```
import pyb

class Encoder:
    ''' This class implements a motor driver for the
    ME405 board. '''

    def __init__ (self, timer, pin1, pin2):
        '''
        Initializes the pins and timer channels for an encoder object.
        To create PB6 and PB7 Encoder reader: \n
            pin1 = pyb.Pin.board.PB6 # Pin A \n
            pin2 = pyb.Pin.board.PB7 # Pin B \n
            timer = 4 \n\n
        To create PC6 and PC7 Encoder reader: \n
            pin1 = pyb.Pin.board.PC6 # Pin A \n
            pin2 = pyb.Pin.board.PC7 # Pin B \n
            timer = 8 \n
        @param timer: Specifies the timer for the encoder
        @param pin1: First pin (A) used to read the encoder
        @param pin2: Second pin (B) used to read the encoder
        '''
        print ('Creating an encoder')
        ## First encoder pin associated with the pin1 (A) input parameter
        self.pin_object_1 = pyb.Pin(pin1)
        ## Second encoder pin associated with the pin2 (B) input paramete
        self.pin_object_2 = pyb.Pin(pin2)
        ## Timer number associated with the assigned pins
        self.timer_val = pyb.Timer(timer)
        ## Timer number associated with the assigned pins (Set timer to h
        self.timer_val.init(prescaler=0,period=0xFFFF)
        ## Initializes channel 1 for pin1 (A) timer input
        self.ch1 = self.timer_val.channel(1,pyb.Timer.ENC_AB,pin=self.pin
        ## Initializes channel 2 for pin2 (B) timer input
        self.ch2 = self.timer_val.channel(2,pyb.Timer.ENC_AB,pin=self.pin
        # Instantaneous encoder reading at time of call
        self.__current_count = 0
        # Difference between last count and current count
        self.__delta_count = 0
        # Encoder reading from previous call
        self.__last_count = 0
        # Current absolute position of the encoder
        self.__encoder_val = 0
        print('Encoder object successfully created')

    def read_encoder(self):
        '''
```

```
        Reads the current encoder value
        @return encoder_val
        '''
        # Read current encoder count
        self.__current_count = self.timer_val.counter()
        # Subtract previous reading from current reading
        self.__delta_count = self.__current_count - self.__last_count
        # Account for 16-bit discontinuity phenomena
        if self.__delta_count > 32767:
            self.__delta_count -= 65535
        elif self.__delta_count < -32768:
            self.__delta_count += 65535
        # Add delta to absolute encoder position
        self.__encoder_val += self.__delta_count
        # Store current encoder reading into previous reading
        self.__last_count = self.__current_count
        #print(self.__encoder_val)
        return self.__encoder_val

    def zero_encoder(self):
        '''
        Resets all encoder parameters to zero
        @return encoder_val
        '''
        self.timer_val.counter(0)   # Reset all encoder parameters to zer
        self.__current_count = 0
        self.__last_count = 0
        self.__delta_count = 0
        self.__encoder_val = 0
        return self.__encoder_val
```