ECE 271 – Microcomputer Architecture and Applications Lecture 17

Vince Weaver

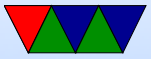
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

28 March 2019

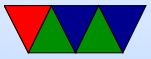
Announcements

- Read Chapter 15
- Note github, ssh keys changed, the whole VM accidentally erased



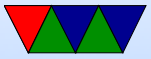
WFI/WFE instructions

- Wait for interrupt or event
- CPU goes into a low-power mode (sleep) waiting for an interrupt
- Can save power when nothing else is going on
- Use SEV to send event to wake from WFE (used on Pi to start secondary processors)



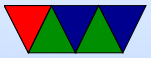
System Timer

- What are times useful for?
- Exact timing
- If you have an OS, a regular timer tick to keep track of time, also context-switch
- Things that run in background. i.e. heartbeat LED, how can you have that run in background? Hook up to timer-tick interrupt



System Timer

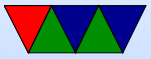
- ARM Cortex-M standardizes on an interface
- Note, it is part of the ARM processor so not documented in the STM32L4 SoC documentation (rather in the "Cortex-M4 Generic Users Guide")
- Four registers
  - SysTick_CTRL (control and status)
  - SysTick_LOAD (reload value)
  - SysTick_VAL (current value)
  - SysTick_CALIB (calibration)



SysTick Control/Status Register

- SysTick_CTRL
  - Bit 0 – ENABLE – enable
  - Bit 1 – TICKINT – enable interrupts when hit zero
  - Bit 2 – CLKSOURCE – 0 = external clock (AHB/8) or 1 (Processor clock)
  - Bit 16 – COUNTFLAG = special event happened
- SysTick_LOAD
  - Bits 23-0 = RELOAD value.

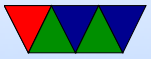
    After counter counts down to zero, reloads



SysTick_LOAD

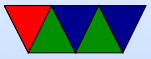
To interrupt every N cycles, set to N-1. 24-bit so up to roughly 16M

- SystTick_VAL
  - Reading gets current value. Writing any value to it resets to 0 (Setting to LOAD on next tick)
- SystTick_CALIB
  - Has the value needed to load to get a certain frequency. STM doc says this is 0x4000270F which gives 1ms when running the clock at 80MHz/8. This is different than what the textbook says.



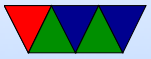
Setting to get the time

- $SysTickPeriod = (1 + SysTick\_Load \times \frac{1}{SysTickClockFreq}$
- So if LOAD is 6 and clock is 1MHz, 7us



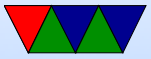
Clocks on the STM32L4Discovery

- HSI16 – high speed internal – 16MHz clock
- MSI – multi-speed internal RC clock – 100kHz to 48MHz
  Note these are RC and best effort and "approximate"
- HSE – high speed external, 4-48MHz
- PLL – phase locked loop, complicated, but in our case essentially a clock multiplier. That's how the board can get up to 80MHz
- Why run at lower speeds? CMOS power equation, power usage is linear with frequency (and square of voltage)

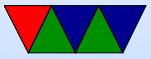


Writing the interrupt handler

- Looks like regular C function
- Our code, just decrements a global variable each time it is called, stopping at zero.
- We call it `SysTick_Handler()`
- How does its address get in the right slot? (address 0x58)
- Linux, we put its address there in an array that gets put in to the right place by the linker script.
- Keil does another trick, there's an existing

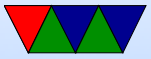


SysTick_Handler() declared as a WEAK symbol. That means it's lesser priority, so if we define a function with the same name the linker will replace the other one with ours.

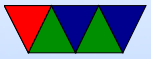


Writing the delay function

- Just loads a value into the global variable that's being decremented by the handler.
- Is this safe? Maybe. What if the write is halfway done and gets interrupted by the interrupt? More a problem if you imagine a two-part value being set, like minutes:seconds. Race condition. Locking? Disabling interrupts?
- Then spin waiting for the value to decrement down to zero.

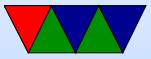


- Does this global value need to be marked volatile?



