Timers and PWM

Gary J. Minden
October 23, 2018
Overview of PWM, Timers, Serial Intf.

• Timers
• Pulse Width Modulator -- PWM
Timers

- Periodic time tick, similar to SysTick
- Count external events
- Measure time of external events
- The TM4C1294 has eight General Purpose Timer Modules (GPTM)
Periodic Time Tick

System Clock
120 MHz

Timer Clock
0.5 mS

Timer Flags/Interrupts

Not to scale
Count Events

External Events

10 Events
-- Measure motion of motor
-- Measure position of sensor
-- Sensor measurement
Measure Time Interval

-- Measure Motor Speed
-- E.g. at 3,000 RPM; shaft rotates 50 times per second; 20 mS Time Interval
Timer Features

- Eight General-Purpose Timer Modules (GPTM), each of which provides two 16-bit timers/counters.
  - As a single 32-bit timer
  - As one 32-bit Real-Time Clock (RTC) to event capture
  - For Pulse Width Modulation (PWM)
  - To trigger analog-to-digital conversions
- 32-bit Timer modes
  - Programmable one-shot timer
  - Programmable periodic timer
  - Real-Time Clock when using an external 32.768-KHz clock as the input
  - User-enabled stalling when the controller asserts CPU Halt flag during debug
  - ADC event trigger
Timer Features

- 16-bit Timer modes
  - General-purpose timer function with an 8-bit pre-scaler (for one-shot and periodic modes only)
  - Programmable one-shot timer
  - Programmable periodic timer
  - User-enabled stalling when the controller asserts CPU Halt flag during debug
  - ADC event trigger
- 16-bit Input Capture modes
  - Input edge count capture
  - Input edge time capture
- 16-bit PWM mode
  - Simple PWM mode with software-programmable output inversion of the PWM signal
Timer Conceptual Diagram

- SysClk
- Scale
- Reload
- Counter
- Reload Value
- 0
- Interrupt
Timer Counter Values

Reload Value

0

Reload Interrupt
Timer Clock Generation

• At nominal 120 MHz
  • 16-bit counter -- 0.548 mS

• Divided System Clock by Pre-Scale Value
  • Pre-Scale Values of 1-256
  • With a Pre-Scale Value of 256, 16-bit counter -- 140.35 mS
Parameter Calculation

System clock is $120 \times 10^6$ (120 MHz), a period of 8.33 ns

$T_T = 8.33 \text{ ns} \times M$, where $M$ is the Reload Value and $0 < M < 2^{16}$ (65,536)

Suppose $T_T = 20$ ms, then $M = \frac{20 \times 10^{-3}}{8.33 \times 10^{-9}} = 2.4 \times 10^6$

$2.4 \times 10^6 > 2^{16}$

We must reduce $M$ to $M < 2^{16}$. We scale the system clock by $K$

$T_T = (8.33 \text{ ns} \times K) \times M$, where $M$ is the Reload Value, $0 < M < 2^{16}$, and $K$ is the scale value, $1 \leq K \leq 256$; many possibilities for $K$ and $M$.

Let’s pick $M = 60,000$; then we have $60,000 = \frac{20 \times 10^{-3}}{(8.33 \times 10^{-9} \times K)}$;

Solving for $K$ we get $K = \frac{20 \times 10^{-3}}{(60,000 \times 8.33 \times 10^{-9})}$; $K = 40.02$, but must be an integer, $K \geq 40$

So, $M = 60,000$ and $K = 40$, $T_T = 19.99$ ms.
GPTM Block Diagram

Figure 9-1. GPTM Module Block Diagram
Programming Example -- FreeRTOS

//
**************************************************************************
// Set up timer for 10 mS interval and handle with interrupts
//
// Author: Gary J. Minden
// Organization: KU/EECS/EECS 388
// Date: 2014–04–21
// Version: 1.0

// Purpose: Set up timer for 10 mS interval and handle
//          with interrupts

// Notes:

//**************************************************************************
//
Programming Example -- FreeRTOS

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl1.h"
#include "inc/hw_types.h"

#include "driverlib/sysctl1.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "Drivers/rit128x96x4.h"
#include "drivers/uartstdio.h"

#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

#include "stdio.h"
Programming Example -- FreeRTOS

//
// Global subroutines and variables
//

extern void Task_TimerInterrupt( void *pvParameters );
extern void Timer_0_A_ISR();

unsigned long int TimeCount = 0;

xSemaphoreHandle Timer_0_A_Semaphore;
void Task_TimerInterrupt( void *pvParameters ) {

    // Constants and Variables
    unsigned long int Hours, Minutes, Seconds, CentiSeconds;
    char DisplayString[24];

