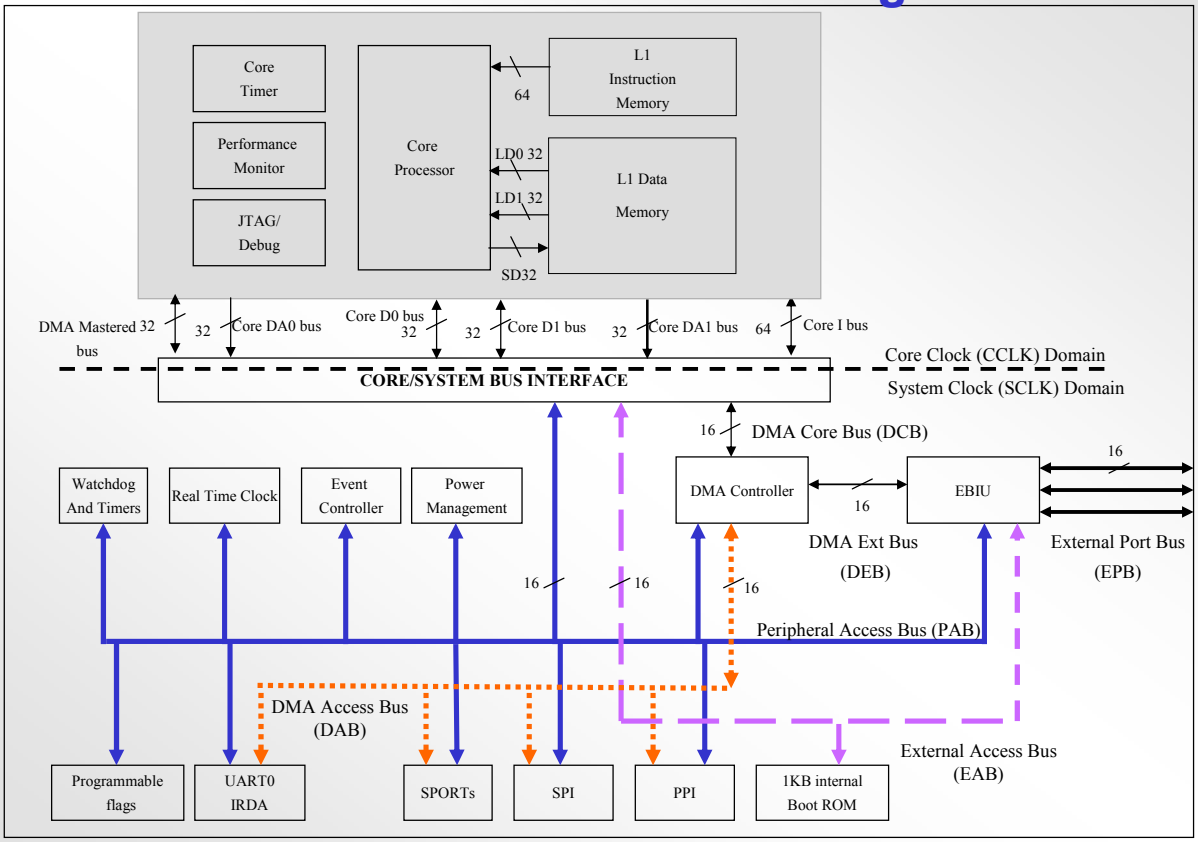
Section 10

Timers
and
Programmable Flags

ADSP-BF533 Block Diagram



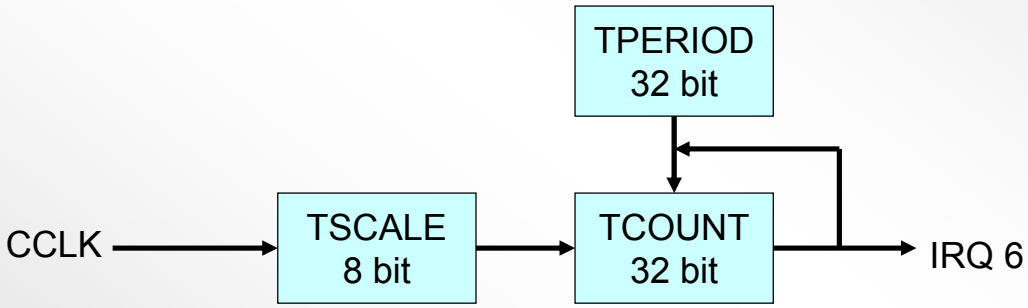
ADSP-BF533 Timers

- Six Timers
  - One Core Timer
  - One Watchdog Timer
  - One Real-Time Clock (RTC)
  - Three Identical 32-bit General Purpose Timers
    - Pulse Capture Mode
    - PWM Mode
    - External Clock Mode

Core Timer

Core Timer

- **Generates interrupts at multiples of CCLK rate**
  - **32-bit tick timer**
- **Dedicated Interrupt Priority 6 (IRQ6 = IVTMR)**
- **Optional Auto-Reload Feature**

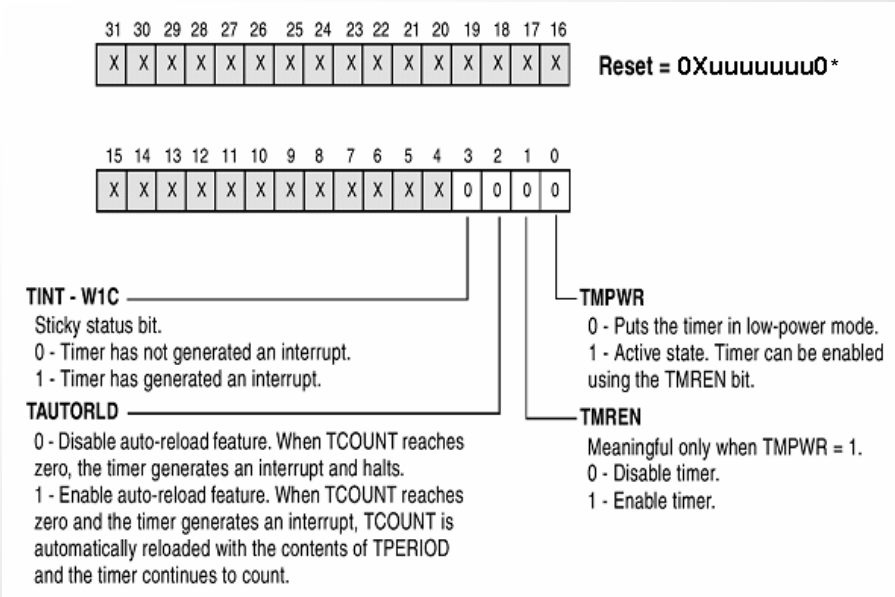


Interrupt Rate = CCLK / [(TSCALE + 1) x TPERIOD]

Core Timer

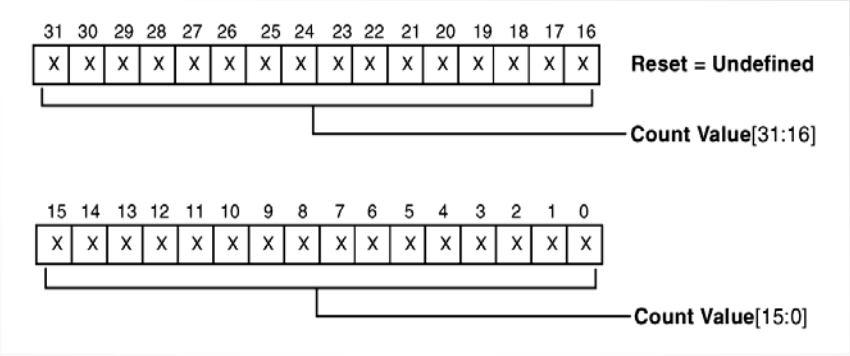
- 3 Bits Enable The Core Timer To Generate Interrupts
  - TMPWR, bit 0 in Core Timer Control Register (TCNTL)
    - Provides Power To Core Timer
  - TMREN, bit 1 in the TCNTL, Enables Core Timer
  - IVTMR, bit 6 in the IMASK register, Enables Core Timer Interrupt
- Core Timer Is Halted In Emulator Mode
- Timer Scale Register (TSCALE) Controls How Often Timer Count Register (TCOUNT) Is Decremented
  - TCOUNT decrements once every TSCALE+1 CCLK cycles
- When TCOUNT Decrements to Zero
  - Interrupt is generated
    - Write 1 to TINT (bit 3 in TCNTL) to clear interrupt
  - If auto-reload is enabled (TAUTORLD, bit 2 in TCNTL)
    - TCOUNT reloaded with Timer Period Register (TPERIOD)

Core Timer Control Register (TCNTL)

