Experiment 9

Timers and Time Base Generation

Timers/Counters

Almost every microcontroller comes with one or more (sometimes many more) built-in timer/-counters, and these are extremely useful to the embedded programmer. The term timer/counter itself reflects the fact that the underlying counter hardware can usually be configured to count either regular clock pulses (making it a timer) or irregular event pulses (making it a counter). Sometimes timers are also called "hardware timers" to distinguish them from software timers which are bits of software that perform some timing function.

What is a Timer?

We use timers every day - the simplest one can be found on your wrist. A simple clock will time the seconds, minutes and hours elapsed in a given day - or in the case of a twelve hour clock, since the last half-day. ARM timers do a similar job, measuring a given time interval.

Micro-controllers, such as the TM4C123 utilize hardware timers to generate signals of various frequencies, generate pulse-width-modulated (PWM) outputs, measure input pulses, and trigger events at known frequencies or delays. The TM4C123 parts have several different types of timer peripherals which vary in their configurability. The simplest timers are primarily limited to generating signals of a known frequency or pulses of fixed width. While more sophisticated timers add additional hardware to utilize such a generated frequency to independently generate signals with specific pulse widths or measure such signals.

How a Timer Works?

An ARM timer in simplest term is a register. Timers generally have a resolution of 16/32 or 32/64 bits. So a 16 bit timer is 16 bits wide, and is capable of holding value within 0-65536. But this register has a magical property - its value increases/decreases automatically at a predefined rate (supplied by user). This is the timer clock and this operation does not need CPU's intervention.

An example of a basic timer is illustrated in Figure 9.2. This timer has four components – a controller, a prescaler (PSC), an "auto-reload" register (ARR) and a counter (CNT). The function of the prescaler is to divide a reference clock to lower frequency. The TM4C123 timers have 8-bit prescaler registers for 16-bit timers which can divide the reference clock by any value 1..255 and 16-bit prescalar for 32-bit timer which can divide the reference clock by any value

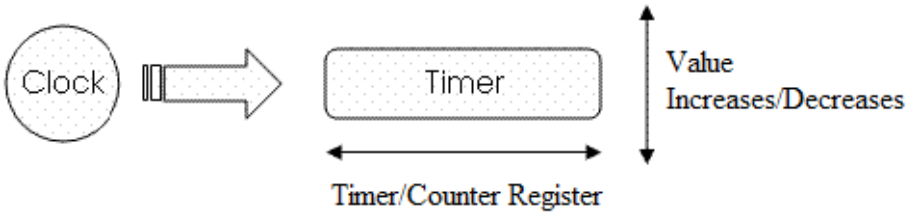

Figure 9.1: Timer Register

1..65535. The counter register can be configured to count up, down, or up/down and to be reloaded from the auto reload register whenever it wraps around (an "update event") or to stop when it wraps around. The basic timer generates an output event (TGRO) which can be configured to occur on an update event or when the counter is enabled (for example on a GPIO input).

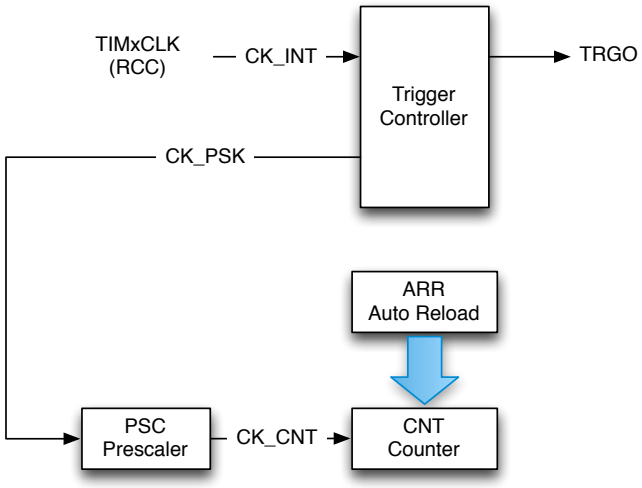

Figure 9.2: Basic Timer

To understand the three counter modes consider Figure 9.3. In these examples, we assume a prescaler of 1 (counter clock is half the internal clock), and a auto reload value of 3. Notice that in "Up" mode, the counter increments from 0 to 3 (ARR) and then is reset to 0. When the reset occurs, an "update event" is generated. This update event may be tied to TRGO, or in more complex timers with capture/compare channels it may have additional effects. Similarly, in "Down" mode, the counter decrements from 3 to 0 and then is reset to 3 (ARR). In Down mode, an update "event" (UEV) is generated when the counter is reset to ARR. Finally, in Up/Down mode, the counter increments to ARR, then decrements to 0, and repeats. A UEV is generated before each reversal with the effect that the period in Up/Down mode is one shorter than in either Up or Down mode.

