ECE 271 – Microcomputer Architecture and Applications Lecture 19

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

31 March 2022

Announcements

- Read Chapter 15.4

Advanced Control-Timers (TIM1/TIM8)

- Note these are *not* the same as the SysTick timer
- Chapter 30 (p907) of STM manual
- 16-bit auto-reload timers
- 16-bit prescalar
- Unrelated to the other TIM timers
- Each timer has 6 channels
- Key registers
  - Counter register `TIM1_CNT`
  - Prescaler register `TIM1_PSC`

- Auto-reload register `TIM1_ARR`
- Repetition count register `TIM1_RCR`

- Upcounting mode
  - Counts from 0 to ARR then restarts at 0
  - The repetition counter says how many times to count before triggering an interrupt?
- Downcounting mode
  - Counts starting at ARR down to 0, reloads ARR
- Center-count mode
  - Counts up then down then up then down

Various Modes

- Can put counter in various modes. We'll use some of them next lab.
- Like 100 pages on this.
- We are setting PWM mode.
- Period set by ARR register, duty cycle by CCMR register
- Can do complicated things like assymmetric mode (with phase shift) or have different channels with different phases.
- Complicated way of setting phase

- Can generate both signal and inverse, and can add “dead time” to the transition
- Can xor together to generate more complex waveforms.

Where does the output go?

- This is configurable too.
- Can put to some but not all of the GPIO pins. Conveniently PE8 (green LED) gets the TIM1 output, but CH1N (so the inverted output)
- Have to set the AF (alternate function) register to pick. Somehow wasn't able to find where these are listed but it is in Appendix I of the textbook.

General Purpose Timers (TIM2/TIM3/TIM4/TIM5)

- Chapter 31 of STM manual
- They are similar to TIM1/TIM8, not immediately obvious how they are different.

Looking ahead to Lab #9

Input Capture

- Input capture is measuring the time between two transitions on a signal
  - Rising/Rising or Falling/Falling or Rising/Falling or Falling/Rising
  - When transition happens, the current timer count (CNT) is saved to the CCR (Compare and capture register)
  - Also an interrupt (or other event such as DMA) can be triggered

- Time elapsed can be calculated by taking the previous timestamp and subtracting from the current one.

What can we measure?

- Pulse width, period, and duty cycle

Capture Config

- Input can be external pin, or can be another internal timer
- Edge detector: can be only falling, only rising, or both
- Digital Input Filter: (debouncing), number of events that must happen before reporting an event

What it does

- Wait until the number of transitions reaches the threshold in the input capture prescaler bits and the CCMR register, a capture happens
  - The free-running counter CNT is latched into the CCR register for that channel
  - There is a 16-bit capture/compare for each capture channel 1-4
  - In the status register SR the CCIF (capture/compare interrupt flag) for the channel is set

- If the flag was already set, it means we captured again before anyone took care of the last one. In that case the CCOF flag (capture/compare over-capture flag) is set.
- If interrupt is enabled in the CCIE bit (capture/compare interrupt enable bit) then an interrupt is generated
- Other stuff happens if you are doing DMA

Configuring Input Capture

- For this example using Channel 1 of Timer 4
- Select active input: CC1S bits in CCMR1
  - 00: Channel 1 is output (this is what we used for PWM)
  - 01: Input, mapped to timer input 1 (TI1)
  - 10: Input, mapped to timer input 2 (TI2)
  - 11: Input, mapped to TRC (trigger output of other counter)
- Set the input filter – 4 bits. 0 means none

- Set the active edge. CC1P and CC1NP. (double check manual)
  - CC1NP=0 CC1P=0 then rising
  - CC1NP=0 CC1P=1 then falling
  - CC1NP=1 CC1P=1 then both
- Set the input prescaler (can stride, only activating every 1, 2, 4 or 8 events)
- Enable input capture (CC1E bit in CCER)
- Enable the interrupt, TIE trigger interrupt and UIE update interrupt
- Enable the counter, set the CEN enable bit in the CR1

register.