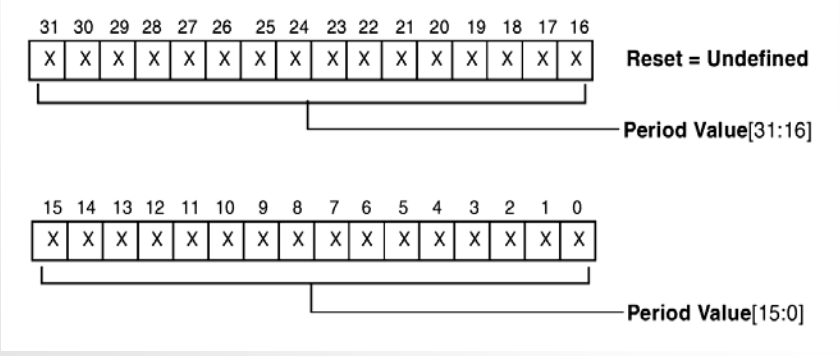


** - Bits 0-3 are 0 at reset, the rest are undefined.*

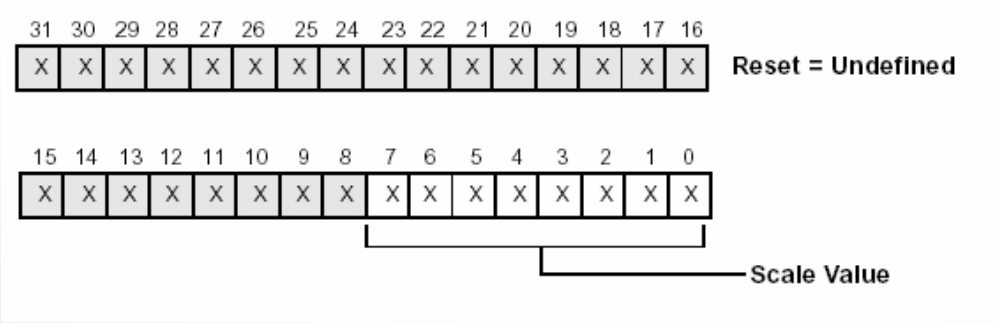
Core Timer Count (TCOUNT) Register



Core Timer Period (TPERIOD) Register



Core Timer Scale Register (TSCALE)



*** Determines How Many CCLK Cycles Per TCOUNT Decrement**

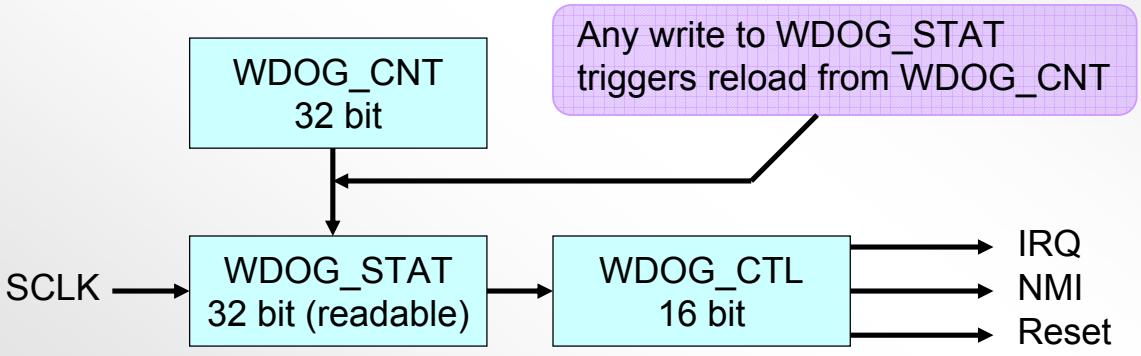
Watchdog Timer

ADSP-BF533 Watchdog Timer

- Peripheral Timer Clocked By The System Clock (SCLK)
- Used To Improve System Reliability By Generating An Event When The Timer Expires Before Being Updated By Software
- Type Of Event Generated By Watchdog Is Programmable In Watchdog Control (WDOG_CTL) Register
  - Reset (software reset takes place)
  - NMI (non-maskable interrupt occurs)
  - General Purpose Interrupt (IVG13, by default)
- Enable Watchdog Timer To Generate Interrupts
  - Set Software Watchdog Timer Interrupt Bit (bit 20 in SIC_IMASK)
  - Set IVG13 (bit13 in IMASK register), by default
    - When interrupt priority is reassigned, the IVGx bit changes
  - Enable timer and type of event generated in WDOG_CTL

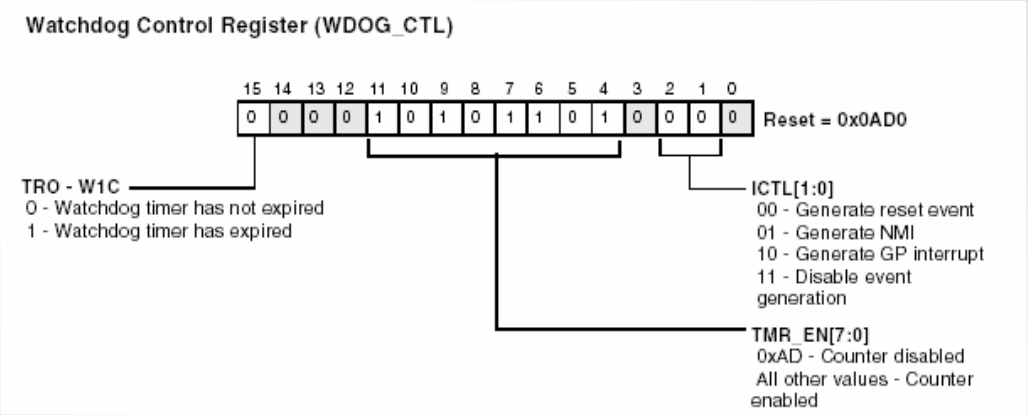
ADSP-BF533 Watchdog Timer

- **Must be periodically serviced by software**
- **Unique 8-bit pattern (0xAD) required to disable watchdog timer**
- **W1C bit in WDOG_CTL (bit 15, TRO) indicates expiration**
  - **Watchdog MUST be disabled to clear TRO bit!**
- **Watchdog Is Halted In Emulator Mode**



Watchdog Interval = WDOG_CNT / SCLK

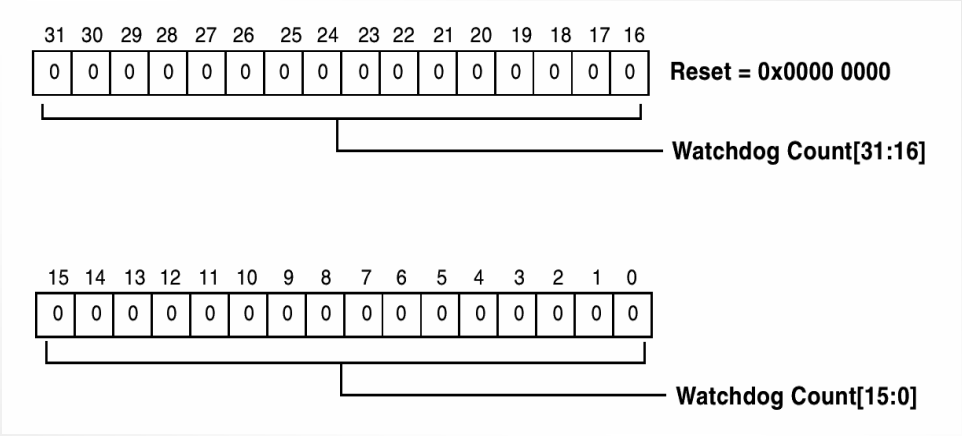
Watchdog Control Register (WDOG_CTL)



When ICTL[1:0] is set to generate a reset event, bit 15 of the WDOG_CTL register is cleared and the Software Watchdog Timer Source bit in the SWRST register is set when the watchdog timer expires. The SWRST register is discussed further in the System Design chapter.

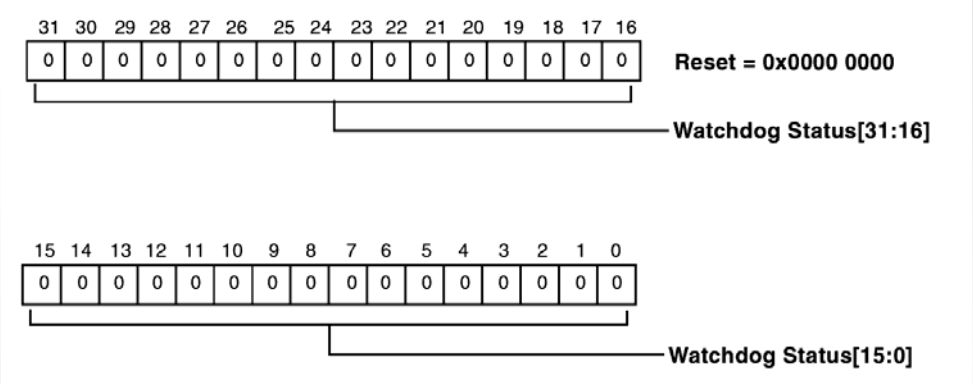
When ICTL[1:0] is set to generate an NMI or GP interrupt, bit 15 of the WDOG_CTL register is sticky and must be cleared in the Interrupt Service Routine when the watchdog timer expires.

