Tristan Video 2:

* PWM Output Explanation (generating sinusoidal outputs using multiple levels)
  + 3 Phase Multi-level converter
  + 16 PWM channel per level (for 1 Phase)
  + Low Freq (60Hz) Leg in addition (for 1 Phase) for H- Bridge
  + Not gates will invert PWMs for H-Bridge
  + HW also handles the deadtime
  + Can also help with battery dynamically balancing
  + HW Currently 4 levels and want to increase and increase to 3 phase
* Main.c Definitions
  + Currently configured 3 phase (would need to change code is wanted 1 phase)
  + Base freq of all PWM outputs
  + Output sinusoidal freq
  + 8 levels per phase (8 \* 3 = 24)
  + Phases:
    - U = 0 (0 - 7) (0)
    - V = 0 + 8 (8 - 15) (120)
    - W = 8 + 8 (16 - 24) (240)
  + Sync syncs up the neg edge 0s of high and low freq waves
  + Dynamically balances (if commented will generate closer to sine wave)
* Testing
  + Refresh
  + Clean
  + Flash
  + Check cpu1 (uncheck cpu2)
* Code
  + Optimized mmc uses optimized pwm to produce the pwms for each level
  + Main does config and setup and loop
* Optimized pwm driver.h
  + OOP approach
    - Structs to store info about module
    - Calling function with pointer to struct so that function can use struct
  + Declarations
    - Module struct to configure pair of PWM peripherals (i.e. clk freq and pwm output freq)
    - Channel struct to configure individual PWM (i.e. specify duty cycle for each)
    - 12 PWM modules with a pair for each (24 total PWM modules)
    - Enum is for pair channels and abbreviations for interrupt conditions
  + Init module (pair of PWMs)
    - Take module object
    - Epwm\_number (1-12)
    - Pwm freq
    - Prescaler clock input (0-32) (already configured in mmc)
  + Init channel
    - Take channel and module objects
    - Channel # enum
    - Initial duty cycle %
  + Set duty cycle
    - Set voltage output of each level
    - Take channel object
    - %
  + Set frequency
    - No reason to set it once PWM has been initialized
  + Registered interrupt handler
    - Configure interrupts to specify which function is called for each interrupt
  + Enable interrupt
    - Initialize interrupt for PWM module with interrupt type specified
* Optimized pwm driver.c
  + Init module
    - Load values
      * Take in inputs and increment based on module and register location
      * Set clock and freq
    - Set up counter mode
      * Check clock boolean
      * Counts up to max then resets to 0 (sawtooth)
      * Disable phase shift
      * Load clk prescaler
    - Sync PWMs together (they count in sync)
      * If 1, send pulse
      * Else shift to adjust
  + Init channel
    - Load values
    - Defines GPIO for specific PWM channel
      * 1- 8/9-12
        + A/B

Pin#/register for mem

* + - Set duty cycle
    - Sync low and high freq legs
    - Config GPIO
    - Set shadow loading and actions
      * Shadow: when you load a new compare value to wait until counter reset to 0
      * Qualifier Actions: what PWM actually forms
        + A/B

Set low

Counter increments up and reaches compare (turns PWM output pin off)

Counter resets 0 and reaches max (turns PWM output pin high)

* + Set duty cycle
    - Calculates compare value based on counter period
      * A/B
  + Set frequency
    - Takes PWM module input and freq and calc timer based period
    - Loaded into timer based period register
  + Registered interrupt handler
    - Takes in and Sets function pointer for whichever PWM # when an interrupt occurs
  + Enable interrupt
    - PWM module peripheral and interrupt peripheral (uses enum interrupt type)
* Optimized mmc module.h
  + Struct
    - # of converter Levels
    - First pwm channel to be used
    - Pwm freq
    - Signal freq
    - Gpio # for output pin of low freq
    - Signal phase shift
    - Cycle counter
    - Counter max
    - Interrupt flag
    - Reordering flag based on levels of charge
    - Interrupt handler #1&2
  + Init
  + Update duty cycle for each level based on normalized voltage input
  + Retrieves SOC (no real software to get this data)
  + Registered interrupt handler
  + Enable interrupt
* Optimized mmc module.c
  + HW limit uses a global array of channel and module objects
  + Init, load, and calculate values
    - For loop cycles through once per PWM module level
      * Figures out which array index and uses info to initialize module and channel
    - Initialize GPIO for function (uses enum 54 - 59, 40 is testing)
  + Update levels
    - Skips over reordering portion (complex and not ideal)
      * Get new SOC in array
      * Find lowest SOC and load into reordered array
    - Check if voltage level is below current level and adjusts by for loops per level
      * +/- cycle of sinusoid
  + Randomizing SOC
  + Registered interrupt handler
  + Enable interrupt
* Main.c
  + TI init
  + Mmc module 3 phases
  + Mmc interrupt handler
  + Mmc interrupt enable
  + For loop interrupt
  + 2 interrupt per mmc module