Overflow

- Our timers only have 16 bits. So if programmed at 1us (1MHz) the counter will overflow after 6.5ms.
- What can we do about that?
- In addition to tracking time, count how many times the counter overflows in between.
- How can we do that? Have it trigger an interrupt and count.
- Total time is $overflows * maxtime + (current - last)$

Simple Interrupt Handler

```c
volatile uint32_t pulse_width=0,last_captured=0,signal_polarity=0;
void TIM4_IRQHandler() {
    uint32_t current_captured;

    if ((TIM4->SR & TIM_SR_CC1IF)!=0) { // check if irq flag set
        current_captured=TIM4->CCR1;  // auto-clears the CC1IF irq flag
        signal_polarity=1-signal_polarity;
        if (signal_polarity==0) { // only cal chwn low
            pulse_width=current_captured-last_captured;
        }
        last_captured=current_captured;
    }
    if ((TIM4->SR & TIM_SR_UIF)!=0) { // check if oflo has taken place
        TIM4->SR &= ~TIM_SR_UIF;  // clear UIF flag to prevent re-enter
    }
}
```

Input Capture with Automatic Reset

- In Section 15.4.2 if you are curious

Lab#9

- First step is input capture, testing with 1Hz from signal generator to make sure your signal capture code is working
- If that works, then we'll test out the ultrasonic distance sensor

Ultrasonic Distance Sensor

- HC-SR04. Generates an ultrasonic signal at 40kHz. Why can't you hear it?
- Receiver detects wave of reflected sound back.
- $Distance = \frac{RoundTripTime \times SpeedofSound}{2}$
- To trigger, send a pulse of 10us to the trigger pin. This makes it send out 8 cycles of 40KHz sound.
- Echo triggers a 5V pulse on the echo pin. Width is proportional to distance.
- $Distance = \frac{PulseWidth(us)}{58} cm$

- $Distance = \frac{PulseWidth(us)}{148} inch$
- Can measure from 2cm to 400cm with a resolution of 0.3cm (pulse width between 150us and 25ms). No echo will give 38ms.
- Hook up 5V, GND. Can take 3.3V trigger. Some inputs 5V tolerant on STM32L.
- Sends 40kHz pulse. Will you hear it?
- Capture time in us. Divide by 148 to get inches.

Why use signal capture?

- Why not send the pulse manually with GPIO and then poll waiting for result?
- How quickly can you read an MMIO value on a board running at 16MHz? How long does each ARM instruction take?

Actually coding this up

- Use the 16MHz HSI clock
- Set up GPIO PE.11 to trigger every 0.065s with a 10us pulse to activate
  Use PWM mode
  Use TIM1 but note we are using Channel 2 (PE11) not inverted channel 1 (PE8) like previous assignment
- Get the echo result on PB.6 and feed to the measurement code
- Do we have to worry about overflow? Yes.

More Floating Point

Floating Point Comparison

- `float f = (5.0 - 1.0/7.0) + (1.0/7.0);`
- `if (f==5.0) printf("Five␣exactly.\n")`
- May work may not. Best way is to have some error (epsilon) and compare if absolute value less than a number.

Special Comparison

- +/- 0 compare equal
- Every NaN not equal to anything, not even itself
- All numbers are greater than -inf but smaller than inf

Floating Point Rounding Rules

- Complex mess, can cause interesting issues
- Especially as in floating point there are extra bits usually kept around for accuracy, but they are rounded off when written out to memory and you have to fit exactly in 32 or 64 bits
- IEEE-754
  - Round to nearest

    What do we do if *exactly* in between?

    round to even

guard bit, round bit, sticky bits

- ◦ Round toward zero (truncate)
- ◦ Round to $+$inf (round up)
- ◦ Round toward -inf (round down)

Floating Point Addition

- Shift smaller fraction to match larger one
- add or subtract based on sign bits
- normalize the sum
- round to appropriate bits
- detect overflow and underflow
- do example
- decimal, $5.2 * 10^0 + 9.5 * 10^1$
  - 0.52*10**1
  - 9.50*10**1

- 10.02
- $1.002 * 10^2$

Floating Point Multiplication

- Identify the sign
- add the exponents
- multiply the fractions (including leading hidden one)
- normalize the results

Transcendentals?

- sin(), cos(), etc
- Lookup tables?
- Newton's approximation
- Taylor series expansion
  For example, $cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - ...$
- cordic – (for when multiplies are slow) lookups, shift, add

Quake Square Root Trick

- Fast inverse square root, popularized by use in Quake source code

```
float InvSqrt(float x) {
    float xhalf=0.5f*x;
    int i=*(int *)&x;
    i=0x5f3759cff - (i>>1);
    x=*(float)&i;
    x=x*(1.5f-xhalf+x*x);
    return x;
}
```

Floating Point Drawbacks

- special hardware
- power hungry, if not commonly used
- chip area, expense
- back in day, special chip
- rounding issues
- money calcs. 1/10 only approximate. .0001100110011
- trouble near zero