Watchdog Count Register (WDOG_CNT)



* WDOG_CNT register is number of SCLK cycles to count before generating event

Watchdog Status Register (WDOG_STAT)



* Reads Of WDOG_STAT Return Current Count Value

* Writes To WDOG_STAT Reload WDOG_STAT With WDOG_CNT Value

Real Time Clock

Real-Time Clock Features

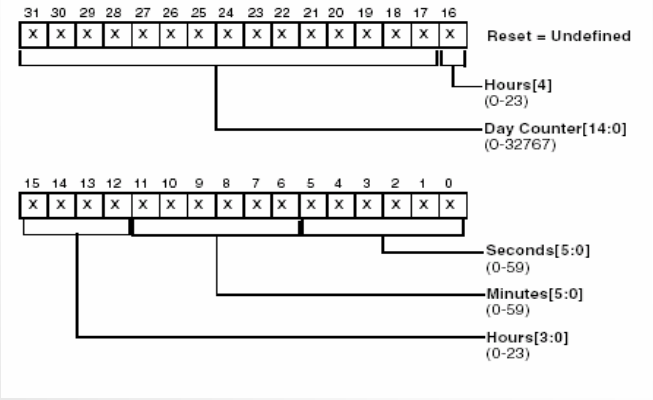
- Provides digital watch features to the processor
  - Time of day, alarm, and stopwatch count-down
- Typically used to implement real-time watch or “life counter”
  - Counts the elapsed time since last system reset
- Uses dedicated power supply pins
  - Independent of any reset
  - Maintains functionality even when rest of processor is powered down
- Primary function is to maintain time of day using 4 counters
  - Seconds, Minutes, Hours, Days
    - Each of the 4 counters can generate an interrupt
    - Each interrupt is independently controlled
- Equipped With Two Alarm features
  - Daily and Day-And-Time
  - Each has its own independently-controlled interrupt

ADSP-BF533 Real-Time Clock (RTC)

- Independent Clock and Power Supply
  - Connect 32.768 kHz crystal to RTXI / RTXO
  - Clock Can Be Set To 1 Hz to Count Time and Days
    - Enabled by setting pre-scalar (bit 0 in RTC_PREN)
    - Otherwise, clock is 32768 Hz

RTC_STAT

RTC_ALARM



  - RTC cannot be disabled by software

ADSP-BF533 RTC Interrupts

- RTC Interrupt May Be Issued Based On 7 Different Events
  - Interval interrupts
    - Every Second, Minute, Hour, and/or Day
  - Alarm interrupts (single or daily)
    - Single occurs when RTC_ALARM matches RTC_STAT
    - Daily recurs when all non-day fields match
  - Stopwatch interrupt
    - 16-bit counter (RTC_SWCNT) decrements every RTC tick
      - Period of up to 18 hours, 12 minutes, and 15 seconds
    - Interrupt when RTC_SWCNT reaches zero, no autoreload
- RTC Interrupt Can Be Used To Wake Processor From Sleep Modes

RTC Wake-Up Event

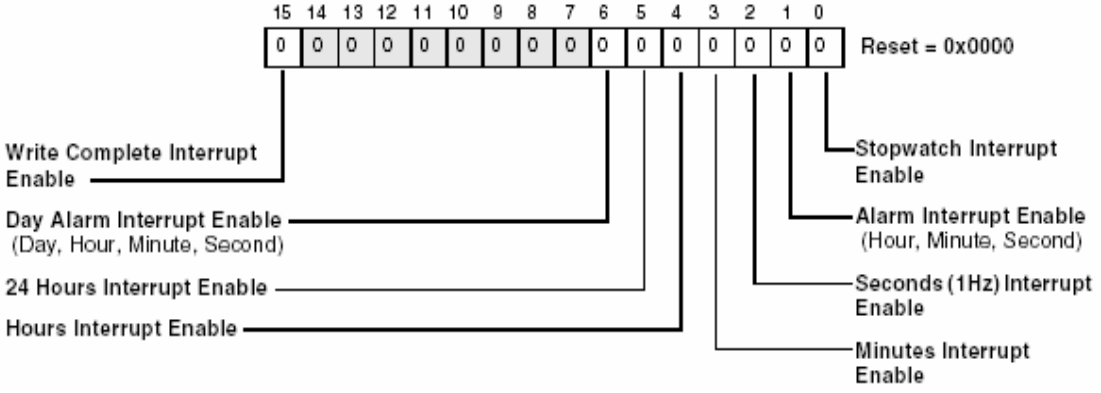
- **Set RTC Wake-Up Event Bit In SIC_IWR**

```
P0.H = HI (SIC_IWR);
P0.L = LO (SIC_IWR);
R0 = [P0];
BITSET (R0, 7);  // Bit 7 is RTC Wake-Up Enable Bit
[P0] = R0;
```

- **Configure RTC and Enable RTC Interrupt**
  - **Wake-Up Can Come From Either Stopwatch OR Alarm Event**
- **Program PLL To Put Processor Into Sleep or Deep Sleep Mode**
- **Upon Wake-Up (Stopwatch Expired or Alarm Active), the ISR:**
  - **Writes RTC_ISTAT to acknowledge RTC wake-up**
  - **If Coming From Sleep Mode, No Further Action Is Required**
  - **If Coming From Deep Sleep Mode**
    - **PLL state is now "Active" (PLL is Bypassed)**
    - **If transition to "Full On" is needed, the BYPASS bit in PLL_CTL must also be cleared**

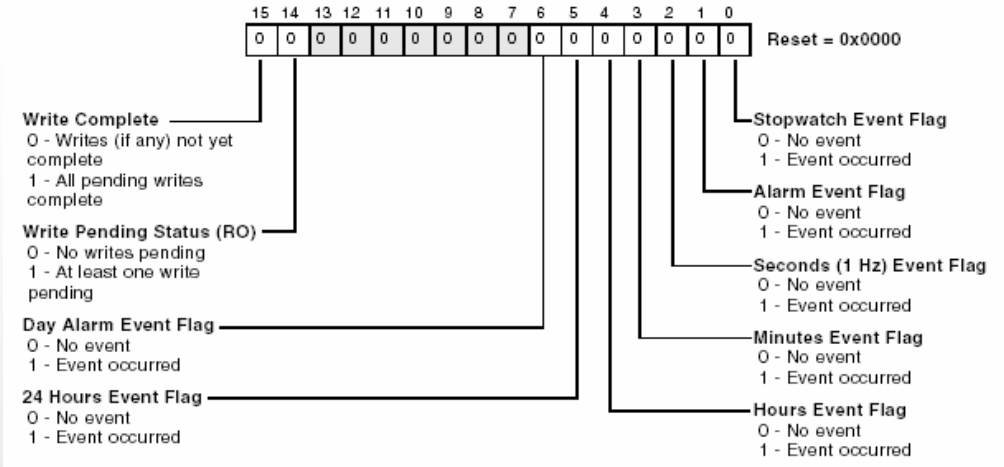
Real Time Clock Interrupt Control Register

RTC Interrupt Control Register (RTC_ICTL)
0 - Interrupt disabled, 1 - Interrupt enabled



Real Time Clock Interrupt Status Register

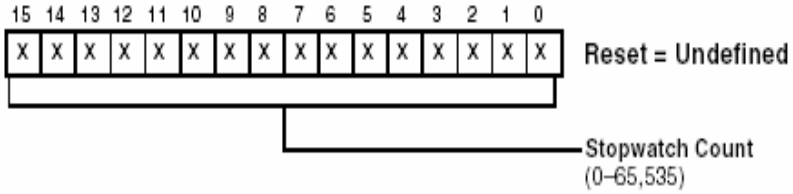
RTC Interrupt Status Register (RTC_ISTAT)
All bits are write-1-to-clear, except bit 14.



*** Any writes to RTC registers pend until the expiration of a RTC tick. Polling bit 15 for a 1 or bit 14 for a 0 will ensure that the RTC state is that which has just been programmed.**

Real Time Clock Stopwatch Count Register

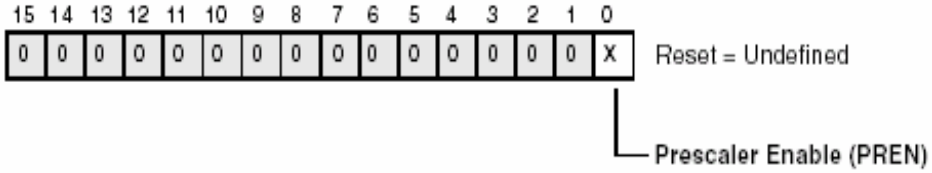
RTC Stopwatch Count Register (RTC_SWCNT)



The count value in this register applies to seconds

Real Time Clock Prescaler Enable Register

Prescaler Enable Register (RTC_PREN)



When bit 0 is set, the RTC runs at 1 Hz.

