TIMERS USING SONIC™ AND COUNT REGISTER IN ORION™

APPLICATION NOTE AN-127

By Sugan Subramanian

INTRODUCTION

This is an Application Note on two timer modules, one based on SONIC™ running at 20MHz and another based on COUNT register present in R4600™ running at 50MHz. The timer-modules are broken up into two groups of functions. The first group of functions is specific to a particular timer. These functions are always written in assembly language and do low level timer specific initialization for starting and stopping the timer. The second set of functions have generic functionalities. These functions are written in C and they are used to keep track of the number of ticks in the timer in microseonds between the period at which the timer is started and stopped, to display time, and to set a constant value based on the current speed at which the processor is set to run. First, we are going to discuss how to measure time using SONIC. Secondly, we going to look at how to keep track of time using R4600's count register. Finally, we will discuss how to use the timers.

SONIC TIMER

SONIC is used to measure time in various boards, including IDT's 79S460-board (an R4x00 evaluation board) and 79S381™-board (a R30xx evaluation board). The SONIC has a 32-bit downcounter that is controlled by SONIC's 16-bit registers, watchdog register1 and watchdog register2. In the SONIC running at 20MHz, each timer-tick represents 200-ns. In general,

t = (1000/f)*4

where t is the time for each timer-cycle in ns and f is the frequency of SONIC in MHz. Moreover, (1000/f) represents time period per cycle. Sonic decrements the counter registers once every four cycles. Mechanisms involved in starting the timer, stopping the timer, and displaying time using SONIC are discussed in the following sections. The following figure describes the header file "sonic_globals.h" used by SONIC-timer's high level functions.

```
/* sonic_globals.h */

/* speed based on user specified
frequency of SONIC */
unsigned int current_speed;
/* speed based on default frequency of
SONIC */
unsigned int def_speed;
/* counter keeps track of the time count
*/
unsigned int counter;
```

Figure 1
sonic_globals.h

Starting SONIC Timer

Starting the timer involves enabling the ST bit in SONIC's control register, and initializing 16-bit watchdog registers 1 and 2 with all their bits set to 1 (0xffff). "TimerStart", a low level function, does the initialization required for starting a timer. "timer_start", a high level function, calls "TimerStart" and keeps track of the number of timer ticks. Figures 2 and 3 describe "TimerStart" and "timer_start" routines for SONIC.

```
/*for 79s460-board */
#include <r4ksonic.h>
/*for 79s381-board
#include <r3ksonic.h>
But, don't include both headers */
#    LOW LEVEL FUNCTION TimerStart

     .globl  TimerStart
     .ent    TimerStart
     .set    noreorder
TimerStart:
     li      v0, SONIC_COMMAND_REG_ST_BIT
     li      t1, SONIC_COMMAND_REG
     nop
     nop
     sw      v0,0(t1)
     nop
     nop
     li      v0, oxffffffff
     sw      v0, SONIC_WATCHDOG1(t1)
     nop
     nop
     sw      v0, SONIC_WATCHDOG2(t1)
     nop
     nop
     j       ra
     nop
     nop
     .end    TimerStart
```

Figure 2
SONIC TimerStart routine

```
     #include <sonic_globals.h>
     /* HIGH LEVEL FUNCTION timer_start */
     unsigned int timer_start()
     {
     if (cur_speed)
/* set cur_speed of the SONIC based on the user
specified frequency */
             def_speed = cur_speed;

     counter = TimerStart();
     return counter;
     }
```

Figure 3
SONIC timer_start routine

Stopping SONIC Timer

The SONIC-timer is stopped by enabling the STP bit in SONIC's control register. By enabling STP bit, the SONIC preserves the previous values of WATCHDOG registers 1 and 2. "TimerStop" returns a 32-bit value that is a concatenation of 16-bit count in watchdog registers 1 and 2. "timer_stop" function calls the low level "TimerStop" to retrieve the current timer value. "timer_stop" returns an unsigned integer value representing the time period between the previous timer initiation and the current instance of execution (in microseconds). Figures 4 and 5 describe "TimerStop" and "timer_stop" routines for SONIC.

```
/*for 79s460-board */
#include <r4ksonic.h>
/*for 79s381-board
#include <r3ksonic.h>
But, don't include both headers */
     .globl  TimerStop
     .ent    TimerStop
     .set    noreorder
TimerStop:
     li      v0, SONIC_COMMAND_REG_STP_BIT
     li      t1, SONIC_COMMAND_REG
     nop
     nop
     sw      v0,0(t1)
     nop
     nop
     lw      v1, SONIC_WATCHDOG1(t1)
     nop
     nop
     lw      v0, SONIC_WATCHDOG2(t1)
     sll     v1,16
     addu    v1,v1,v0
     addu    v0,v1,v1
     j       ra
     nop
     .end    TimerStop
```