    // Initialize Timer_0_A_Semaphore
    vSemaphoreCreateBinary( Timer_0_A_Semaphore );
}
/
// Adapted from TI Stellaris Timer Example
//
SysCtlPeripheralEnable( SYSCTL_PERIPH_TIMER0 );
IntRegister( INTTIMER0A, Timer_0_A_ISR );
TimerConfigure( TIMER0_BASE,
                  TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_PERIODIC );
TimerPrescaleSet( TIMER0_BASE, TIMER_A, 9 ); // Set K (actual – 1)
TimerLoadSet( TIMER0_BASE, TIMER_A, 50000 ); // Set M
// Enable Timer_0_A interrupt
TimerIntEnable( TIMER0_BASE, TIMER_TIMA_TIMEOUT );
// Enable Timer_0_A interrupt in NVIC
IntEnable( INT_TIMER0A );
// Enable (Start) Timer
TimerEnable( TIMER0_BASE, TIMER_A );
//
// Initialize time of day to 00:00:00.00
//
Hours = 0;
Minutes = 0;
Seconds = 0;
CentiSeconds = 0;
Programming Example -- FreeRTOS

```c
while( 1 ) {
    xSemaphoreTake( Timer_0_ASemaphore, portMAX_DELAY );
    CentiSeconds++;
    if ( CentiSeconds >= 100 ) {
        CentiSeconds = 0;
        Seconds++;
        // If Seconds is a multiple of 10, print the TOD
        // at the end of the outer most if-statement
        if ( Seconds % 10 == 0 ) {
            bPrintTimeOfDay = true;
        } else {
            bPrintTimeOfDay = false;
        }
    }
    if ( Seconds >= 60 ) {
        Seconds = 0;
        Minutes++;
        if ( Minutes >= 60 ) {
            Minutes = 0;
            Hours++;
            if ( Hours >= 24 ) {
                Hours = 0;
            }
        }
    }
}
```
if ( bPrintTimeOfDay ) {
   UARTprintf( "Time: %02d:%02d:%02d:%02d\n",
               Hours, Minutes, Seconds, CentiSeconds );
   bPrintTimeOfDay = false;
}
}
Programming Example -- FreeRTOS

//****************************************************************************
// Define an interrupt service routine for Timer_0_A
// We'll try the TI ARM Compiler pragma
//
void Timer_0_A_ISR() {
    portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;

    // Clear interrupt and increment count
    TimerIntClear( TIMER0_BASE, TIMER_TIMA_TIMEOUT );
    TimeCount++;

    // "Give" the Timer_0_A_Semaphore
    xSemaphoreGiveFromISR( Timer_0_A_Semaphore,
                           &xHigherPriorityTaskWoken );
Programming Example -- FreeRTOS

//
// If xHigherPriorityTaskWoken was set to true,
// we should yield. The actual macro used here is
// port specific.
//
portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
Timer Exercise

Compute $K$ and $M$ for $T_T = 30$ ms
Timer Exercise

Compute $K$ and $M$ for $T_T = 17$ ms
Timer Exercise

Compute K and M for $T_T = 45$ ms
Timer Exercise

Compute $K$ and $M$ for $T_T = 3$ ms
Timer Exercise

Compute $K$ and $M$ for $T_T = 23$ ms
Timer Exercise

Compute $K$ and $M$ for $T_T = 77$ ms
Pulse Width Modulator
PWM

- Generate a repeating pulse with variable width

- Useful for controlling various devices, including motors, servos, and electronics

- Use a low-pass filter to get DC output. DC voltage varies as pulse width. Longer pulse, higher voltage
PWM Features

- Four PWM generator blocks, each with one 16-bit counter, two PWM comparators, a PWM signal generator, a dead-band generator, and an interrupt/ADC-trigger selector
- One fault input in hardware to promote low-latency shutdown
- One 16-bit counter
- Two PWM comparators
- PWM generator
- Dead-band generator
- Optional output inversion of each PWM signal (polarity control)
- Optional fault handling for each PWM signal
- Can initiate an ADC sample sequence
PWM Block Diagram

Figure 23-1. PWM Module Diagram
PWM General Operation

Figure 16-3. PWM Count-Down Mode

- Load
- CompA
- CompB
- Zero
- Load
- Zero
- A
- B
- Dir

ADown
BDown

© G. J. Minden 2013
PWM Up/Down Mode

Figure 16-4. PWM Count-Up/Down Mode
PWM Example

Figure 16-5. PWM Generation Example In Count-Up/Down Mode

Load
CompA
CompB
Zero
PWMA
PWMB
PWM Example Program

//*****************************************************************************/
//—Task_Motor_PWM_0_4
//
// Author: Gary J. Minden
// Organization: KU/EECS/EECS 388
// Date: 2017–11–03 (B71103)
// Platform: TI Tiva TM4C1294 evaluation board
// Version: 1.0
//
// Purpose: This task implements several PWM control motor tests. The tests include:
// (1) Alternating CW, pause, CCW motion
//
// APIs are defined for:
// (1) Enable/Disable the motor for all tasks.
// (2) Set the motor speed and direction based on percentage of PWM period.
// (3) Set the motor speed and direction based on RPM and direction.
//
// Set up PWM 0 output 4 (PWM_0_4) for 20 mS period to control a servo. We will use PWM_0 output
// 4 connected through PortG<0>. We will not use interrupts.
//
//*****************************************************************************/
PWM Example Program

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"

#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>

#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "Drivers/uartstdio.h"