When bit 0 is cleared, the RTC runs at 32.768 kHz.

General Purpose Timers

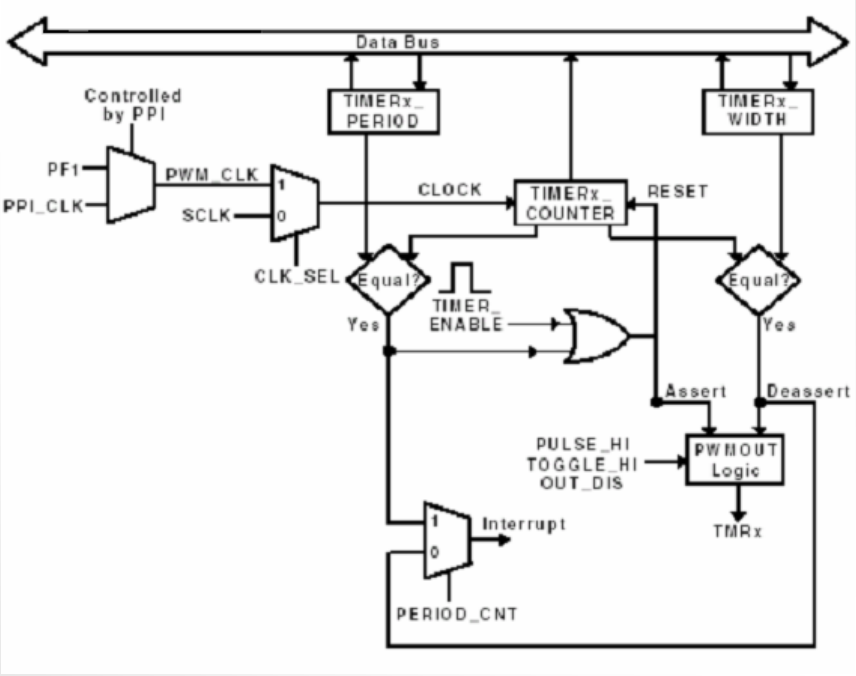
Three Peripheral Timers of the ADSP-BF533

- Three Identical Timers Can Be Configured In 3 Modes
  - Pulse Width Modulation (PWM_OUT)
  - Width and Period Capture (WDTH_CAP)
  - External Event Counter (EXT_CLK)
- Dedicated Pins TMR2, TMR1, TMR0
- One Programmable Interrupt Each
- Three 32-bit Registers Each
  - Width (TIMERx_WIDTH)
  - Period (TIMERx_PERIOD)
  - Counter (TIMERx_COUNTER) (read-only)
- One 16-bit Configuration Register Each (TIMERx_CONFIG)
- Three Common Registers Affect All 3 Timers Simultaneously
  - Timer Enable
  - Timer Disable
  - Timer Status (Interrupt requests, overflows, slave enables)

PWM_OUT Mode

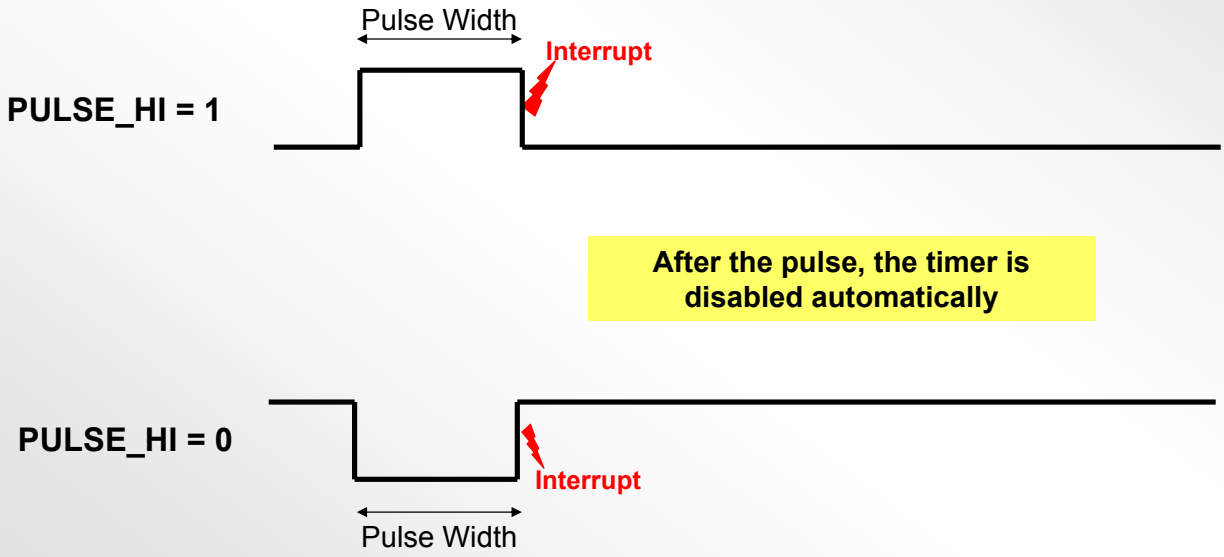
- Timers clocked from one of three sources
  - Internally using SCLK
  - Internally by PPI Clock when PPI is enabled
  - Externally using PF1 pin when PPI is disabled
- TMRx pins are outputs by default
  - Can be tristated using OUT_DIS in TIMERx_CONFIG, which reduces power consumption when the output signal is not in use
- Two PWM_OUT modes
  - PWM Waveform Generation (PERIOD_CNT = 1)
    - Interrupt when Period expires
    - Use this mode for generating periodic interrupts
  - Single Pulse Generation (PERIOD_CNT = 0)
    - Stops and disables itself after first Pulse Width expires
    - Interrupt when Pulse Width expires
    - Use this mode for programmable software delay

PWM_OUT Mode Block Diagram



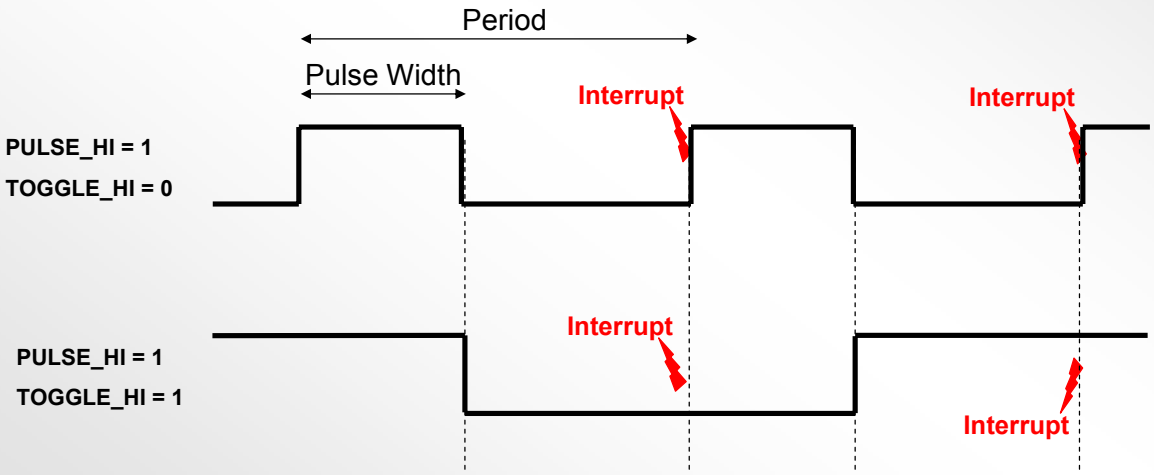
Single Pulse Generation

- **Single Pulse Generation** (PERIOD_CNT = 0)
  - TOGGLE_HI has no effect
  - PULSE_HI determines active state of single pulse



Waveform Generation

- **PWM Waveform Generation** (PERIOD_CNT = 1)
  - TMRx pin polarity is programmable (PULSE_HI bit)
  - Toggle Waveform Every Period or Every Other Period (TOGGLE_HI bit)
  - Period and Pulse Width adjustable
  - On-the-fly Update Procedure

