import machine

import time

# Set up LED output pin

led\_pin = machine.Pin(2, machine.Pin.OUT)

# Set up analogue input pin and max voltage range

analog\_pin = machine.ADC(0)

analog\_max\_range = 3300

# Set up error LED output pin

error\_led\_pin = machine.Pin(3, machine.Pin.OUT)

# Set up serial communication

uart = machine.UART(1, 9600)

# Define the tasks

def task1():

# Output a digital signal using LED

led\_pin.on()

time.sleep\_us(200)

led\_pin.off()

time.sleep\_us(50)

led\_pin.on()

time.sleep\_us(50)

led\_pin.off()

time.sleep\_us(50)

led\_pin.off()

time.sleep\_us(50)

led\_pin.on()

time.sleep\_us(30)

led\_pin.off()

time.sleep\_us(50)

led\_pin.on()

def task2():

# Measure the frequency of a 3.3v square wave signal, once every 20ms

signal\_frequency = machine.freq\_counter(14)

if signal\_frequency >= 1000:

scaled\_frequency = 99

elif signal\_frequency <= 333:

scaled\_frequency = 0

else:

scaled\_frequency = int((signal\_frequency - 333) \* 0.067)

return scaled\_frequency

def task3():

# Measure the frequency of a second 3.3v square wave signal, once every 8ms

signal\_frequency = machine.freq\_counter(15)

if signal\_frequency >= 1000:

scaled\_frequency = 99

elif signal\_frequency <= 500:

scaled\_frequency = 0

else:

scaled\_frequency = int((signal\_frequency - 500) \* 0.05)

return scaled\_frequency

def task4():

# Read one analogue input, and compute a filtered analogue value, by averaging the last 4 readings

readings = []

for i in range(4):

readings.append(analog\_pin.read())

time.sleep\_ms(5) # wait 5ms between readings

avg\_analog\_in = sum(readings) / 4

# Visualize error if average\_analog\_in > half of maximum range

if avg\_analog\_in > (analog\_max\_range / 2):

error\_led\_pin.on()

else:

error\_led\_pin.off()

return avg\_analog\_in

def task5():

# Log frequency values in comma delimited format once every 100ms

freq2 = task2()

freq3 = task3()

uart.write("%d,%d\n" % (freq2, freq3))

# Define the task schedule

task\_list = [task1, task2, task3, task4, task5]

task\_period = [4, 20, 8, 20, 100] # in ms

task\_last\_time = [0] \* len(task\_list)

# Define the main loop

while True:

current\_time = time.ticks\_ms()

for i in range(len(task\_list)):

if time.ticks\_diff(current\_time, task\_last\_time[i]) >= task\_period[i]:

task\_last\_time[i] = current\_time

task\_list[i]()

import time

import machine

import ustruct

led = machine.Pin(2, machine.Pin.OUT) # LED pin

adc = machine.ADC(0) # Analogue input pin

baudrate = 9600 # Serial port baud rate

# Function to output a digital signal using LED

def task1():

led.value(1)

time.sleep\_us(200)

led.value(0)

time.sleep\_us(50)

led.value(1)

time.sleep\_us(30)

led.value(0)

time.sleep\_us(50)

led.value(0)

time.sleep\_us(50)

led.value(1)

time.sleep\_us(30)

led.value(0)

# Function to measure the frequency of a 3.3v square wave signal

def task2():

start\_time = time.ticks\_us()

while time.ticks\_diff(time.ticks\_us(), start\_time) < 20000:

pass

return int((1 / (time.ticks\_diff(time.ticks\_us(), start\_time) / 1000000)) \* 2)

# Function to measure the frequency of a second 3.3v square wave signal

def task3():

start\_time = time.ticks\_us()

while time.ticks\_diff(time.ticks\_us(), start\_time) < 8000:

pass

return int((1 / (time.ticks\_diff(time.ticks\_us(), start\_time) / 1000000)) \* 2)