Figure 4
SONIC TimerStop routine

```
/* HIGH LEVEL FUNCTION timer_stop */
extern unsigned int counter;
unsigned int timer_stop()
{
     counter -= TimerStop();
/*
     when counter == i,
     i*100 == time in nanosecs
     (i/1000)*100 == time in microsecs
*/
     return counter/10;
}
```

Figure 5
SONIC TimerStop routine

Displaying Time (SONIC)

"disp_time" function displays the time. It is always called after a call to "timer_stop". It takes a parameter which is usually zero, otherwise represents current time-count in microseconds. It displays the timer period in the following format:

" %dS %dmS %duS" where %d represents the number of seconds in the time count; mS - represents the number of milliseconds in the time count; and uS - represents the number of microseconds in the time count. Only the non-zero units are displayed. Figure 6 describes the fuction "disp_time" for SONIC.

```
extern unsigned int counter, def_speed,
     cur_speed;

void disp_time(unsigned int i)
{
     unsigned int temp_counter=counter;

     if (i)
            temp_counter = i;

     printf("elasped time = ");
     if (temp_counter >
            1000000)
     {
            printf("%dS ",
            temp_counter
            /1000000);
            temp_counter %=1000000;
     }

     if (temp_counter > 1000)
     {
            printf("%dmS",
            temp_counter
            /(1000));
            temp_counter %=1000;
     }

     if (temp_counter)
     {
            printf("%duS ",
            temp_counter);
     }

     printf("\n");
}
```

Figure 6
SONIC disp_time routine

Setting Up SONIC timer speed

"set_timer_speed" takes an integer that represents the clock frequency of SONIC in MHz as its parameter and sets a constant that represents the speed of SONIC in ns for the given frequency. This constant is used by the "disp_time" to display the time correctly. The following figure has the C-source for "set_timer_speed".

```
extern unsigned int cur_speed;
void set_timer_speed(int speed)
{
     cur_speed = 1000/speed*2;
}
```

Figure 7
SONIC set_timer_speed routine

Difference between SONIC timer on 79s460-board and 79s381-board

It's recommended to include the header file, "timer_sonic.h" before encorporating the timer routines in his/her code. Only difference between the SONIC timer routine for IDT's 79s460-board (R4xxx evaluation board) and 79s381-board (R30xx evaluation board) is the SONIC chip's base address. The header file "r4ksonic.h" has R4xxx board specific SONIC base address and "r3ksonic.h" has R30xx board specific SONIC base address. The following figures describe these header files.

```
/* include file r4ksonic.h */

#define      SONIC_BASE
     0xbf600000
#define      SONIC_COMMAND_REG
     0xbf600000
#define      SONIC_WATCHDOG1
     0xa4

#define      SONIC_WATCHDOG2
     0xa8
#define      SONIC_COMMAND_REG_ST_BIT
     0x20
#define      SONIC_COMMAND_REG_STP_BIT
     0x10
```

Figure 8
r4ksonic.h

```
/* include file r3ksonic.h */

#define      SONIC_BASE
     0xbfb00000
#define      SONIC_COMMAND_REG

     0xbfb00000
#define      SONIC_WATCHDOG1
     0xa4
#define      SONIC_WATCHDOG2
     0xa8
#define      SONIC_COMMAND_REG_ST_BIT
     0x20
#define      SONIC_COMMAND_REG_STP_BIT
     0x10
```

Figure 9
r3ksonic.h

Constraints on SONIC timer

The SONIC timer module on a SONIC running at 20mHz is capable of counting upto 7 minutes. If the SONIC is running at a different frequency, function "set_timer_speed" should be called before calling "timer_start".