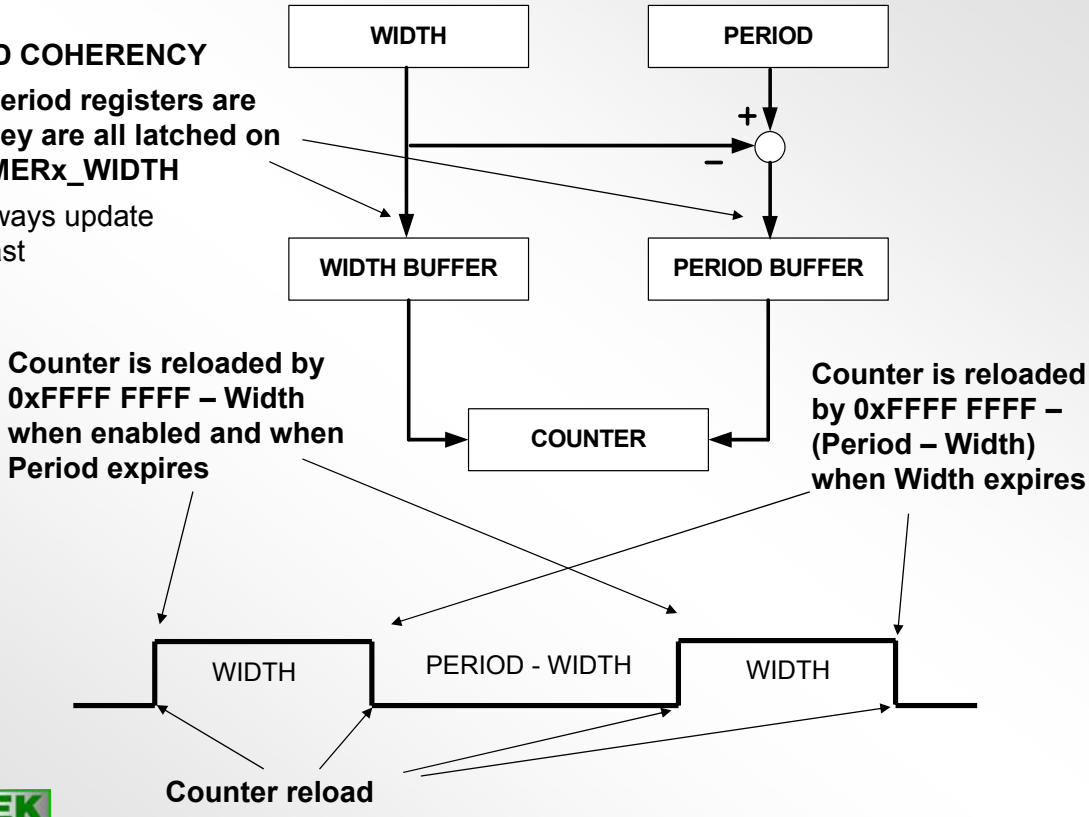


PWM Mode Timer Reload

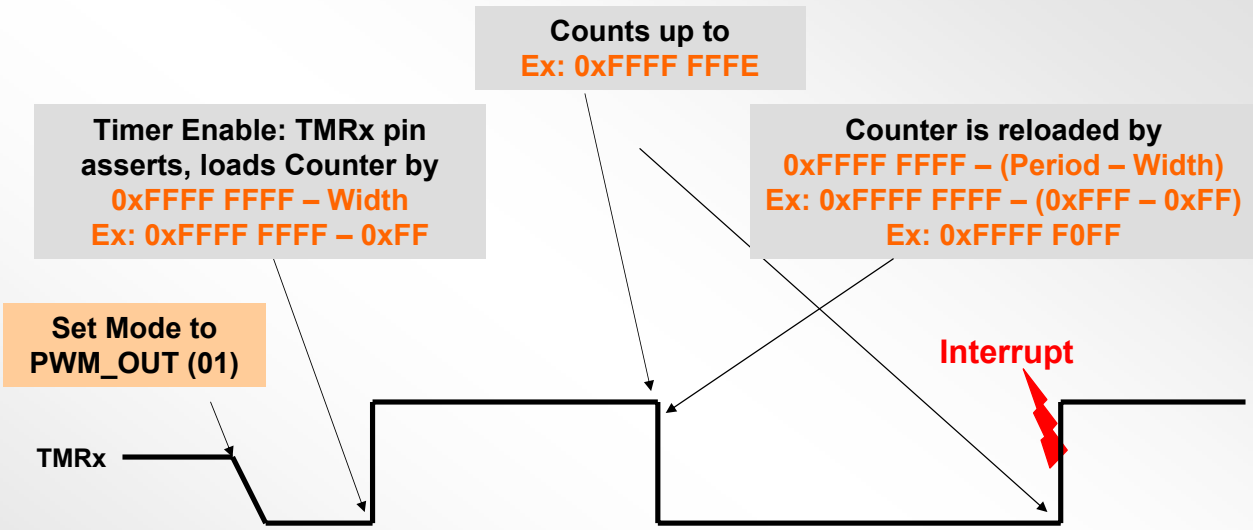
PROTECTED COHERENCY

Width and Period registers are buffered. They are all latched on writes to TIMERx_WIDTH

Therefore always update width word last



PWM_OUT Mode Example



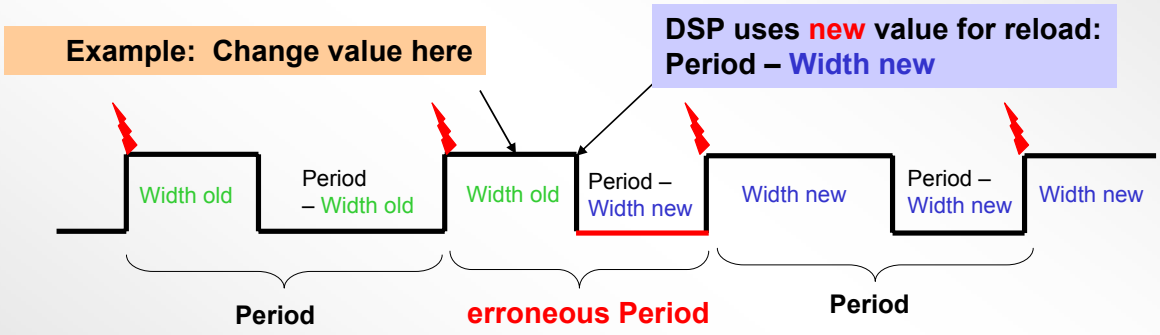
Example:
Width=0xFF
Period=0xFFF

TMRx Pin is driven actively while the Mode is 01. It's driven low when Timer is disabled

Example assumes PULSE_HI = 1, PERIOD_CNT=1, TOGGLE_HI = 0

Updating PWM Waveform On The Fly

Exercise: change Width value and keep Period value



* Width value is latched on the rising edge

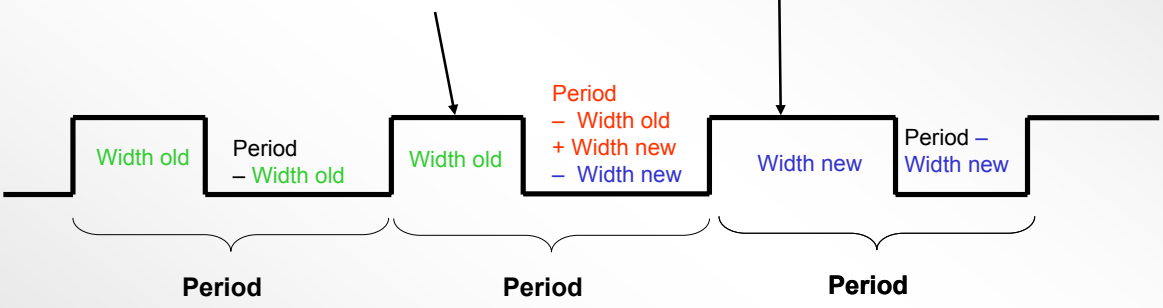
* Period value is latched on the falling edge

On-the-fly Update Procedure

When updating width value, use this procedure to prevent erroneous periods

Width reg = Width new
Period reg = (Period - Width old + Width new)

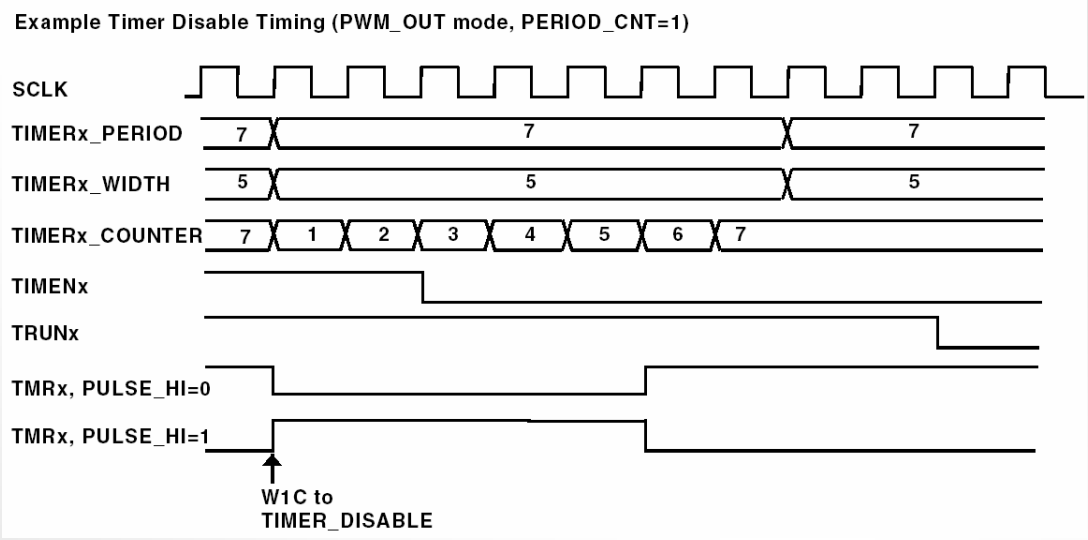
Change Period register back to old Period value