# Function to read one analogue input and compute a filtered analogue value

def task4():

readings = []

for i in range(4):

readings.append(adc.read())

average = sum(readings) / 4

if average > (3.3 / 2):

led.value(1)

else:

led.value(0)

return average

# Function to log information to the serial port

def task5(freq2, freq3, avg):

if freq2 < 333:

freq2 = 0

elif freq2 > 1000:

freq2 = 99

else:

freq2 = int((freq2 - 333) / 6.67)

if freq3 < 500:

freq3 = 0

elif freq3 > 1000:

freq3 = 99

else:

freq3 = int((freq3 - 500) / 5)

data = ustruct.pack("<ii", freq2, freq3)

machine.UART(1, baudrate).write(data)

# Main loop to execute the tasks periodically

while True:

task1()

freq2 = task2()

freq3 = task3()

avg = task4()

task5(freq2, freq3, avg)

time.sleep\_ms(350)

import time

import ustruct

# LED pin

led\_pin = 2

# Analogue input pin

adc\_pin = 0

# Serial port baud rate

baudrate = 9600

# Function to output a digital signal using LED

def task1():

led\_value = 1

time.sleep(0.0002)

led\_value = 0

time.sleep(0.00005)

led\_value = 1

time.sleep(0.00003)

led\_value = 0

time.sleep(0.00005)

led\_value = 0

time.sleep(0.00005)

led\_value = 1

time.sleep(0.00003)

led\_value = 0

# Function to measure the frequency of a 3.3v square wave signal

def task2():

start\_time = time.time()

while time.time() - start\_time < 0.02:

pass

return int((1 / (time.time() - start\_time)) \* 2)

# Function to measure the frequency of a second 3.3v square wave signal

def task3():

start\_time = time.time()

while time.time() - start\_time < 0.008:

pass

return int((1 / (time.time() - start\_time)) \* 2)

# Function to read one analogue input and compute a filtered analogue value

def task4():

readings = []

for i in range(4):

readings.append(adc\_pin)

average = sum(readings) / 4

if average > (3.3 / 2):

led\_value = 1

else:

led\_value = 0

return average

# Function to log information to the serial port

def task5(freq2, freq3, avg):

if freq2 < 333:

freq2 = 0

elif freq2 > 1000:

freq2 = 99

else:

freq2 = int((freq2 - 333) / 6.67)

if freq3 < 500:

freq3 = 0

elif freq3 > 1000:

freq3 = 99

else:

freq3 = int((freq3 - 500) / 5)

data = ustruct.pack("<ii", freq2, freq3)

# Write data to the serial port

with open("/dev/serial1", "wb") as ser:

ser.write(data)

# Main loop to execute the tasks periodically

while True:

task1()

freq2 = task2()

freq3 = task3()

avg = task4()

task5(freq2, freq3, avg)

time.sleep(0.1)

import time

import ustruct

led\_pin = 2 # LED pin

adc\_pin = 0 # Analogue input pin

baudrate = 9600 # Serial port baud rate

# Function to output a digital signal using LED

def task1():

led\_value = 1

for i in range(8):

if led\_value == 1:

time.sleep\_us(200)

led\_value = 0

else:

time.sleep\_us(50)

led\_value = 1

return led\_value

# Function to measure the frequency of a 3.3v square wave signal

def task2():

start\_time = time.time()

while time.time() - start\_time < 0.02:

pass

return int((1 / (time.time() - start\_time)) \* 2)

# Function to measure the frequency of a second 3.3v square wave signal

def task3():

start\_time = time.time()

while time.time() - start\_time < 0.008:

pass

return int((1 / (time.time() - start\_time)) \* 2)

