Exercise in Embedded Computing: Timing Services and Introduction to PWM

Sivan Toledo, Tel-Aviv University

October 22, 2009

The purpose of this exercise is develop two general-purpose timing abstractions and to use them in experiments. The experiments focus on pulse-width modulation (PWM), which is a technique that we will use here to control the intensity of LEDs. Many microcontrollers contain hardware PWM modules, but here we will implement PWM in software.

1. Implement a periodic software-invocation interface. The function `systickInit()` should configure a timer/counter to invoke an ISR every millisecond. The ISR should call `systick_periodic_task()`. Implement the module in a file `systick.c`. The file will be included into a program using a `#include` preprocessor directive. The main program will define `systick_periodic_task()`. For example, the following code keeps track of time and toggles an LED about 4 times per second.

```
uint32_t msecs = 0;
void systick_periodic_task(void) {
  msecs++;
  if ((msecs & 0xff) == 0) { /* 8 low-order bits are zero, */
                             /* happens every 256ms        */
    ... /* toggle an LED */
  }
}
```

   Note that we have avoided division and avoided computing remainders, both of which could be slow. In large programs that require modularity, `systick_periodic_task` can call periodic tasks of other modules, say the switch debounce module.

2. Implement LED blinking using the `systick` module. The LED should toggle about every second; either 1000 or 1024 ms is fine.

3. A good way to implement short delays in the microsecond range is with a loop. This strategy does waste processor cycles (and power), but if the delay is short and not too frequent, the overall waste is small. Implement a function `busywait(uint32_t microseconds)` that waits in a loop and returns after a given number of microseconds. An initialization function `busywaitInit()`

should calibrate the number of iterations per μs (or the number of μs per iteration on slow processors). The calibration constant should be stored in a variable. Assume that `busywaitInit` will be called before interrupts are enabled. Test the `busywait` module with a program that uses it to blink an LED once per second.

4. If you turn an LED on and off rapidely (say faster than 100 Hz), the human eye perceives the light as steady but dimmed, not as blinking. Dimming a light source this way is called *pulse-width modulation*. Other actuators, like motors, can also be controlled this way. Implement a program that allows the user to dim one of the strong LEDs on the board using PWM. Use the LCD backlight or the RGB LED rather than the weaker red indicator LEDs. The user should be able to increase the light intensity by pushing the joystick switch to the right and decrease it by pushing the joystick to the left. There should be 9 levels of intensity (off, 12.5% of the time on, 25% of the time on, 37.5% of the time on, up to always on).

5. Now change the levels of intensity to a logarithmic scale: off, $1/256$ of the time, $2/256$, $4/256$, $8/256$, and so on. Does this scale look more natural than the linear scale or less natural?

6. The PWM frequency is a design parameter. What do you think are the negative aspects of a very high PWM frequency? What are the negative aspects of a slow PWM frequency?

7. The intensity of an incandescent bulb can also be controlled with PWM. Does the bulb stop emitting light during the off period of the PWM cycle?

8. Additional ideas:

   (a) Generalize the `systick` module so that `systick_periodic_task` is called only every `SYSTICK_QUANTUM` milliseconds, not every millisecond. The module can assume that `SYSTICK_QUANTUM` is an integer.

   (b) Suppose that some other module `m` is a client of `systick` and that it requires a `SYSTICK_QUANTUM` that divides `M_QUANTUM`. How can the module test at compile time that the constraint holds? Assume that the sources of both modules are included into a main program, and that `systick` is included first.

   (c) Write a program that changes the color of the RGB LED every time a button is pressed (cycle between 6 colors) using PWM. Can you control the color and the intensity separately?