When updating period value only

1) Simply reload TIMERx_PERIOD
2) Read and write back TIMERx_WIDTH

Disabling the Timer

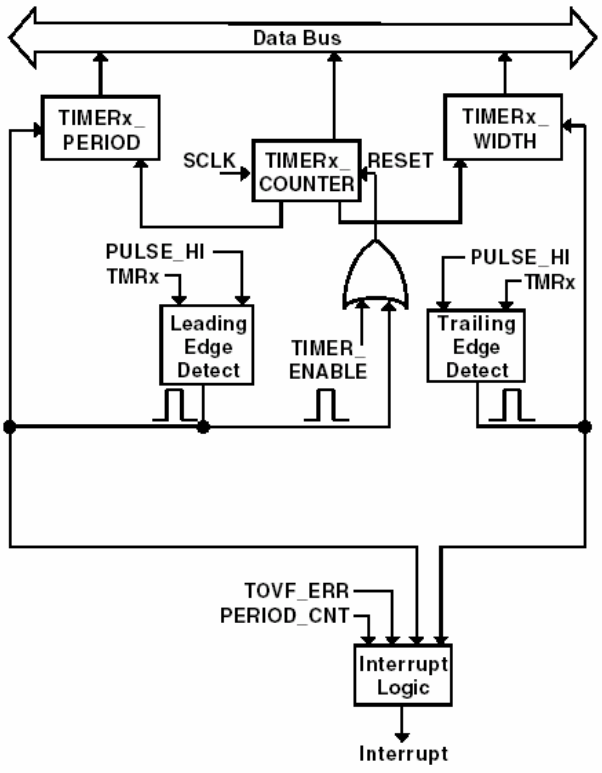


- In PWM_OUT mode, timer will run to the end of a cycle if TIMER_DISABLE bit set
  - TIMENx will reflect disable, but TRUNx indicates timer is running
  - Timer can be stopped immediately by clearing TRUNx bit

WDTH_CAP Mode

- Timers are clocked internally (SCLK)
- TMRx pins are inputs
  - PULSE_HI (active high or low)
  - TIN_SEL (capture TMRx or UART RX)
- Period and Width are counted at the same time
- Flexible Interrupt generation
  - PERIOD_CNT (End of Width or end of Period)
- Period and Width registers are read-only
- To enable (synchronization latency)
  - Set to WDTH_CAP mode
  - Add SSYNC
  - Set TIMENx in TIMER_ENABLE (timer starts 3 SCLK cycles later)

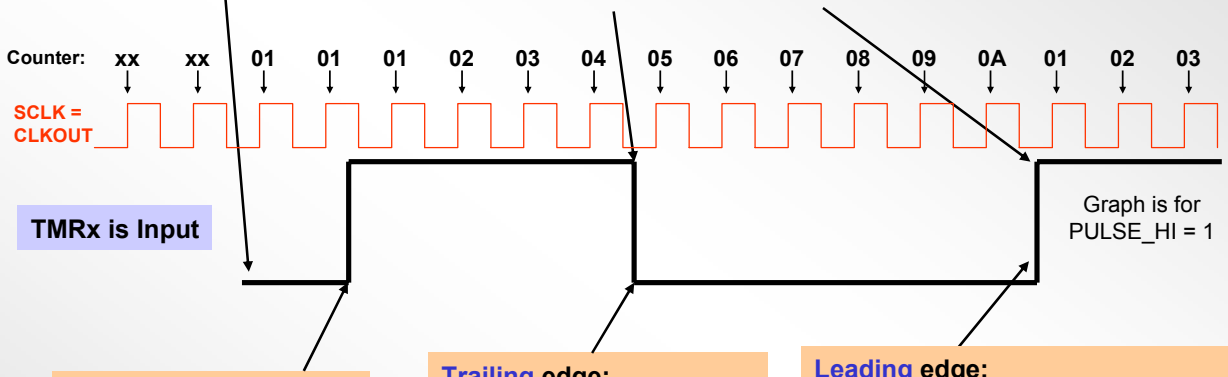
WDTH_CAP Mode Block Diagram



WDTH_CAP Mode Example

Timer enable:
Counter loaded by 0x0000 0001

Interrupt is generated:
PERIOD_CNT = 0: End of Width
PERIOD_CNT = 1: End of Period



First Leading edge:
Timer starts counting

Trailing edge:
Width register is loaded with Count value (0x04)

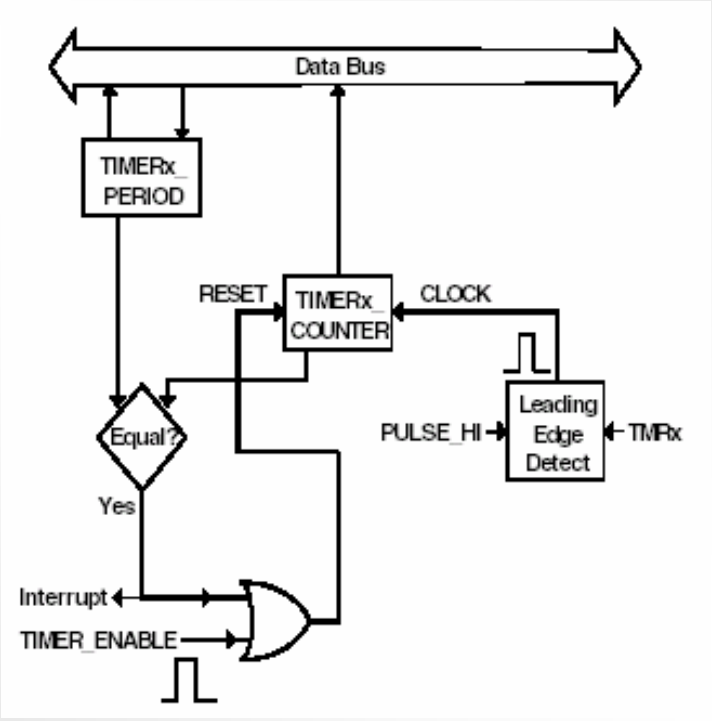
Leading edge:
1) Period register is loaded with Count value (0x0A)
2) Counter resets to 0x0000 0001

When counter reaches 0xFFFF FFFF:
Interrupt issued
Overflow bit set
Timer stopped / disabled

EXT_CLK Mode

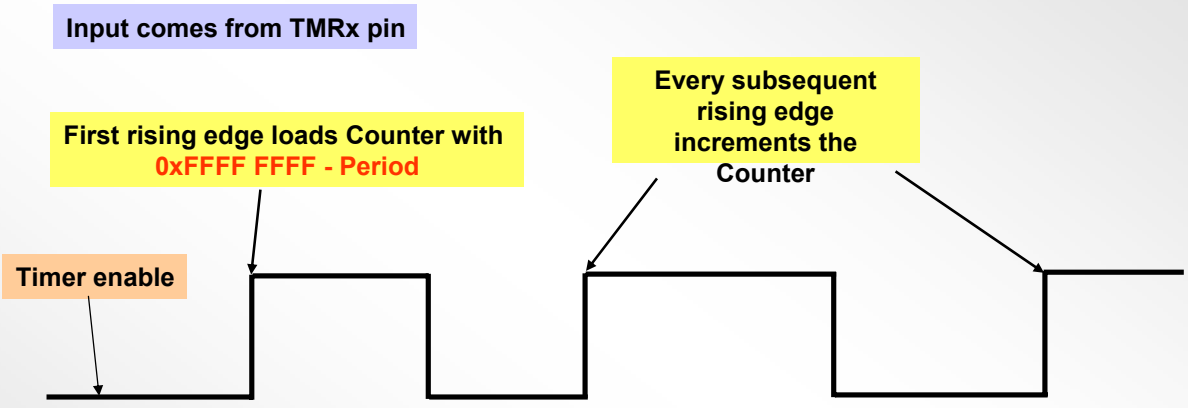
- External Clock through input TMRx Pins
- Clock counts rising or falling edges
- Width Register is unused
- Period Register is Read/Write
- When Period expires, interrupt or wake-up event is generated (if enabled)

EXT_CLK Mode Block Diagram



EXT CLK Mode

Load Period with number of rising clock edges between interrupts



Unused bits / registers:
- TIMERx_WIDTH
- OVF_ERRx
- PERIOD_CNT
- UART_RX_SEL
- TOGGLE_HI

When reaching count value 0xFFFF FFFE
an Interrupt is generated
Counter reloads to 0xFFFF FFFF - Period

