Pulse Width Modulation (PWM)

- A controlled means of turning an output an & off.
- We define a frequency and a duty cycle
- image
- Duty cycle get a percentage o the frequency
  - Amount we're on (50% here)
- The duty cycle provides for a control variable
  - RPM goes up/down with duty cycle
  - Current goes up/down with duty cycle
- Frequency is set based on the physical characteristics of our device
  - If frequency too slow/long then it's jumpy
    - Result: all on followed by all off
  - If frequency is too fast then it doesn't see the offs
    - Result: always on
- 2 notes:
  - 1) at physical layer this is how all communications are done(WIFI, send message)
    - Bit timing to indicate 0 vs 1
  - 2) generating a sine wave
    - (assume 8-bit resolution)
    - image
    - We can vary the D.C. to match the wave pattern
    - image
    - To generate any kind of wave form, we define the D.C.(Duty Cycle) values that give us a nice waveform
      - Implement as a table…

Coding Considerations

1) PWMing
   - We use a timer to define our frequency
     - On overflow timer, turn output on
     - Wait for D.C. time units & turn output off
       - How?
     - Recall we can modify the data register

        - Set timer to run for D.C. on & off times
            - On overflow, toggle output & set data register
- The easier way: Output Compare s.f.u.
    - Set up a timer to run with our frequency
        - Compare unit uses this timer
    - On overflow the output automatically turns on
        - The output is tied to the compare unit
    - Has a compare register (1 for each output) set with our D.C. value
        - When timer's data register match the compare register, the output automatically turn off.
        - Give us "set it & forget it"
    - When do we change the compare register value?
        - E.g. setting D.C. after a compare match occurred
            - image
        - Always change value at beginning of a cycle
            - In overflow ISR
        - Microcontroller may latch the compare register at start of a cycle
            - Only change when safe
    - For D.C. of 0% or 100%, turn <u>off</u> the s.f.u (the timer…)
- Notes:
    1) We calculate our duty cycle based on timer frequency/resolution
        - E.g. 16-bit timer
            - 50% D.C.
            - Compare register get 50% of max timer value
                - x7FFF
    2) requires a dedicated timer
        - we can reuse this timer
            a) multiple outputs on the same timer
            b) still time events off it
        - caveat: can only do this if frequency match
    3) we can have more than 1 compare register for one output
        - e.g. count up & down registers
            - gives "fancy" waveforms
    4) our compare match has an interrupt
        - we can do stuff when we know we turn off
            - we can use this for scheduling
                - software timers
                - Ignore the output
                - If we have 3 compare registers, we get 4 interrupt from 1 timer.

A2 required documents

- PH6 output to control fan
- Chapter20 (20.4.3 , 20.10)

A2

- 8pin connector connect to LED pin
- App(BUS)
    - Pin DIO24 - DIO31
- Fan
    - Connect to portH
- APP(Signal)
    - PIN DIO24

Programming considerations(continue):

2) Frequency Capture (Assignment 3)
    - Inverse of a PWM
    - We capture the timer value
        - To determine tie between => frequency
    - Timer overflows and input events are non-deterministic.
    - What if capture occurs at an overflow?
        - Can have capture spread across multiple overflows?
        - Can have captures "just" before/after an overflow
    - We have 2 interrupts that can fire at the "same time"
        - All of this is similar to tracking time
        - But with an extra variable(capture value)
    - Solution
        - Use interrupt priorities:
            - <u>Must</u> control which interrupt can override the other
                - i.e. SIGNAL vs ISR
    - e.g. overflow interrupt fires but capture fires & run first
        - overflow count <u>has not</u> incremented, but we need to consider the overflow
        - since we know the priorities:
            - check overflow interrupt <u>flag</u>
            - if flag <u>set</u>, internally add 1 to the count for our capture calculation
    - what about a capture <u>then</u> an overflow?
        - E.g. we capture a timer value of xfffc (almost overflow)
            - In this scenario, when we check the overflow interrupt flag, it's set
            - Musr consider the capture value before we actually add 1 to the count.

--- End of analog I/O ---

Communications

- We need to talk to off microcontroller components that handle other system tasks.
- We can use parallel or serial methods to talk to deveices
    a. Parallel – addr & data bus(es)
        - Lots of lines (wires)
        - Parallel communications examples:
            - External memory
                - RAM, flash
            - LCD
        - Most often has custom protocol
    b. Serial
        - Use 2-4 lines (wires)
        - Requires bit—by-bit transfers
        - Very desirable(useful), because they are:
            - low noise
            - Run for longer distances(longer wires)
            - E.g. Ethernet
        - We have standard protocols
            - SPI, I^2C, UART (USB, Ethernet)
    - Common examples (communicate to _):
        - E^2
        - Sensor/driver modules
            - See Arduino
        - Other microcontroller
        - Computers (e.g. configuration/diagnostics)
- 2 classes of serial protocols:
    1) Simple devices communications
    2) (*)Inter-processor communications

Device/Peripheral Protocols:

- SPI (serial peripheral interface)
    - A 4-wire synchronous protocol
        - All transmit are referenced off a common clock
    - 1 controller (controls the clock) and 1 or more responders
        - These responders live on the bus - 3 shared lines – SPI bus

          --- or ---
        - Chained
    - Lines:
        - Serial clock (SCK)
        - MOSI (will not learned and don't use)
        - MISO (will not learned and don't use)
        - (will use in Assignment and test)Chip select for each responder ($\overline{cs}$)

- Controller has 1 output for each responder
- Enable responder to talk to on the SPI bus

test1

- FSM – don’t do flow chart