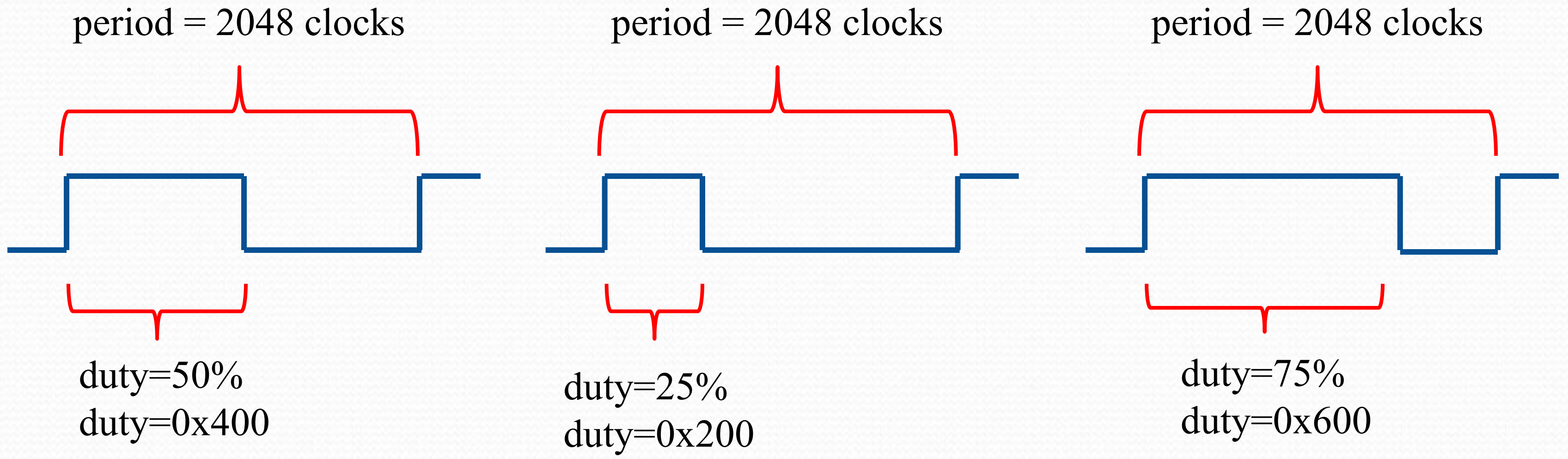
What is PWM (Pulse Width Modulation)



- A PWM signal is used to control the magnitude of something. In our case it will be used to control the magnitude of drive to a DC motor.
- The signal typically will have a fixed frequency and the duty cycle will determine the magnitude.
- Essentially, we are turning the motor on full bore for the duration of the high pulse of the PWM signal, and off for the remainder of the cycle. If the period of the PWM signal is short enough then this bang/bang control is essentially filtered and looks like a nice analog drive at whatever percentage the duty cycle is.

What is PWM (Pulse Width Modulaiton)

- PWM is better than analog drive because it is more efficient. When the switches to drive the motor are on, they are on hard so have very little resistance and therefore negligible I^2R loss. When they are off they have no current therefore no I^2R loss. If we did an actual analog drive to the motor coils we would have I^2R losses over partially on pass elements.

- A frequency in the 20kHz area is appropriate for a DC motor. We have a 50MHz clock on the DE0-Nano, so 11-bit period works pretty nice.