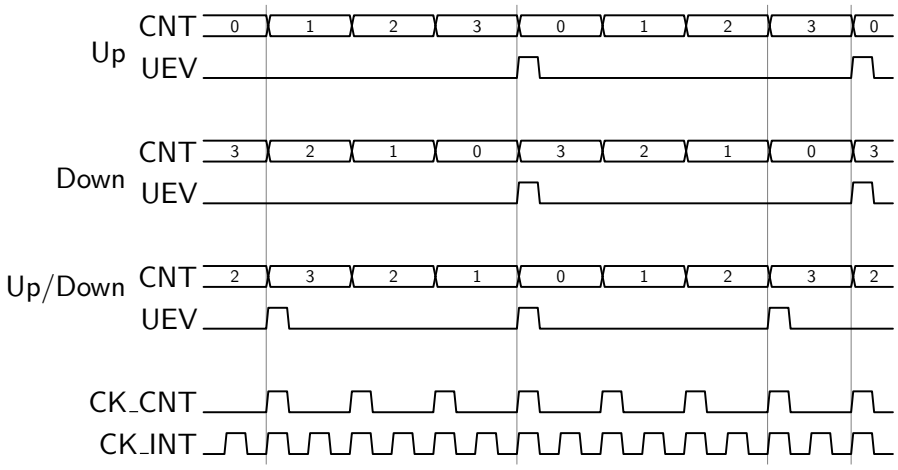

Figure 9.3: Counter Modes (ARR=3, PSC=1)

Clock Configuration

The clock system on the Tiva-C Launchpad is extremely flexible. The clock tree of system clock configuration is shown in Figure 9.4