Example assumes PULSE_HI=1

BF533 32 Bit Timer Registers

TIMER0_COUNTER	0xFFC0 0604
TIMER1_COUNTER	0xFFC0 0614
TIMER2_COUNTER	0xFFC0 0624
TIMER0_PERIOD	0xFFC0 0608
TIMER1_PERIOD	0xFFC0 0618
TIMER2_PERIOD	0xFFC0 0628
TIMER0_WIDTH	0xFFC0 060C
TIMER1_WIDTH	0xFFC0 061C
TIMER2_WIDTH	0xFFC0 062C

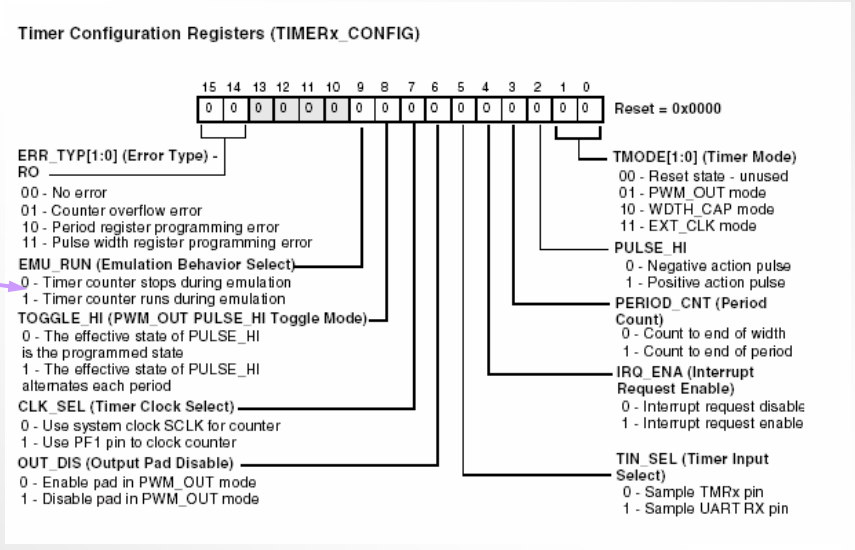
When disabled, the Counter retains its state. Enabling the Timer initializes the Counter.

BF533 Timer Configuration Register

TIMER0_CONFIG 0xFFC0 0600
TIMER1_CONFIG 0xFFC0 0610
TIMER2_CONFIG 0xFFC0 0620

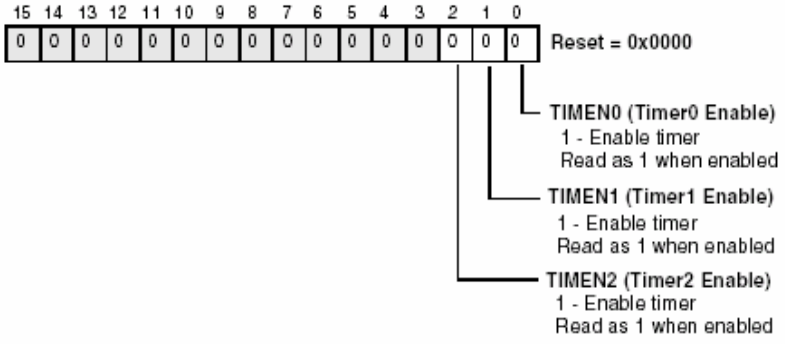
Do not alter while timer is enabled!

Timers can be configured to stop or continue running during emulation



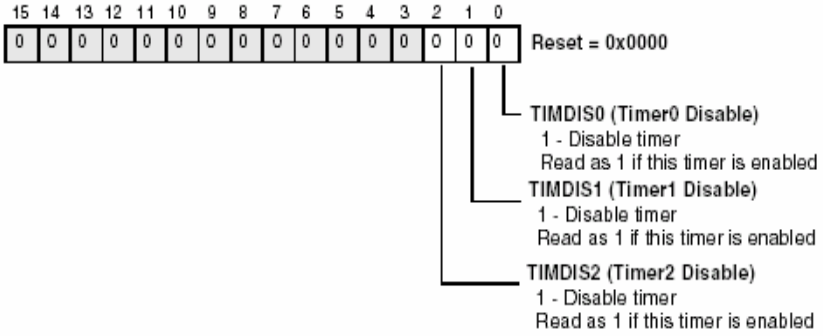
BF533 Timer Enable Register

Timer Enable Register (TIMER_ENABLE)



BF533 Timer Disable Register

Timer Disable Register (TIMER_DISABLE)



If in PWM_OUT mode, timer will continue running until end of PWM pulse after write to this register

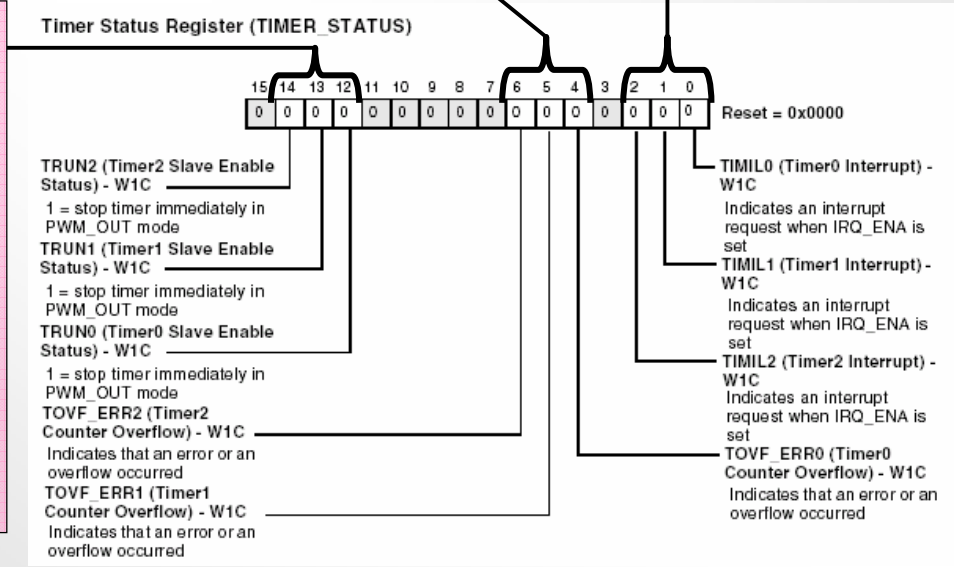
BF533 Timer Status Register

TIMERx Overflow and Error (TOVF_ERRx)
- Write-1-to-Clear
- Set when Counter Overflows or Period / Width registers are initialized with erroneous values
- Error type is accessible in TIMERx_CONFIG

TIMERx Interrupt Latch (TIMILx)
- Write-1-to-Clear
- Must be cleared by Timer ISR

TIMERx Slave Enable Status (TRUNx)

- Write-1-to-Clear in PWM_OUT mode only
- In PWM_OUT mode, remains set until period ends, even if TIMERx is disabled before the period expires
- In WDTH_CAP and EXT_CLK modes, it is identical to TIMENx
- Indicates whether or not TIMERx is running



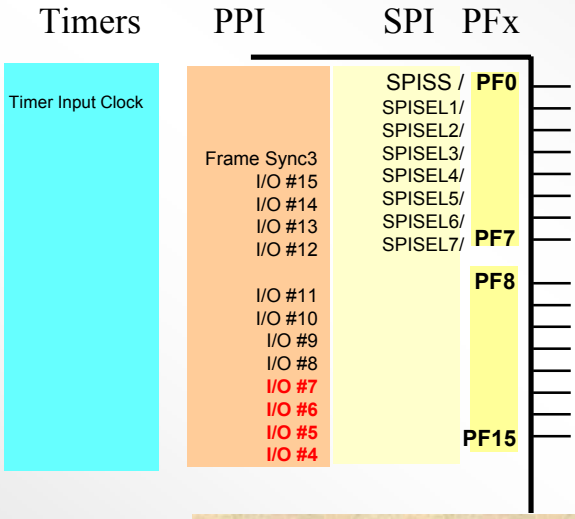
ADSP-BF533 Programmable Flags

Programmable Flags

- **Features:**
  - 16 bi-directional general purpose programmable flags
    - Each flag pin can be configured as Input or Output
  - Two independent interrupt channels (A/B)
    - Level or edge sensitive trigger of input source
    - Rising or falling edge trigger of input source
    - Single edge or both edges trigger of input source

Programmable Flag Pins/Multiplexed Functionality

16 bi-directional general-purpose I/O pins available
Each can be configured as an **output**, **input**, or an **interrupt** pin



Two Interrupt Requests (FLAGA/FLAGB)

When PPI is enabled PF12-15 are always enabled as PPI I/O data lines.
Other multiplexed pins are enabled by writing to the function's registers that use these pins, e.g., **SPI_FLG**, **PPI_CTL**, and **TIMERx_CONFIG** registers for **SPI**, **PPI**, and **Timers 0,1,2** functionality, respectively.