$$\frac{50MHz}{2^{11}} = 24.4\text{kHz}$$

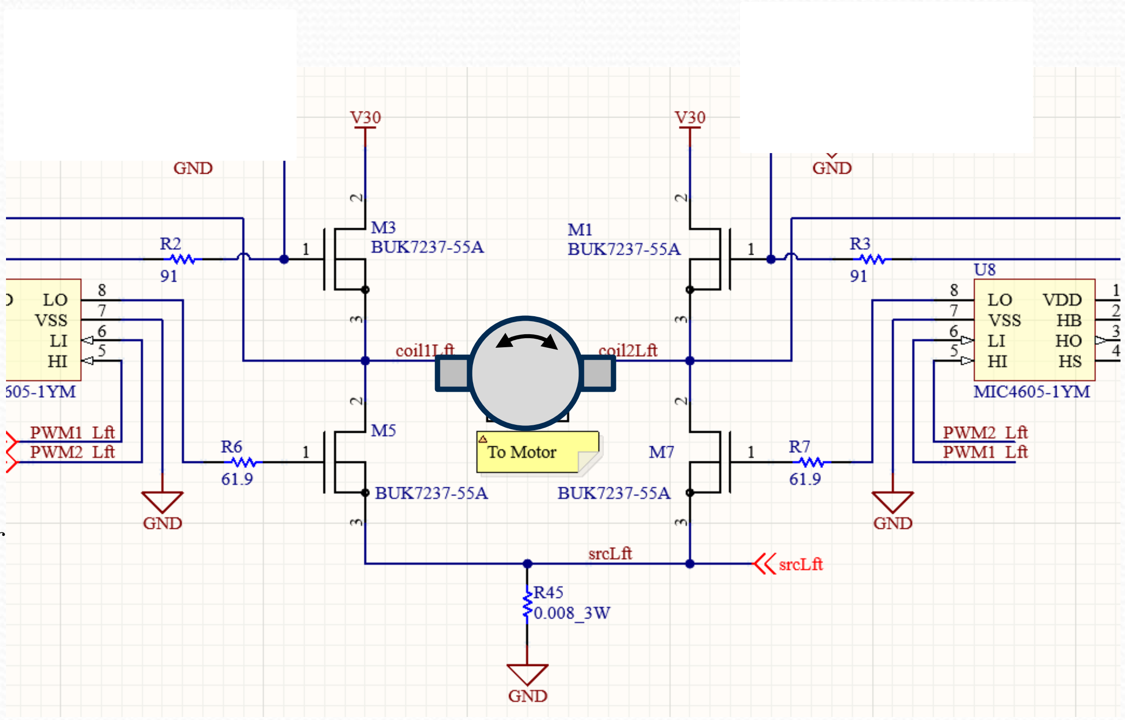
- A PWM module cannot achieve both zero duty cycle and 100% duty cycle. Think about it. If zero means zero duty then what does 0x7FF mean? It means we are on for 2047 out of 2048 clocks, so not quite 100%. We are fine with this.

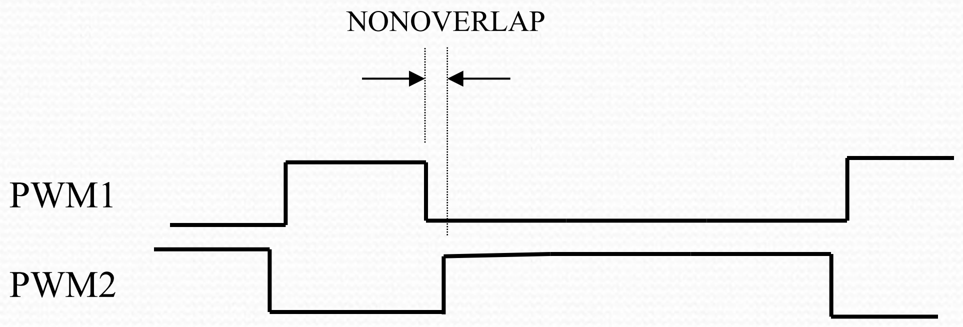
Exercise 10: PWM11 (need for non-overlap)

- The motors can be driven forward or reverse from the 30V battery through the circuit shown. Obviously, the I/Os of our little FPGA cannot drive large motors directly so there are power MOSFETs involved.

- PWM1 & PWM2 are complementary signals *(opposite of each other)* but they also have a non-overlap time *(time in which they are both low)*