#include "Drivers/Processor_Inititalization.h"
#include "Tasks/Task_ReportData.h"

#include "stdio.h"

#include "FreeRTOS.h"
#include "task.h"
PWM Example Program

///
/// Defines
///
/// Define a scale for specifying the pulse width.
/// The scale is 1000 / 20 ms. A value of 100
/// represents ( 75 / 1000 ) ms = 1.5 ms. This
/// is the neutral position.
///
#define PWM_Period 1000
#define PWM_State_Low 50
#define PWM_State_High 100
#define PWM_State_Neutral 75
PWM Example Program

///
///   PWM Parameters
///
/// Example to be presented in lecture
///
#define PWM_0_4_Period XXX
#define PWM_0_4_Low_Count ((PWM_0_4_Period * PWM_State_Low) / PWM_Period)
#define PWM_0_4_High_Count ((PWM_0_4_Period * PWM_State_High) / PWM_Period)
#define PWM_0_4_Neutral_Count ((PWM_0_4_Period * PWM_State_Neutral) / PWM_Period)
PWM Example Program

```c
// Define PWM duty cycle states. We alternate between a 1.0 mS
// pulse and a 2.0 mS pulse. The full period (1000) is 20.0 mS.
// 1.0 mS is 1/20 (50/1000) and 2.0 mS is 2/20 (100/1000).
//
typedef enum { PWM_CCW_100, PWM_Stop, PWM_CW_100 } PWM_States;
PWM_States PWM_CurrentState = PWM_Stop;
```
PWM Example Program

//
// Task initialization
//
extern void Task_Motor_PWM_0_4( void *pvParameters ) {

//*****************************************************************************/
//
// Constants and Variables
//
// Time interval between changes in duty cycle, 0.5 Seconds
//
uint32_t DutyCycleDelta = pdMS_TO_TICKS( 1000 );

//
// Configure PWM_0_0 for XX mS period.
//
SysCtlPeripheralEnable( SYSCTL_PERIPH_GPIOG );
SysCtlPeripheralEnable( SYSCTL_PERIPH_PWM0 );
 PWM Example Program

    //
    // Configure the GPIO pin muxing to select PWM_0_4 functions for these pins.
    // This step selects which alternate function is available for these pins.
    // This is necessary if your part supports GPIO pin function muxing.
    // Consult the data sheet to see which functions are allocated per pin.
    // Set GPIO PortG<0> as PWM pins to output the PWM_0_4 signal.
    // PWM signals were taken from Table 10–2 of the TM4C1294 datasheet.
    //
    GPIOPinConfigure( GPIO_PG0_M0PWM4 );
    GPIOPinTypePWM( GPIO_PORTG_BASE, GPIO_PIN_0 );

    //
    // Configure PWM0 to count down without synchronization.
    //
    PWMClockSet( PWM0_BASE, PWM_SYSCLK_DIV_64 );
    PWMPGenConfigure( PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN |
                      PWM_GEN_MODE_NO_SYNC );
    PWMPGenPeriodSet( PWM0_BASE, PWM_GEN_2, PWM_0_4_Period );

    //
    // Set PWM_0_4 to a duty cycle of PWM_0_4_Neutral_Count
    //
    PWMPulseWidthSet( PWM0_BASE, PWM_OUT_4, PWM_0_4_Neutral_Count );
PWM Example Program

//
// Enable the PWM_0_4 output signals.
//
PWMOutputState( PWHO_BASE, PWM_OUT_4_BIT, true );

//
// Enable the PWM generators.
//
PWMGenEnable( PWHO_BASE, PWM_GEN_2 )
PWM Example Program

    //
    // Main task.
    //
    while ( 1 ) {

        //
        // *******Your code here********
        //

        //
        // Task delay.
        //
        vTaskDelay( DutyCycleDelta );

    }

}
PWM Parameters

- We need two time durations: the period and the pulse width
- The system clock frequency is 120 MHz which has a period of 8.33 ns
- The basic time duration function is:

  \[ T_p = (8.33 \text{ ns} \times K) \times M \]

  where \( T_p \) is the time duration, e.g. period, \( K \) is the scaling factor, and \( M \) is the re-load value.

  - \( K \) and \( M \) must be integers
  - \( K = 1, 2, 4, \ldots, 64 \) and \( M < 2^{16} \)
  - Determining \( K \) and \( M \) is an iterative process
PWM Parameters

• Suppose we want $T_p = 33$ ms

$$33 \text{ ms} = (8.33 \text{ ns} \times K) \times M$$

Pick $M = 50,000$, then

$$K = \frac{33 \text{ ms}}{(8.33 \text{ ns} \times 50,000)} = 79.23$$

Maximum $K$ is 64, so set $K = 64$, then

$$M = \frac{33 \text{ ms}}{(8.33 \text{ ns} \times 64)} = 61,900$$

$M$ must be an integer less than 65,536
PWM Parameters

- K is common for period and pulse width
- Pulse duration is less than period, do \( M_w \) is less than \( M_p \)
- Compute \( M_p \) as above