PFx Control Registers

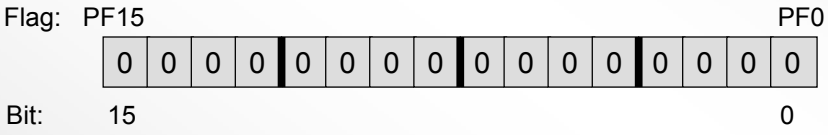
- **PFx are programmed with a group of flag configuration registers**
  - **Flag Direction register *(FIO_DIR)***
  - **Flag Input Enable Register *(FIO_INEN)***
  - **Flag Interrupt Sensitivity *(FIO_EDGE)***
  - **Flag Set on Both Edges Register *(FIO_BOTH)***
  - **Flag Polarity register *(FIO_POLAR)***
  - **Flag Control registers**
    - ***Clear and Set (FIO_FLAG_C and FIO_FLAG_S)***
    - ***Data and Toggle (FIO_FLAG_D and FIO_FLAG_T)***
  - **Flag Interrupt Mask Registers *(FIO_MASKA_C, FIO_MASKA_S, FIO_MASKA_D, FIO_MASKA_T FIO_MASKB_C, FIO_MASKB_S, FIO_MASKB_D, FIO_MASKB_T)***

Flag Direction Register (FIO_DIR)

Configures a flag pin as an *Input* or *Output*

Each bit of the FIO_DIR register corresponds with each of the 16 available flag pins

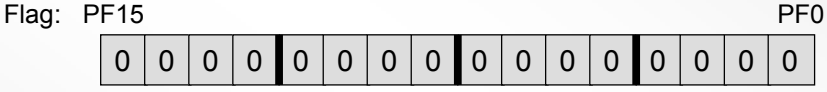
Reset = 0x0000



- Writing a "**1**" to a bit of the FIO_DIR register configures the corresponding flag pin as an **output**;
- Writing a "**0**" configures the corresponding flag pin as an **input**

Flag Input Enable Register *(FIO_INEN)*

The **Flag Input Enable Register** enables input buffers on any flag pin that is being used as an input.



A "**0**" leaves the Input Buffer Disabled <==> disconnecting external pads

A "**1**" leaves the Input Buffer Enabled

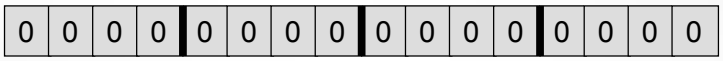
When input buffers are Disabled, the processor samples the pads as zeros, therefore no pull up or pull down resistors are needed

If ALL input buffers are Disabled, then software can set values in the FIO registers and read them back, **however,** if any input is enabled then a Zero will be read back on the flag block for all input flags.

Flag Interrupt Sensitivity (FIO_EDGE) and Flag Set on Both Edges Register (FIO_BOTH)

Flag Interrupt Sensitivity Register (FIO_EDGE) specifies the Flag pins for either level or edge sensitivity.

Flag: PF15 PF0

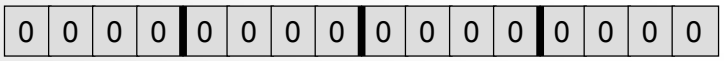


FIO_EDGE

A "**0**" configures the corresponding Flag Pin as a **level sensitive**, a "**1**" as an **edge sensitive** input.

Flag Set on Both Edges Register (FIO_BOTH) configures sensitivity for either one or both edges**.**

Flag: PF15 PF0



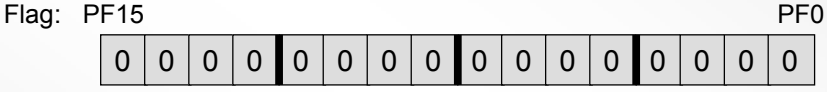
FIO_BOTH

A "**1**" configures the corresponding Flag Pin for **both-edges sensitivity**. An interrupt request is generated on each edge.

A "**0**" configures Flag Pin as **rising**- **or falling**-edge sensitivity (determined by the value in **FIO_POLAR**).

Flag Polarity Register (FIO_POLAR)

The **Flag Polarity Register** selects an active high or low polarity of an input signal.



A "**0**" configures the corresponding flag pin as **active high** or **rising edge** input.

A "**1**" configures the corresponding flag pin as **active low** or **falling edge** input

Flag Polarity applies for Flag Pins used as inputs or interrupts

Flag Control Registers (FIO_FLAG_C and FIO_FLAG_S)

Flag Clear Register (**FIO_FLAG_C**) is used to **clear** the Flag Pin.
When it is used as an interrupt pin you have to clear it after an interrupt occured.

Flag: PF15 PF0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIO_FLAG_C

Write 1 to **clear**

Writing a **0** has no effect

Flag Set Register (**FIO_FLAG_S**) is used to **set** the Flag Pin.

Flag: PF15 PF0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIO_FLAG_S

Write 1 to **set**

Read either register for flag state (input or output)

Setting a flag pin that is configured as interrupt, allows software configurable interrupts

Flag Data Register *(FIO_FLAG_D)* *and* Flag Toggle Register *(FIO_FLAG_T)*

The **Flag Data Register** ***(FIO_FLAG_D)***
Directly specifies the state of all PFx pins.

Flag: PF15 PF0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIO_FLAG_D

A "**0**" clears the state of the pin.

A "**1**" sets the state of the pin.

Flag Toggle Register ***(FIO_FLAG_T)***
toggles the state of all PFx pins**.**

Flag: PF15 PF0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIO_FLAG_T

Write a 1 to toggle

Flag Interrupt A and B Mask *Clear* and *Set* Registers (FIO_MASKx_C and FIO_MASKx_S)

FIO_MASKx_C is used to **disable** the Flag as interrupt source.

Flag: PF15 PF0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Write 1 to **clear**

FIO_MASKx_S is used to **enable** the Flag as interrupt source.

Flag: PF15 PF0

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

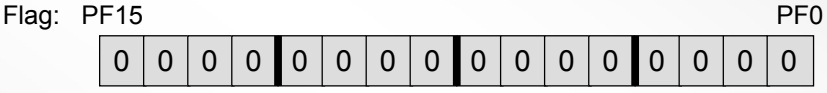
Write 1 to **set**

An interrupt is created with a logical OR of all enabled Flags.

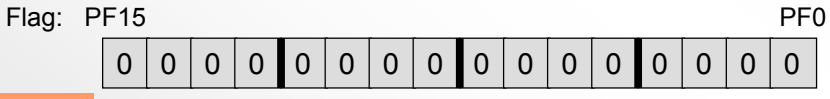
FIO_MASKA_S/ FIO_MASKA_C and **FIO_MASKB_S/ FIO_MASKB_C** registers allow two different Interrupt priorities for all Flag Pins. (FlagA and FlagB interrupt)

Flag Interrupt A and B Mask *Data* and *Toggle* Registers (FIO_MASKx_D and FIO_MASKx_T)

FIO_MASKx_D is used to directly specify the mask value of the Flag(s) as interrupt source(s).



FIO_MASKx_T is used to **toggle** the state of the interrupt mask of the Flag(s) as interrupt source(s).



Write 1 to **toggle**

Clearing FIO Interrupts

- **Edge sensitive interrupts must be cleared within the ISR**
  - To clear an interrupt, write 1 to the corresponding PF bit in the FIO_FLAG_C register **(recommended way)**

    or

    a 0 (zero) to the corresponding PF bit in the FI0_FLAG_D register
    - **NOTE**: FIO_FLAG_D affects all 16 bits, so care must be taken to not inadvertently change the state of other pins.

Example Configurations

/* If using pre-defined registers (such as the FIO_DIR), then the defBF533.h header file must be included. */

```
#include <defBF533.h>
```

/* Setting the Flag Direction Register */

```
P0.H = HI(FIO_DIR);      // assign P0 the address of
P0.L = LO(FIO_DIR);      // FIO_DIR register
R0.L = 0x000f;           // load value of 0xf into R0
W[P0] = R0.L;            // store value
```

/*The result of the last store instruction sets PF0-3 as outputs (writing a 1 to bits 0-3 of the FIO_DIR register). */

Example Configurations

/* If using pre-defined registers (such as the FIO_DIR), then the defBF533.h header file must be included. */

```
#include <defBF533.h>
```

/* Setting the Flag Set Register – Write 1 to Set */

```
P0.L = LO(FIO_FLAG_S);
P0.H = HI(FIO_ FLAG_S);
R0.L = 0x000f;
W[P0] = R0.L;
```

/* This sequence of code sets PF0-3 (configured as outputs on the previous slide). */

/* Setting the Flag Clear Register – Write 1 to Clear */

```
P0.L = LO(FIO_FLAG_C);
P0.H = HI(FIO_ FLAG_C);
R0.L = 0x000f;
W[P0] = R0.L;
```

/* This sequence of code clears PF0-3 (previously set in the code above). */