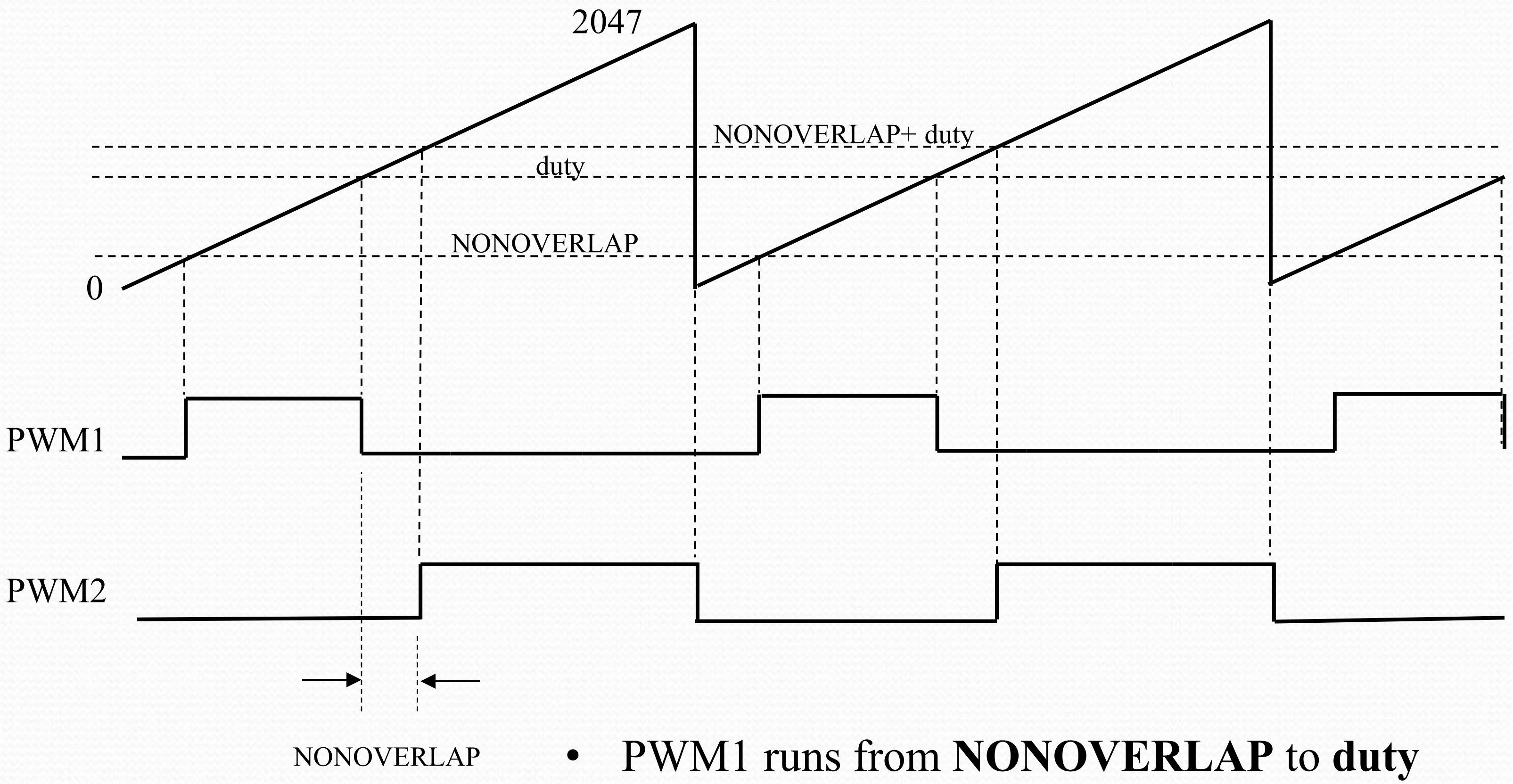
- Why is the non-overlap necessary? Think about it…
  If both the power MOSFETs in a stack were on, then current would be shooting through the transistors from 30V supply to ground. We need to ensure both MOSFETs in a stack are never on at same time.





Exercise 10: PWM11 (Implementing non-overlap)

