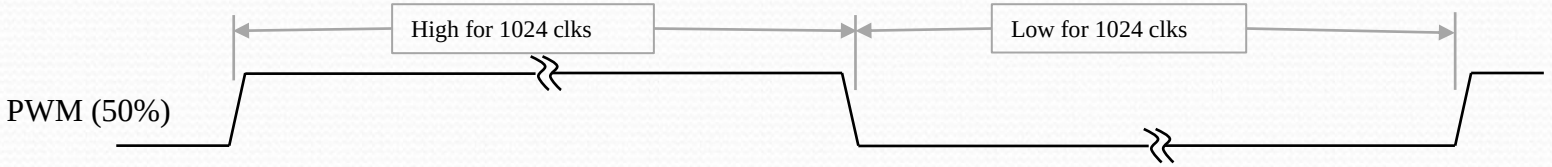
Exercise 9 (HW3 Problem2) (PWM):

- We need to have a way of varying the strength of the drive to the eBike motor. This will be done through **P**ulse **W**idth **M**odulation (PWM).

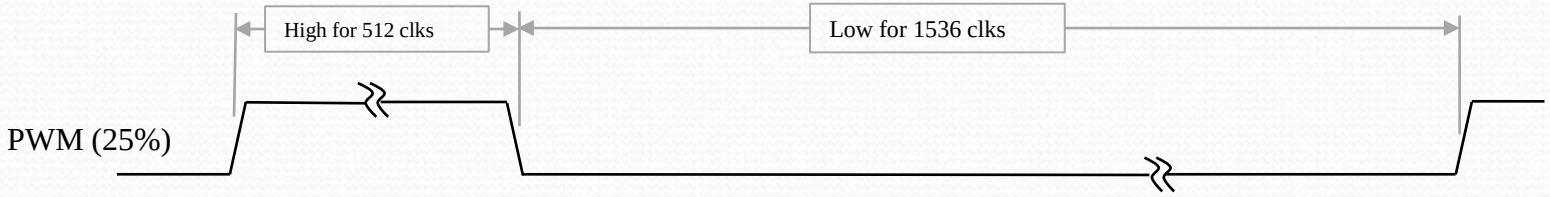
- PWM is commonly used as a simple way of varying intensity. It can be used on an LED. Turn the LED on at full brightness for 100usec then off for 100usec. The human eye will average the light intensity (your retina integrates), so the light will look like the LED is driven at ½ intensity.

- The same works with motor control. Drive the motor coil at full voltage for 50usec and off for 150usec. The inductance in the coil will "average" the current and it will look like the motor is driven at 25%.

- Consider an 11-bit PWM signal being driven at 50% duty cycle. The period of the PWM waveform is 2048 clocks (2^{11}). Since our system clock is 50MHz this is 40usec.



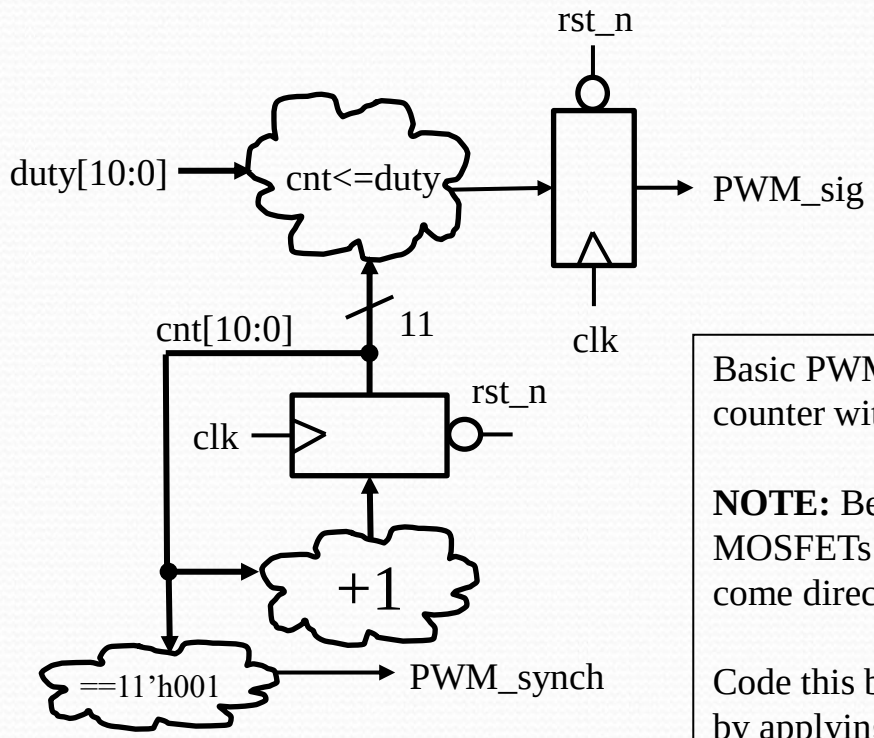
- Second example is 25% duty cycle



Exercise 9 (HW3 Problem2):

A PWM module cannot achieve both zero duty cycle and 100% duty cycle. Think about it. If zero means zero duty then what does 0x3FF mean? It means we are on for 2047 out of 2048 clocks, so not quite 100%. We are fine with this.

Signal:	Dir:	Description:
clk	in	50MHz system clk
rst_n	in	Asynch active low
duty[10:0]	in	Specifies duty cycle (unsigned 11-bit)
PWM_sig	out	PWM signal out (glitch free)
PWM_synch	out	When cnt is 11'h001 output a signal to allow commutator to synch to PWM



When we implement our brushless DC motor driver we will want to synch its commutation to the PWM period.

Basic PWM implementation is not too hard. Just an 11-bit counter with simple comparison logic.

NOTE: Because we use our PWM signal to switch power MOSFETs we cannot afford for it to glitch, therefore, it must come directly out of a flop.

Code this basic PWM and create a testbench for it. Simulate by applying a few different duty[10:0] input values and observing the output.

Submit: PWM.sv, **PWM_tb.sv**, and waveform images.