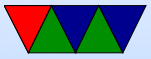
Lab #8 Notes

- Pulse-Width Modulation (PWM)
- Sends stream of 0s and 1, but the average voltage is in between which is useful sometimes
- Dimm an LED by pulsing it rapidly
- Control a servo motor by sending it proper stream of pulses
- Can play audio via 1-bit signal



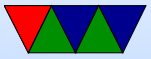
Setting up PWM

- Similar to SysTick timer but a bunch more register settings
- There is a sawtooth carrier signal (draw graph) and a threshold when it is crossed (CCR) and the value it counts down from (ARR)
- $duty\_cycle = \frac{pulseontime(T_{on})}{pulseswitchingperiod(T_s)} \times 100\%$
  $= \frac{T_{on}}{T_{on}+T_{off}} \times 100\%$
- Three factors matter:
  - comparison between timer counter (CNT) and the

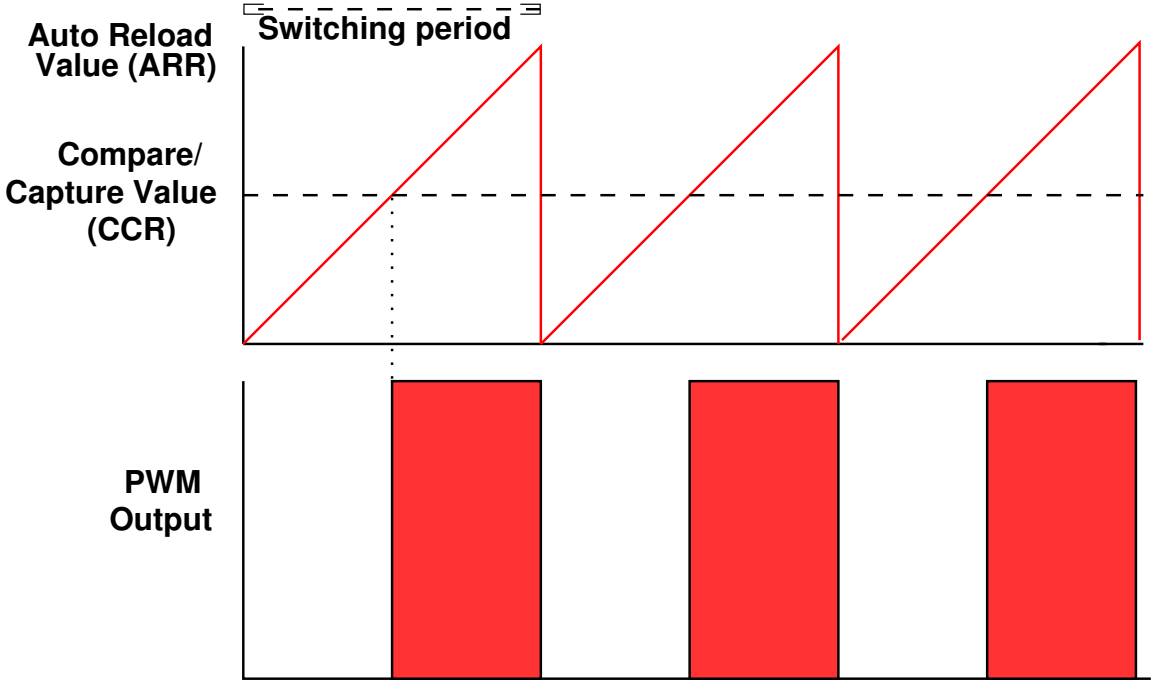


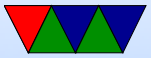
reference value (CCR)