# Function to read one analogue input and compute a filtered analogue value

def task4():

readings = []

for i in range(4):

readings.append(adc\_pin.read())

average = sum(readings) / 4

if average > (3.3 / 2):

led\_value = 1

else:

led\_value = 0

return average, led\_value

# Function to log information to the serial port

def task5(freq2, freq3, avg, led\_value):

if freq2 < 333:

freq2 = 0

elif freq2 > 1000:

freq2 = 99

else:

freq2 = int((freq2 - 333) / 6.67)

if freq3 < 500:

freq3 = 0

elif freq3 > 1000:

freq3 = 99

else:

freq3 = int((freq3 - 500) / 5)

data = ustruct.pack("<ii", freq2, freq3)

serial\_port.write(data)

# Main loop to execute the tasks periodically

while True:

led\_value = task1()

freq2 = task2()

freq3 = task3()

avg, led\_value = task4()

task5(freq2, freq3, avg, led\_value)

time.sleep(0.1)

import time

import ustruct

led\_pin = 2 # LED pin

analogue\_input\_pin = 0 # Analogue input pin

baudrate = 9600 # Serial port baud rate

# Function to output a digital signal using LED

def task1():

led\_value = 1

time.sleep\_us(200)

led\_value = 0

time.sleep\_us(50)

led\_value = 1

time.sleep\_us(30)

led\_value = 0

time.sleep\_us(50)

led\_value = 0

time.sleep\_us(50)

led\_value = 1

time.sleep\_us(30)

led\_value = 0

# Function to measure the frequency of a 3.3v square wave signal

def task2():

start\_time = time.time()

while time.time() - start\_time < 0.02:

pass

return int((1 / (time.time() - start\_time)) \* 2)

# Function to measure the frequency of a second 3.3v square wave signal

def task3():

start\_time = time.time()

while time.time() - start\_time < 0.008:

pass

return int((1 / (time.time() - start\_time)) \* 2)

# Function to read one analogue input and compute a filtered analogue value

def task4():

readings = []

for i in range(4):

readings.append(analogue\_input\_pin)

average = sum(readings) / 4

if average > (3.3 / 2):

led\_value = 1

else:

led\_value = 0

return average

# Function to log information to the serial port

def task5(freq2, freq3, avg):

if freq2 < 333:

freq2 = 0

elif freq2 > 1000:

freq2 = 99

else:

freq2 = int((freq2 - 333) / 6.67)

if freq3 < 500:

freq3 = 0

elif freq3 > 1000:

freq3 = 99

else:

freq3 = int((freq3 - 500) / 5)

data = ustruct.pack("<ii", freq2, freq3)

# code to write data to the serial port

# Main loop to execute the tasks periodically

while True:

task1()

freq2 = task2()

freq3 = task3()

avg = task4()

task5(freq2, freq3, avg)

time.sleep(0.35)

The program implements a machine monitor system using ESP32C3 kit without an interrupt. The system executes 5 periodic tasks with specified timings.

The first task task1() outputs a digital signal using LED. The LED should be turned HIGH for 200μs, then LOW for 50μs, then HIGH, then LOW for 50μs, then LOW for 50μs, then HIGH, then HIGH again for 30μs, then LOW for 50μs, then HIGH, and repeat the same pattern once every 4ms.

The second task task2() measures the frequency of a 3.3v square wave signal, once every 20ms. The frequency should be in the range 333Hz to 1000Hz and the signal should be a standard square wave (50% duty cycle). The accuracy should be 2.5% or acceptable.

The third task task3() measures the frequency of a second 3.3v square wave signal, once every 8ms. The frequency should be in the range 500Hz to 1000Hz and the signal should be a standard square wave (50% duty cycle). The accuracy should be 2.5% or acceptable.

| **Task** | **Period (ms)** | **Rate (Hz)** | **Start Time (ms)** | **End Time (ms)** |
| --- | --- | --- | --- | --- |
| Task 1 | 4 | 250 | 0 | 4 |
| Task 2 | 20 | 50 | 4 | 24 |
| Task 3 | 8 | 125 | 8 | 16 |
| Task 4 | 20 | 50 | 24 | 44 |
| Task 5 | 100 | 10 | 100 | 200 |

The table shows the 5 tasks in the program, their period in milliseconds, rate in Hertz, start time in milliseconds, and end time in milliseconds. The start time of each task is calculated based on the end time of the previous task. For example, the start time of Task 2 is 4ms, which is the end time of Task 1. The end time of each task is calculated as **start time + period**. For example, the end time of Task 1 is 4ms, which is **0 + 4**.

