```
import array
import micropython
import utime
from machine import Pin
from micropython import const

class InvalidChecksum(Exception):
    pass

class InvalidPulseCount(Exception):
    pass

MAX_UNCHANGED = const(100)
MIN_INTERVAL_US = const(200000)
HIGH_LEVEL = const(50)
EXPECTED_PULSES = const(84)

class DHT11:
    _temperature: float
    _humidity: float

    def __init__(self, pin):
        self._pin = pin
        self._last_measure = utime.ticks_us()
        self._temperature = -1
        self._humidity = -1

    def measure(self):
        current_ticks = utime.ticks_us()
        if utime.ticks_diff(current_ticks, self._last_measure) <
MIN_INTERVAL_US and (
            self._temperature > -1 or self._humidity > -1
        ):
            # Less than a second since last read, which is too soon
according
            # to the datasheet
            return

        self._send_init_signal()
        pulses = self._capture_pulses()
        buffer = self._convert_pulses_to_buffer(pulses)
        self._verify_checksum(buffer)

        self._humidity = buffer[0] + buffer[1] / 10
        self._temperature = buffer[2] + buffer[3] / 10
        self._last_measure = utime.ticks_us()

    @property
    def humidity(self):
        self.measure()
        return self._humidity

    @property
    def temperature(self):
```

```
        self.measure()
        return self._temperature

    def _send_init_signal(self):
        self._pin.init(Pin.OUT, Pin.PULL_DOWN)
        self._pin.value(1)
        utime.sleep_ms(50)
        self._pin.value(0)
        utime.sleep_ms(18)

    @micropython.native
    def _capture_pulses(self):
        pin = self._pin
        pin.init(Pin.IN, Pin.PULL_UP)

        val = 1
        idx = 0
        transitions = bytearray(EXPECTED_PULSES)
        unchanged = 0
        timestamp = utime.ticks_us()

        while unchanged < MAX_UNCHANGED:
            if val != pin.value():
                if idx >= EXPECTED_PULSES:
                    raise InvalidPulseCount(
                        "Got more than {}
pulses".format(EXPECTED_PULSES)
                    )
                now = utime.ticks_us()
                transitions[idx] = now - timestamp
                timestamp = now
                idx += 1

                val = 1 - val
                unchanged = 0
            else:
                unchanged += 1
        pin.init(Pin.OUT, Pin.PULL_DOWN)
        if idx != EXPECTED_PULSES:
            raise InvalidPulseCount(
                "Expected {} but got {}
pulses".format(EXPECTED_PULSES, idx)
            )
        return transitions[4:]

    def _convert_pulses_to_buffer(self, pulses):
        """Convert a list of 80 pulses into a 5 byte buffer
        The resulting 5 bytes in the buffer will be:
            0: Integral relative humidity data
            1: Decimal relative humidity data
            2: Integral temperature data
            3: Decimal temperature data
            4: Checksum
        """
```

```
    # Convert the pulses to 40 bits
    binary = 0
    for idx in range(0, len(pulses), 2):
        binary = binary << 1 | int(pulses[idx] > HIGH_LEVEL)

    # Split into 5 bytes
    buffer = array.array("B")
    for shift in range(4, -1, -1):
        buffer.append(binary >> shift * 8 & 0xFF)
    return buffer

def _verify_checksum(self, buffer):
    # Calculate checksum
    checksum = 0
    for buf in buffer[0:4]:
        checksum += buf
    if checksum & 0xFF != buffer[4]:
        raise InvalidChecksum()
```