import RPi.GPIO as GPIO

import time

input\_pin = 27

timer\_ms = 1000

GPIO.setmode(GPIO.BCM)

GPIO.setup(input\_pin, [GPIO.IN](http://gpio.in/))

start\_time = time.time()

pulses = 0

while (time.time() - start\_time) < timer\_ms/1000:

if GPIO.input(input\_pin) == 1:

pulses += 1

frequency = pulses \* 1000 / timer\_ms

print("Frequency:", frequency, "Hz")

GPIO.cleanup()

[03/02, 1:58 pm] Sonal: from micropython import const

import rp2

from rp2 import PIO, asm\_pio

@asm\_pio(sideset\_init=PIO.OUT\_HIGH)

def gate():

    """PIO to generate gate signal."""

    mov(x, osr) # load gate time (in clock pulses) from osr

    wait(0, pin, 0) # wait for input to go low

    wait(1, pin, 0) # wait for input to go high - effectively giving us rising edge detection

    label("loopstart")

    jmp(x\_dec, "loopstart") .side(0) # keep gate low for time programmed by setting x reg

    wait(0, pin, 0) # wait for input to go low

    wait(1, pin, 0) .side(1) # set gate to high on rising edge

    irq(block, 0) # set interrupt 0 flag and wait for system handler to service interrupt

    wait(1, irq, 4) # wait for irq from clock counting state machine

    wait(1, irq, 5) # wait for irq from pulse counting state machine

@asm\_pio()

def clock\_count():

    """PIO for counting clock pulses during gate low."""

    mov(x, osr) # load x scratch with max value (2^32-1)

    wait(1, pin, 0) # detect falling edge

    wait(0, pin, 0) # of gate signal

    label("counter")

    jmp(pin, "output") # as long as gate is low //

    jmp(x\_dec, "counter") # decrement x reg (counting every other clock cycle - have to multiply output value by 2)

    label("output")

    mov(isr, x) # move clock count value to isr

    push() # send data to FIFO

    irq(block, 4) # set irq and wait for gate PIO to acknowledge

@asm\_pio(sideset\_init=PIO.OUT\_HIGH)

def pulse\_count():

    """PIO for counting incoming pulses during gate low."""

    mov(x, osr) # load x scratch with max value (2^32-1)

    wait(1, pin, 0)

    wait(0, pin, 0) .side(0) # detect falling edge of gate

    label("counter")

    wait(0, pin, 1) # wait for rising

    wait(1, pin, 1) # edge of input signal

    jmp(pin, "output") # as long as gate is low //

    jmp(x\_dec, "counter") # decrement x req counting incoming pulses (probably will count one pulse less than it should - to be checked later)

    label("output")

    mov(isr, x) .side(1) # move pulse count value to isr and set pin to high to tell clock counting sm to stop counting

    push() # send data to FIFO

    irq(block, 5) # set irq and wait for gate PIO to acknowledge

def init\_sm(freq, input\_pin, gate\_pin, pulse\_fin\_pin):

    """Starts state machines."""

    gate\_pin.value(1)

    pulse\_fin\_pin.value(1)

    max\_count = const((1 << 32) - 1)

    sm0 = rp2.StateMachine(0, gate, freq=freq, in\_base=input\_pin, sideset\_base=gate\_pin)

    sm0.put(freq)

    sm0.exec("pull()")

    sm1 = rp2.StateMachine(1, clock\_count, freq=freq, in\_base=gate\_pin, jmp\_pin=pulse\_fin\_pin)

    sm1.put(max\_count)

    sm1.exec("pull()")

    sm2 = rp2.StateMachine(2, pulse\_count, freq=freq, in\_base=gate\_pin, sideset\_base = pulse\_fin\_pin, jmp\_pin=gate\_pin)

    sm2.put(max\_count-1)

    sm2.exec("pull()")

    sm1.active(1)

    sm2.active(1)

    sm0.active(1)

    return sm0, sm1, sm2

if \_\_name\_\_ == "\_\_main\_\_":

    from machine import Pin

    import uarray as array

    update\_flag = False

    data = array.array("I", [0, 0])

    def counter\_handler(sm):

        print("IRQ")

        global update\_flag

        if not update\_flag:

            sm0.put(125\_000)

            sm0.exec("pull()")

            data[0] = sm1.get() # clock count

            data[1] = sm2.get() # pulse count

            update\_flag = True

    sm0, sm1, sm2 = init\_sm(125\_000\_000, Pin(15, Pin.IN, Pin.PULL\_UP), Pin(14, Pin.OUT), Pin(13, Pin.OUT))

    sm0.irq(counter\_handler)

    print("Starting test")

    i = 0

    while True:

        if update\_flag:

            clock\_count = 2\*(max\_count - data[0]+1)

            pulse\_count = max\_count - data[1]

            freq = pulse\_count \* (125000208.6 / clock\_count)

            print(i)

            print("Clock count: {}".format(clock\_count))

            print("Input count: {}".format(pulse\_count))

            print("Frequency: {}".format(freq))

            i += 1

            update\_flag = False