The fourth task task4() reads one analogue input, and computes a filtered analogue value by averaging the last 4 readings. The analogue input should be connected to a maximum of 3.3Volts, using a potentiometer. The task also visualizes an error (using a LED) whenever the average analogue input is greater than half of the maximum range.

The fifth task task5() logs the following information once every 100ms in comma-delimited format to the serial port at a baud rate of 9600 bits per second.

a) Frequency value measured by Task 2 (Hz, as integer)

b) Frequency value measured by Task 3 (Hz, as integer)

The frequencies should be scaled and bounded between 0 to 99. For example, the output “0,0” will mean that Task 2 has measured a frequency of 333Hz or less, and Task 3 has measured a frequency of 500Hz or less. The output “99,99” will mean that both tasks have measured frequencies equal or above 1000Hz.

The main loop while True: executes the tasks periodically. The program uses the time module to keep track of the time and ustruct module to pack the data in the required format.

+-----+-----+-----+-----+-----+-----+-----+-----+

|Task1|Task2|Task3|Task4|Task5|Task1|Task2|Task3|

+-----+-----+-----+-----+-----+-----+-----+-----+

0ms 4ms 8ms 24ms 100ms 104ms 124ms 132ms

In the chart, each column represents a unit of time, and each row represents a task. The width of the row indicates the duration of the task, and the spacing between the rows indicates the inter-task time.

The chart shows that Task 1 is executed first and takes 4ms, then Task 2 is executed and takes 20ms, then Task 3 is executed and takes 8ms, then Task 4 is executed and takes 20ms, then Task 5 is executed and takes 100ms, and the pattern repeats.

The cyclic executive chart provides a simple and clear view of the system's timing behavior, making it easier to understand and analyze the system's performance.

+-----+-----+-----+-----+-----+-----+-----+-----+

|Task1|Task2|Task3|Task4|Task5|Task1|Task2|Task3|

+-----+-----+-----+-----+-----+-----+-----+-----+

0ms 4ms 8ms 24ms 100ms 104ms 124ms 132ms

20ms 8ms 20ms 100ms

In the chart, each column represents a unit of time, and each row represents a task. The width of the row indicates the duration of the task, and the spacing between the rows indicates the inter-task time.

The chart shows that Task 1 is executed first and takes 4ms, then Task 2 is executed and takes 20ms, then Task 3 is executed and takes 8ms, then Task 4 is executed and takes 20ms, then Task 5 is executed and takes 100ms, and the pattern repeats within the specified time constraints.

The chart provides a clear view of the system's timing behavior, making it easier to understand and analyze the system's performance within the specified time constraints.

| **ask** | **Period (ms)** | **Deadline (ms)** | **Start Time (ms)** | **End Time (ms)** |
| --- | --- | --- | --- | --- |
| Task 1 | 4 | 4 | 0 | 4 |
| Task 2 | 20 | 20 | 4 | 24 |
| Task 3 | 8 | 8 | 8 | 16 |
| Task 4 | 20 | 20 | 24 | 44 |
| Task 5 | 100 | 100 | 100 | 200 |
| Task 1 | 4 | 4 | 104 | 108 |
| Task 2 | 20 | 20 | 124 | 144 |
| Task 3 | 8 | 8 | 132 | 140 |
| Task 4 | 20 | 20 | 144 | 164 |
| Task 5 | 100 | 100 | 200 | 300 |