R4600 TIMER

R4600 microprocessor has a COUNT register in CoProcessor 0 (CP0). This COUNT register in CP0 increments its count by one on every timer tick of the R4600 processor. In the R4600 processor running at 50MHz, each timer-tick represents 20-ns. In general,

t = (1000/f)

where t is the time for each timer-tick in nano-seconds, f is the frequency of R4600 processor in MHz, and (1000/f) represents the time per cycle. The following figure describes the header file used by the high level functions of R4600 timer.

```
/* orion_globals.h */

/* speed based on user specified
frequency of R4600 processor */
unsigned int current_speed;
/* speed based on default frequency of
R4600 processor */
unsigned int def_speed;
/* counter keeps track of the time count
*/
unsigned int counter;
```

Figure 10
orion_globals.h

Starting R4600 Timer

Timer is started by resetting the value of the COUNT register in R4600 microprocessor to zero. The following piece of assembly code and c-code shows how to start the timer.

```
#include "r4kcp0.h"
# Timer is started by assigning zero to
# COUNT register located in CP0

     .globl  TimerStart
     .ent    TimerStart
     .set    noreorder
TimerStart:
     and     v0, $0
     mtc0    v0, CP0_COUNT
     nop
     nop
     j       ra
     nop
     nop
     .end    TimerStart
```

Figure 10
R4600 TimerStart routine

```
/* r4kcp0.h */
#define CP0_CONTEXT $4
#define CP0_BVADDR  $8
#define CP0_COUNT          $9
#define CP0_COMPARE $11

/* r4kcp0.h ---contd. */
#define      v0             $2
#define      v1             $3
#define      ra             $31
Figure 11
R4600 header file r4kcp0.h

#include <orion_globals.h>

unsigned int timer_start()
{
     if (cur_speed)
          def_speed = cur_speed;
```

```
        counter = TimerStart();
        return counter;
    }
```

Figure 12
R4600 timer_start routine

Stopping R4600 Timer

Stopping the timer involves simply getting the contents of the COUNT register that represents the most recent timer tick count and converting that timer tick count to microseconds. The following piece of assembly code and c-code shows how to stop the timer.

```
#include "r4kcp0.h"
     .globl  TimerStop
     .ent    TimerStop
TimerStop:
     mfc0    v0, CP0_COUNT
# The timer is stopped by getting
# the most recent value of COUNT
# register
     nop
     nop
     j       ra
     nop
     nop
     .set    reorder
     .end    TimerStop
```

Figure 13
R4600 TimerStop routine

```
extern unsigned int counter;

unsigned int timer_stop()
{
     counter = TimerStop() - counter;
/*
     when counter == i,
     i*20 == time in nanosecs
     (i/1000)*20 == time in microsecs
*/
     return counter/50;
}
```

Figure 14
R4600 timer_stop routine

Displaying Time (R4600)

"disp_time" function is very similar to the one presented previously for the SONIC timer module. Only difference in this case is that the time displayed is based on the frequency of the R4600 in the 79s460-board (50-MHz).

Constraints on R4600 timer

The timer module is capable of counting upto 85 seconds assuming that the R4600 processor is set to run at 50MHz. If the R4600 processor is set to run at a different frequency, function "set_timer_speed" should be called before calling "timer_start" so that "disp_time" displays the correct time. The following piece of c-code describes the "set_timer_speed" function.

```
extern unsigned int cur_speed;
void set_timer_speed(int speed)
{
     cur_speed = 1000/speed;
}
```

Figure 15
R4600 set_timer_speed routine

TIMER MODULE USAGE

The procedures to use SONIC timer for 79s460-board, SONIC timer for 79s381-board, and R4600-timer for 79s460-board are the same. The following figure describes it. Moreover, IDT-C 5.0 is shipped with SONIC-timer/79s460-board and R4600-timer/79s460-board. In IDT-C 5.0, source code for SONIC-timer/79s460-board is located under "/IDTC/timers/SONIC-timer"; and source code for R4600-timer/79s460-board is located under "/IDTC/timers/ORION-timer". Figures 16 and 17 give the general procedure to use the timer routines.

```
#include <timer.h>
main()
{
timer_start();
.
/* main body */
.
timer_stop();
disp_time(0);
}
```

Figure 16
How the timer routine is to be used

```
/* timer.h */
unsigned int timer_start();
unsigned int timer_stop();
void disp_time(unsigned int);
```

Figure 17
timer.h

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.