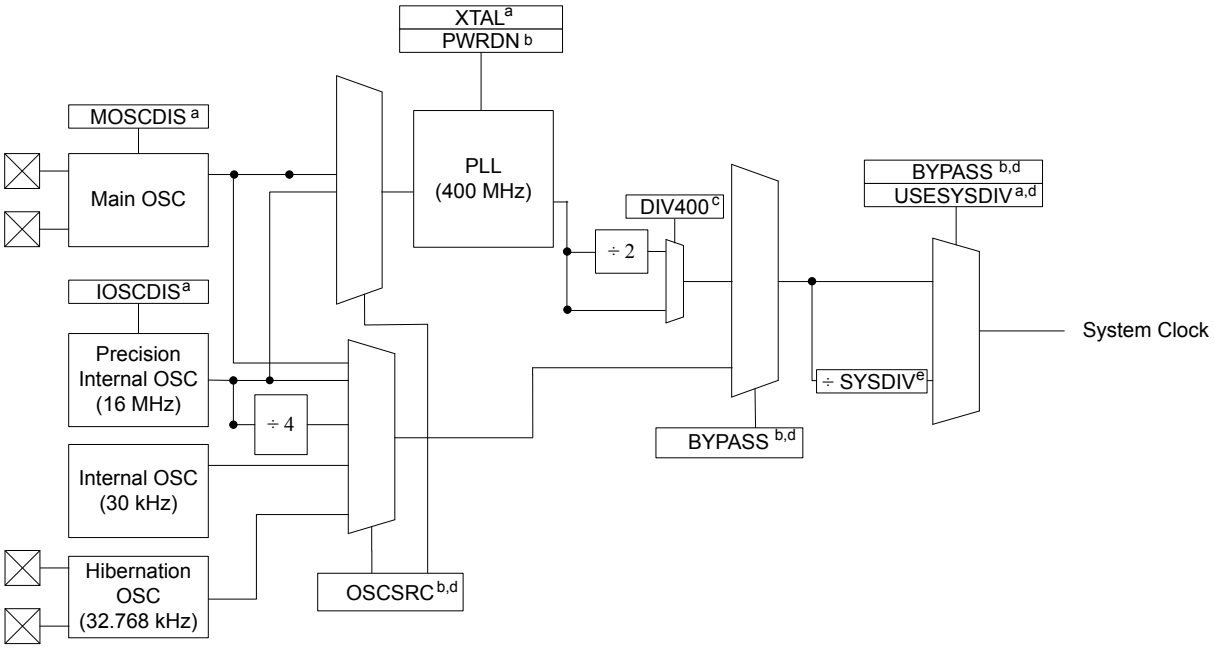

Figure 9.4: System Clock Tree

The internal system clock (SysClk), is derived from any of the four reference sources plus two others: the output of the main internal PLL and the precision internal oscillator divided by four (4 MHz ± 1%). The frequency of the PLL clock reference must be in the range of 5 MHz to 25 MHz (inclusive). Following table shows how the various clock sources can be used in a system.

Clock Source	Drive PLL?		Used as SysClk?	
Precision Internal Oscillator	Yes	BYPASS = 0, OSCSRC = 0x1	Yes	BYPASS = 1, OSCSRC = 0x1
Precision Internal Oscillator divide by 4 (4 MHz ± 1%)	No	-	Yes	BYPASS = 1, OSCSRC = 0x2
Main Oscillator	Yes	BYPASS = 0, OSCSRC = 0x0	Yes	BYPASS = 1, OSCSRC = 0x0
Low-Frequency Internal Oscillator (LFIOSC)	No	-	Yes	BYPASS = 1, OSCSRC = 0x3
Hibernation Module 32.768-kHz Oscillator	No	-	Yes	BYPASS = 1, OSCSRC2 = 0x7

Figure 9.5: Clock Source Options

The Run-Mode Clock Configuration (RCC) and Run-Mode Clock Configuration 2 (RCC2) registers provide control for the system clock. The RCC2 register is provided to extend fields that offer additional encodings over the RCC register. When used, the RCC2 register field values are used by the logic over the corresponding field in the RCC register. In particular, RCC2 provides for a larger assortment of clock configuration options. These registers control the following clock functionality:

- Source of clocks in sleep and deep-sleep modes
- System clock derived from PLL or other clock source
- Enabling/disabling of oscillators and PLL
- Clock divisors
- Crystal input selection

To configure RCC and RCC2 clock configuration registers consult the data sheet of TM4C123.

Timers in TM4C123

The Tiva-C General-Purpose Timer Module (GPTM) in TM4C123 contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks. Each 16/32-bit GPTM block provides two 16-bit timers/counters(referred to as Timer A and Timer B) that can be configured to operate independently as timers or event counters. The register map of the GPTM is given in the data sheet. Some key registers are described below:

SYSCTL_RCGCTIMER_R - enable clock for GPTM (enable clock for Timer0)

TIMER0_CTL_R - control Timer0 module

TIMER0_CFG_R - control global operation of Timer0 module (use 32-bit mode)

TIMER0_TAMR_R - control the mode for Timer0

TIMER0_TAPR_R - set the prescalar for Timer0

TIMER0_TAILR_R - load the starting count value into the timer when counting down and set the upper bound for the timeout event when counting up

How to Configure a Timer for Periodic Interrupts

The GPTM is configured for Periodic mode by the following sequence:

1. Enable the General Purpose Timer Module(GPTM) by asserting the appropriate TIMERn bit in the RCGCTIMER register.
2. Ensure the timer is disabled (the TnEN bit in the GPTMCTL register is cleared) before making any changes.
3. Write the GPTM Configuration Register (GPTMCFG) with a value of 0x04 to configure in 16-bit mode.
4. Write a value of 0x2 in the TnMR field of the GPTM Timer n Mode Register (GPTMTnMR) to configure it in periodic mode
5. Load the period value into the GPTM Timer n Interval Load Register (GPTMTnILR)
6. Load the prescale value into the GPTM Timer n Prescale Register (GPTMTnPR).
7. Assert Timer A Time-Out Clear Interrupt (TATOCINT) bit of GPTM Interrupt Clear Register(GPTMICR) to clear the time-out flag.
8. Set Timer A Time-Out Interrupt Mask (TATOIM) bit of GPTM Interrupt Mask Register(GPTMIMR) to enable the time-out interrupt.
9. Set the priority in the correct NVIC Priority Register.
10. Enable the correct interrupt in the correct NVIC Priority Register
11. Set the TnEN bit in the GPTMCTL register to enable the timer and start counting.