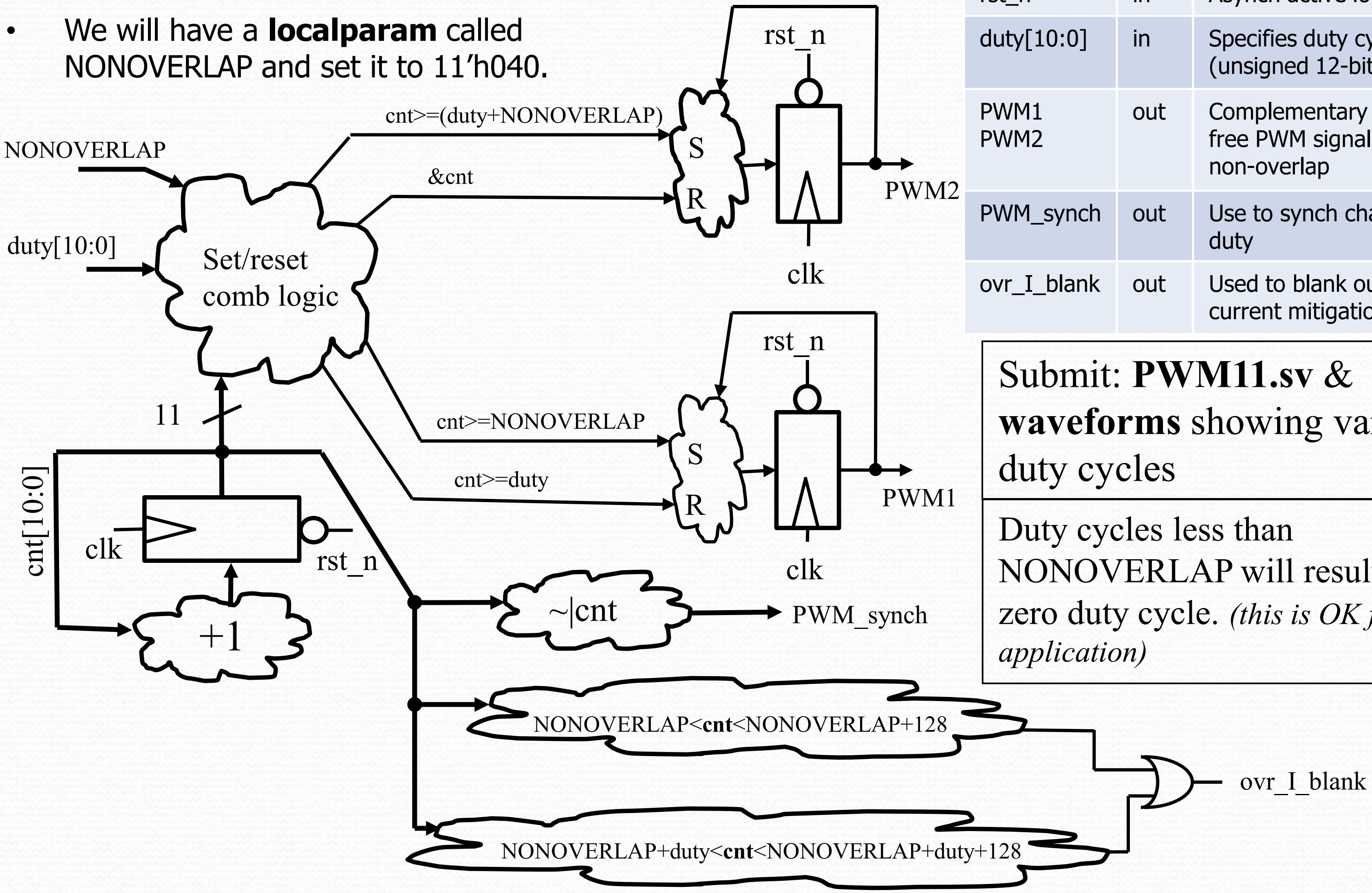
- Think of the 11-bit free-running counter. It ramps from 0 to 2047 then rolls over to 0 again. Think of it plotted as an analog wave.



- PWM1 runs from **NONOVERLAP** to **duty**
- PWM2 runs from **duty+NONOVERLAP** to 2047

Exercise 10 (*PWM11 Implementation*):

- We will have a **localparam** called NONOVERLAP and set it to 11'h040.



Signal:	Dir:	Description:
clk	in	50MHz system clk
rst_n	in	Asynch active low
duty[10:0]	in	Specifies duty cycle (unsigned 12-bit)
PWM1 PWM2	out	Complementary glitch free PWM signals with non-overlap
PWM_synch	out	Use to synch changes in duty
ovr_I_blank	out	Used to blank out over current mitigation

Submit: **PWM11.sv** & **waveforms** showing various duty cycles

Duty cycles less than NONOVERLAP will result in zero duty cycle. *(this is OK for our application)*

What is PWM (PWM_synch & ovr_I_blank)

- The Duty cycle applied to the PWM is generated from another unit. However, changes in the PWM duty cycle should be synched with the PWM cycle so changes to duty do not occur in the middle of a PWM period.

- We need to generate a **PWM_synch** signal to allow the unit driving duty to synchronize changes in duty to the PWM cycle. This signal can simply be when the 11-bit counter is all zeros.

- We have an over current detect circuit that will shut down motor drive if instantaneous current in the motor is too high. Over current detect circuits have a tendancy to false trigger early in the application of a new PWM pulse to the motor.

- We need to generate a blanking signal so we ignore the over current detect if we are in the first 128 clock cycles of either **PWM1** or **PWM2**. This signal can be assigned based on range checking **cnt** vs NONOVERLAP and **duty.**