Using the AT91 Timer/Counter

Ron Goldman

> *The Sun SPOT processor board includes two AT91 Timer Counters that can be used in a variety of ways. This application note describes both how to use the AT91 Timer Counter to measure elapsed time and to perform a periodic task.*

Each Sun SPOT processor board has two AT91 Timer Counters that are part of the ARM920T system-on-a-chip. Each of the AT91 Timer Counters includes three identical 16-bit Timer Counter channels. Of these six Timer Counter channels, four are available for SPOT applications, while two are reserved for system use.

The Timer Counter can operate in two distinct modes: Capture & Waveform generation. Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation.

Each channel has three external clock inputs, five internal clock inputs and two multi-purpose input/output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts.

Several signals are provided on the SPOT processor board via the top connector, though the initial eDemo sensor board does not make them available. The available signals are the three external clock inputs (TCLK0, TCLK1, TCLK2) and the general purpose input/output pins for channel zero (TIOA0, TIOB0).

For a full description of the AT91 Timer Counter please refer to the Atmel documentation for the AT91RM9200 (available at http://www.atmel.com/dyn/resources/prod_documents/doc1768.pdf). In this note we will discuss a few basics and give an example using the Timer to generate a periodic interrupt. For a full list of Timer Counter bit definitions please see the Interface defined in the file com.sun.spot.peripheral.TimerCounterBits.java.

Functional Description

Each channel is organized around a 16-bit counter. The value of the counter is incremented at each positive edge of the selected clock. When the counter has reached the value 0xFFFF and passes to 0x0000, an overflow occurs and the TC_COVFS bit in the Status Register is set. Each channel also has three registers (RA, RB and RC) that can be used in various ways.

There are several types of triggers that can be specified to reset the counter and start the counter clock. These include a software trigger that can be explicitly called in the SPOT application and a Compare RC Trigger that causes a reset when the counter value matches the RC value.

The rate at which the Timer counts is determined by which clock is used. There are five internal clock inputs. In the AT91RM9200, these clock inputs are connected to the Master Clock (MCK), to the Slow Clock (SLCK) and to divisions of the Master Clock. For the Sun SPOTs the Master Clock speed is 59,904 Khz and the Slow Clock is 32.768 Khz. The available clock speeds are shown in Table 1.

TC Clock Input	Clock	Clock Speed (Khz)	Time for One Tick (usec)	Maximum Duration (msec)
TC_CLKS_MCK2	MCK / 2	29,952	0.0334	2.188
TC_CLKS_MCK8	MCK / 8	7,488	0.1335	8.752
TC_CLKS_MCK32	MCK / 32	1,872	0.5342	35.009
TC_CLKS_MCK128	MCK / 128	468	2.1368	140.034
TC_CLKS_SLCK	SLCK	32.768	30.5176	2,000.0

Table 1. Available Clock Speeds

Example 1: Measuring a Time Interval

To use the Timer to measure a time interval use Capture Mode, enable the clock to start it counting, and at the end of the interval just read the counter value:

```
import com.sun.spot.peripheral.*;
import com.sun.spot.util.*;

public class TimerCounterSample implements TimerCounterBits {

    public double measureInterval() {
        IAT91_TC timer = Spot.getInstance().getAT91_TC(0);   // Get a Timer Counter
        timer.configure(TC_CAPT | TC_CLKS_MCK32);            // Use fast clock speed
        timer.enableAndReset();                              // Start counting

        ... interval to measure ...

        int cntr = timer.counter();          // Get number of elapsed clock ticks
        timer.disable();                     // Turn off the counter
        double interval = cntr * 0.5342;     // Convert to time in microseconds
        return interval;
    }
}
```

Example 2: Perform a Periodic Task

To perform a periodic task every 25 milliseconds we want to create a loop that blocks, waiting for the AT91 Timer Counter to generate an interrupt. To do this modify the above code to set the RC Register to the number of counts that will span the desired period, and enable interrupts on RC Compare:

```
    public void periodicTask() {
        IAT91_TC timer = Spot.getInstance().getAT91_TC(0);        // Get a Timer Counter
        int cnt = (int)(25000 / 0.5342);          // number of clock counts for 25 msecs
        timer.configure(TC_CAPT | TC_CPCTRG | TC_CLKS_MCK32);   // enable RC compare
        timer.setRegC(cnt);
        timer.enableAndReset();

        while (true) {                            // Start periodic task loop
            timer.enableIrq(TC_CPCS);             // Enable RC Compare interrupt
            timer.waitForIrq();                   // Wait for interrupt
            timer.status();                       // Clear interrupt pending flag
            doTask();                             // Method will be called every 25 msecs
        }
    }
```

Note: While the above code will generally call `doTask()` every 25 milliseconds, sometimes the call will be delayed because the interrupt occurs during a GC or a long-lived native operation or while a higher priority thread is running. While testing the above code the delay was observed to be either just a few (1-3) milliseconds for incremental GC and about 20-30 milliseconds for a full GC.

Note: The Green release did not include a definition for `TC_CPCTRG`. It should be defined as:

```
int TC_CPCTRG = 0x4000;
```

A Java thread detects an interrupt by calling `waitForIrq()`. This method performs a Channel IO request. If the Timer Counter interrupt bit is set then the request returns immediately. If not, the calling thread is blocked until the interrupt occurs. To clear the interrupt bit call `status()`.

To handle an interrupt in Java you must:

1. Call `configure(int)` to configure the timer so that it will generate an interrupt request.
2. Call `enableAndReset()` to start the timer counting
3. Call `enableIrq(int)` to enable one or more of the interrupt sources associated with this TC channel.
4. Call `waitForIrq()` to wait for the interrupt.
5. Call `status()` to clear the interrupt.
6. Call `enableIrq(int)` to allow another interrupt.
7. Repeat from (4)

If you don't want more interrupts then don't call `enableIrq(int)`.

About Sun Labs

Established in 1990, Sun Microsystems Laboratories is the applied research and advanced development arm of Sun Microsystems, Inc., with locations in California and Massachusetts. Sun Labs is one of the ways Sun invests in the future, and is responsible for many of the technology advancements that have made Sun a technology powerhouse—including asynchronous and high-speed circuits, optical interconnects, 3rd-generation Web technologies, sensors, network scaling and Java technologies. Although many companies have R&D groups, Sun Labs can claim one of the highest rates of technology transfer in the industry.