Note1: *The register map for System Control module (for RCGCTIMER) is to be consulted from article* ***5.4 - System Control - Register Map*** *of the controller datasheet.*

Note2: *The register map for Timers is to be consulted from article* ***11.5 - General-Purpose Timers - Register Map*** *of the controller datasheet.*

Note3: *Consult table 2-8 and 2-9 for the entries of the vector table from article* ***2.5.2 The Cortex-M4F Processor - Exception Types*** *of the datasheet.*

Note4: *The register map for NVIC is to be consulted from article* ***3.2 - Cortex-M4 Peripherals - Register Map*** *of the controller datasheet.*

Note5: *The register map for GPIO is to be consulted from article* ***10.4 - General-Purpose Input/Outputs (GPIOs) - Register Map*** *of the controller datasheet.*

Source Code

Complete source code for generating the periodic interrupts for Timer 0A (configured in 16-bits). This program toggles the state of PF1 with 1 Hz frequency.

Example 9.1 (timer.h).

```
#include "TM4C123.h"

void SystemInit (void)
{
  /* ----------------------------FPU settings ----------------------------*/
  #if (__FPU_USED == 1)
  SCB->CPACR |= ((3UL << 10*2) |              /* set CP10 Full Access */
  (3UL << 11*2)  );               /* set CP11 Full Access */
  #endif
}

//constant values
#define   TIM0_CLK_EN           0x01
#define   TIM0_EN               0x01
#define   TIM_16_BIT_EN         0x04
#define   TIM_TAMR_PERIODIC_EN  0x02
#define   TIM_FREQ_10usec       159
#define   TIM0_INT_CLR          0x01
#define   EN0_INT19             0x80000
#define   PORTF_CLK_EN          0x20
#define   TOGGLE_PF1            0x02
#define   LED_RED               0x02

//function headers
void GPIO_Init(void);
void Timer_Init(unsigned long period);
void DisableInterrupts(void);
void EnableInterrupts(void);
void WaitForInterrupt(void);

void Timer_Init(unsigned long period)
{
  //enable clock for Timer0
  SYSCTL->RCGCTIMER  |= TIM0_CLK_EN;

  //disable Timer0 before setup
  TIMER0->CTL  &= ~(TIM0_EN);
  //configure 16-bit timer mode
  TIMER0->CFG  |= TIM_16_BIT_EN;
  //configure periodic mode
  TIMER0->TAMR  |= TIM_TAMR_PERIODIC_EN;
  //set initial load value
  TIMER0->TAILR  = period;
  //set prescalaer for desired frequency 100kHz
  TIMER0->TAPR  = TIM_FREQ_10usec;
```

```
  //clear timeout interrupt
  TIMER0->ICR  = TIM0_INT_CLR;
  //enable interrupt mask for Timer0A
  TIMER0->IMR  |= TIM0_EN;

  DisableInterrupts();
  //Set priority for interrupt
  NVIC_SetPriority(TIMER0A_IRQn,2);
  //enable interrupt 19 (timer0A interrupt)
  NVIC_EnableIRQ(TIMER0A_IRQn);

  //enableTimer0A
  TIMER0->CTL  |= TIM0_EN;

  EnableInterrupts();
}


void GPIO_Init()
{
  //configure red led (PF1)
  SYSCTL->RCGCGPIO  |= PORTF_CLK_EN;
  GPIOF->DEN  |= LED_RED;
  GPIOF->DIR  |= LED_RED;
}


void TIMER0A_Handler(void)
{
  TIMER0->ICR  = TIM0_INT_CLR;
  GPIOF->DATA  ^= TOGGLE_PF1;
}



int main(void)
{
  //generate asquare wave for 2Hz
  Timer_Init(50000);

  //initialize PF1 as digital output
  GPIO_Init();

  while(1)
    WaitForInterrupt();
}
```