Core Timer Service Overview

The core timer is a facility built into the MIPS M4K processor core in the PIC32 microcontroller. It is made up of a 32-bit counter register and a 32-bit compare register. The counter register increments at 1/2 the processor clock frequency (SYSCLK). The default SYSCLK frequency is 80Mhz, so the core timer counter increments at 40Mhz. The compare register can be used to trigger an interrupt (core timer interrupt) when the counter matches the value loaded into the compare register.

In the chipKIT MPIDE system, the core timer is used to manage the timing of events at 25ns (nanosecond) resolution. The core timer service facility allow service functions to be registered that will be called by the core timer interrupt service routine (ISR). A core timer service function indicates to the core timer ISR the system time (i.e. core timer counter value) at which it should be called next. This allows a service function to schedule itself to be executed at any time up to 90 seconds in the future with 25ns resolution. This allows event timing to be done with great precision.

In the chipKIT MPIDE system, the core timer service facility is used to implement the low level timing function `millis()`. When the system starts up, it registers a core timer service function that schedules itself to be called once each millisecond. This service function then maintains the system millisecond tick counter that is returned by the `millis()` function.

A core timer service function is a callback routine that is registered with the CoreTimerHandler Interrupt Service Routine (ISR). This functon will be called when the core timer counter has reached the core timer service's trigger time. Each time a core timer service function is called, the current core timer counter value is passed in as a parameter. The core timer service function returns the counter value of the next time it wishes to be called, i.e. the next trigger time.

The core timer service function is guaranteed to be called no earlier than it's next scheduled time, but may be called late. This can occur if interrupts have been disabled, as the core timer ISR will not execute again until interrupts are re-enabled. This will also occur when writing to the flash memory in the PIC32 microcontoller. When a flash memory page write is performed, the processor stops executing instruction for 20ms, although the core timer counter continues to increment. In general, the current time passed in will typically be a few ticks after the requested trigger time. This is usually not a problem as the tick period is 25 nsec. However, if interrupts have been disabled, the service function may be called as much 20-50ms late. It is up to the service function to handle being called late.

The core system time (i.e. core system counter value) is a 32 bit unsigned integer and wraps once every 2^32 /

40,000,000 or ~107.3741824 seconds. When the service functon is called, the current time as represented by this 32 bit unsigned integer is passed in as a parameter. Up to 90 seconds may be added to the current time to specify the next trigger time. Do not worry about the 32 bit unsigned integer wrapping in value as the core timer service assumes the next 90 seconds of time is in the future. Do not exceed 90 seconds as the CoreTimerHandler potentially regards anything beyond that is a time before the current time. It is a requirement that a core timer service function return a next trigger time 0-90 seconds in the future. It may not return something that the CoreTimerHandler ISR may regard to be in the past, that is, do not subtract from the current time, always add to it.

When a core timer service function is registered, the first call to the function will occur on the next regularly scheduled call to the CoreTimerHandler ISR. The system always has a millisecond CoreTimer Service registered and this will typically ensure that a newly registered Service will be called within 1 ms of being registered. However, should interrupts been disabled, this may be late up to 20-50ms.

Currently, a maximum of 3 core timer services functions can be registered simultaneously. One is always taken by the `millisecondCoreTimerService`. Therefore there are two available slots open for use. To register a core timer service function, call `attachCoreTimerService()` passing a pointer to the service function. The `attachCoreTimerService()` function will return false (0), if there are no open slots available. You may de-register (remove) a core timer service function by calling `detachCoreTimerService()` passing a pointer to the function to remove. Once removed the slot becomes available for another core timer service function to be registered. NEVER remove the system `millisecondCoreTimerService` function.

The rules for a core timer service function:

1. Do NOT set the core timer "compare" register directly! If you don't know what this is, GOOD, don't fool with it.
2. Do not do anything that could cause the CoreTimerHandler ISR to be called recursively. Primarily, this means do not enable interrupts as the core timer interrupt flag is still set and will immediately cause the system to call CoreTimerHandler ISR recursively.
3. The current time is passed to the service function as a parameter. This is usually several ticks after the requested trigger time, but if interrupts were disabled this may be as much as 20-50ms late.
4. The current system time is obtained by reading the core timer counter register when the CoreTimerHandler ISR is entered. This is the value passed as a parameter to the service function. It may actually be several ticks old. For this reason is okay to read the core timer counter register directly. Typically this is not necessary as the current time is only out-of-date by a few nsec, which is just the delay in the instructions executed to call the callback Service.
5. A Service will never be called before the requested trigger time, but it may be called late.
6. Do not return a next trigger time more than 90 seconds in the future. Each tick is 25 nsec, there are 40,000,000 ticks in a second; therefore do not add more than 90*40,000,000 to the current time for the next trigger time. Do not attempt to return a negative time, always add time to the current time, do not subtract. It is okay for the 32 bit unsigned integer value returned to wrap when added to the current system time to determine the next trigger time.
7. If the service function returns a next trigger time that is very near to the current time, it is possible for that system time to have already passed. Under this condition the CoreTimerHandler will immediately call the service function again with an updated current time without exiting the CoreTimerHandler ISR.
8. Because the CoreTimerHandler ISR may immediately call a service function without exiting the ISR (as in the previous rule), it is imperative that the service function execute in a timely manner. If the service function takes too long to execute, it is possible to create a condition where the CoreTimerHandler ISR is never exited. In this case, no processor time will be given to execute anything else, include the primary

sketch. Remember, the service function is executing within the context of an interrupt, so the function's code should be written to complete very quickly.

9. Once a Service is registered, it is typically called for the first time within 1 ms of registration unless interrupts were disabled, and then it could be as much as 20-50ms late.
10. Do NOT remove the pre-registered millisecondCoreTimerService CoreTimer Service, this will break the system!

Examples:

Example 1

The simplest of core time service examples: simply schedule a callback at a particular frequency (in this case, 10KHz) and toggle an output pin in the callback. This callback produces a 500nS pulse on pin 4 every 100uS.

```
/* CoreTimer demo1 : demonstrates a simple callback scheduled at
a single frequency (10Khz). This example code is in the public domain. */

void setup() {
  pinMode(4, OUTPUT); // Use IO pin 4 to show operation of callback
  attachCoreTimerService(MyCallback);
}

// We don't need to do anything in the main loop
void loop() {
}

// For the core timer callback, just toggle the output high and low
// and schedule us for another 100uS in the future. CORE_TICK_RATE
// is the number of core timer counts in 1 millisecond. So if we
// want this callback to be called every 100uS, we just divide
// the CORE_TICK_RATE by 10, and add it to the current time.
// currentTime is the core timer clock time at the moment we get
// called.
uint32_t MyCallback(uint32_t currentTime) {
  digitalWrite(4, HIGH);
  digitalWrite(4, LOW);
  return (currentTime + CORE_TICK_RATE/10);
}
```

Share and Enjoy

Posted by mchpacademic