- the PWM output mode
- the polarity bit

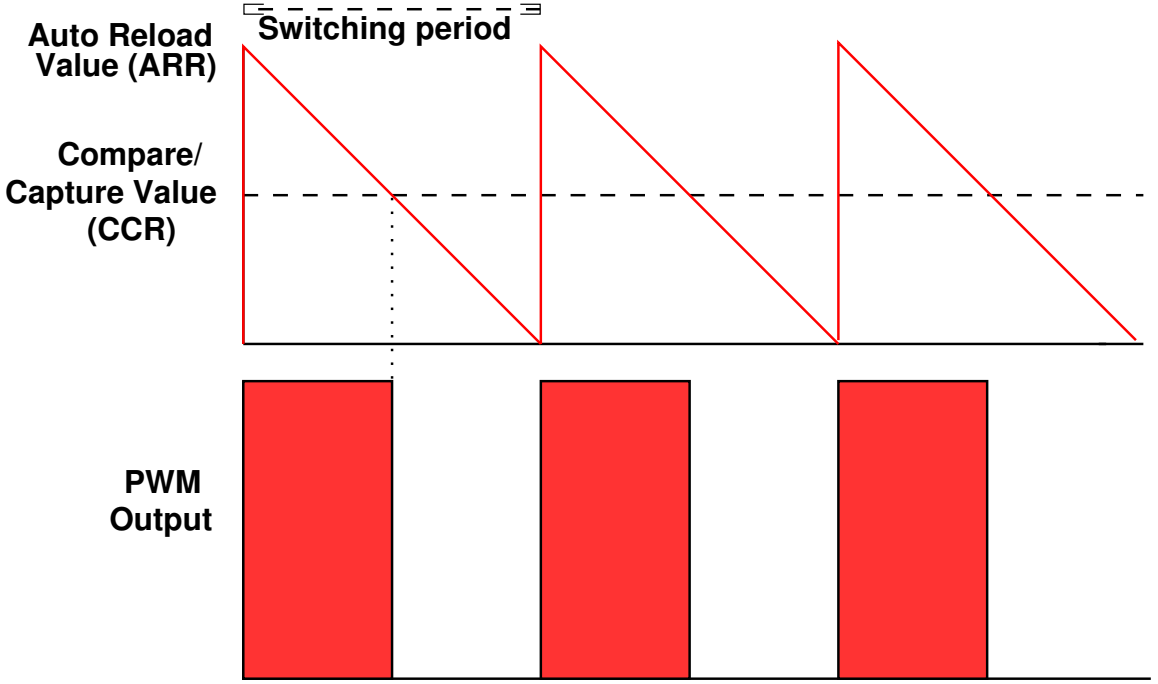


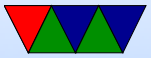
Up-Counting


Switching period
Auto Reload Value (ARR)
Compare/ Capture Value (CCR)
PWM Output

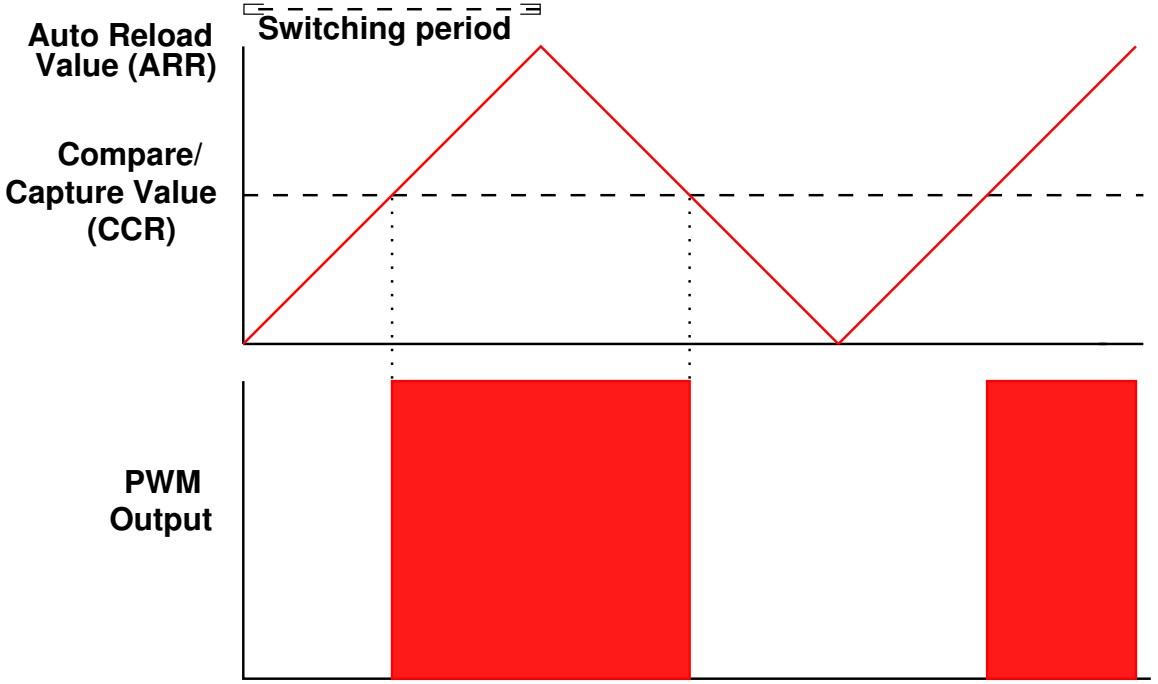


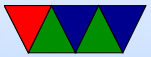
Down-Counting





Center-Counting





Timer can have multiple outputs

- With lots of other features too