1 PWM with generic dead band and duty cycle resolution

[5]:
```
from pynq import Overlay
ol=Overlay("pwm_ultrasound_pulser.bit")
from pynq import MMIO
RANGE = 8 # Number of bytes; 8/4 = 2x 32-bit locations which is all we need for␣
 ↪this example

duty_address = ol.ip_dict['axi_gpio_duty']['phys_addr']
duty_register = MMIO(duty_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
duty_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to␣
 ↪output

band_address = ol.ip_dict['axi_gpio_band']['phys_addr']
band_register = MMIO(band_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
band_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to␣
 ↪output

flags_address = ol.ip_dict['axi_gpio_flags']['phys_addr']
flags_register = MMIO(flags_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
flags_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to␣
 ↪output

start_delay_address = ol.ip_dict['axi_gpio_start_delay']['phys_addr']
start_delay_register = MMIO(start_delay_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
start_delay_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state␣
 ↪to output

train_length_address = ol.ip_dict['axi_gpio_train_length']['phys_addr']
```