Stress tests for Raspberry Pi 4 and 3B+

Björn Kasper (bjoern.kasper@online.de)

May 19, 2021

Contents

1 Introduction

The aim of this notebook is to stress the Raspberry Pi 4 for deciding between different cases and cooling types.

Sources:

- `https://github.com/nschloe/stressberry`
- `https://www.pragmaticlinux.com/2020/06/check-the-raspberry-pi-cpu-temperature/`

2 Implementation of helper functions

2.1 Load globally used libraries and set plot parameters

[1]:
```
import subprocess
#from time import sleep
from os import cpu_count
import threading
import time

import pandas as pd
import numpy as np
import prettytable as pt

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
%matplotlib inline

# FutureWarning: Using an implicitly registered datetime converter for a matplotlib␣
↪plotting method.
# The converter was registered by pandas on import.
# Future versions of pandas will require you to explicitly register matplotlib␣
↪converters.
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()

#import matplotlib.pyplot as plt
#%matplotlib inline

from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf', 'png')

plt.rcParams['savefig.dpi'] = 80
plt.rcParams['savefig.bbox'] = "tight"

plt.rcParams['figure.autolayout'] = False
plt.rcParams['figure.figsize'] = 10, 6
plt.rcParams['axes.labelsize'] = 18
plt.rcParams['axes.titlesize'] = 20
plt.rcParams['font.size'] = 16
plt.rcParams['lines.linewidth'] = 2.0
plt.rcParams['lines.markersize'] = 8
plt.rcParams['legend.fontsize'] = 14

# Need to install dependent package first via 'apt install cm-super'
plt.rcParams['text.usetex'] = True
plt.rcParams['font.family'] = "serif"
plt.rcParams['font.serif'] = "cm"
#plt.rcParams['text.latex.preamble'] = r'\usepackage{subdepth},␣
↪\usepackage{type1cm}'
#plt.rcParams['text.latex.preamble'] = [r'\usepackage{amsmath}' ,␣
↪r'\usepackage[T1]{fontenc}', r'\usepackage{subdepth}', r'\usepackage{type1cm}']
#plt.rcParams['text.latex.unicode'] = True
```

```
/home/bk/jupyter-env/lib/python3.7/site-packages/ipykernel_launcher.py:25:
DeprecationWarning: `set_matplotlib_formats` is deprecated since IPython 7.23,
directly use `matplotlib_inline.backend_inline.set_matplotlib_formats()`
```

2.2 Variant 1: Function for reading the CPU core temperature

This implementation retrieves the temperature information from the system file /sys/class/thermal/thermal_zone0/temp.

[2]:
```
import os

# Function for reading the CPU core temperature
# Found here: https://www.pragmaticlinux.com/2020/06/
↪check-the-raspberry-pi-cpu-temperature/
def get_cpu_temp_old():
    """
    Obtains the current value of the CPU temperature.
    :returns: Current value of the CPU temperature if successful, zero value␣
↪otherwise.
    :rtype: float
    """
    # Initialize the result.
    result = 0.0
    # The first line in this file holds the CPU temperature as an integer times␣
↪1000.
    # Read the first line and remove the newline character at the end of the string.
    if os.path.isfile('/sys/class/thermal/thermal_zone0/temp'):
        with open('/sys/class/thermal/thermal_zone0/temp') as f:
            line = f.readline().strip()
        # Test if the string is an integer as expected.
        if line.isdigit():
            # Convert the string with the CPU temperature to a float in degrees␣
↪Celsius.
            result = float(line) / 1000
    # Give the result back to the caller.
    return result
```

2.3 Variant 2: Function for reading the CPU core temperature (used here)

This implementation retrieves the temperature information from the command line tool vcgencmd. In the bash console you can get the same result by issuing:

$ vcgencmd measure_temp

[3]:
```
#import subprocess

# Function for reading the CPU core temperature
# Found here: https://github.com/nschloe/stressberry/blob/main/stressberry/main.py
def get_cpu_temp(filename=None):
    """Returns the core temperature in Celsius."""
    if filename is not None:
        with open(filename) as f:
            temp = float(f.read()) / 1000
    else:
        # Using vcgencmd is specific to the raspberry pi
        out = subprocess.check_output(["vcgencmd", "measure_temp"]).decode("utf-8")
        temp = float(out.replace("temp=", "").replace("'C", ""))

    return temp
```

2.4 Function for reading the CPU core frequency

The frequency information is retrieved from the command line tool `vcgencmd` also. In the bash console you can get the same result by issuing:

```
$ vcgencmd measure_clock arm
```

[4]:
```
#import subprocess

# Function for reading the CPU core frequency
# Found here: https://github.com/nschloe/stressberry/blob/main/stressberry/main.py
def get_cpu_freq(filename=None):
    """Returns the CPU frequency in MHz"""
    if filename is not None:
        with open(filename) as f:
            frequency = float(f.read()) / 1000
    else:
        # Only vcgencmd measure_clock arm is accurate on Raspberry Pi.
        # Per: https://www.raspberrypi.org/forums/viewtopic.php?
↪f=63&t=219358&start=25
        out = subprocess.check_output(["vcgencmd", "measure_clock arm"]).
↪decode("utf-8")
        frequency = float(out.split("=")[1]) / 1000000

    return frequency
```

2.5 Helper functions for stressing all cores of the CPU

Stress is created by the command line tool `stress`. It has to be installed first by issuing:

```
$ sudo apt install stress
```

[5]:
```
# Helper function to call the 'stress' command line tool
def stress_cpu(num_cpus, time):
    subprocess.check_call(["stress", "--cpu", str(num_cpus), "--timeout",␣
↪f"{time}s"])
    return
```

[6]:
```
#from time import sleep
#from os import cpu_count

# Function for stressing all cores of the CPU
# Found here: https://github.com/nschloe/stressberry/blob/main/stressberry/main.py
def run_stress(stress_duration=300, idle_duration=120, cores=None):
    """Run stress test for specified duration with specified idle times
    at the start and end of the test.
    """
    if cores is None:
        cores = cpu_count()

    print(f"Preparing to stress [{cores}] CPU Cores for [{stress_duration}]␣
↪seconds")
    print(f"Idling for {idle_duration} seconds...")
    time.sleep(idle_duration)

    print(f"Starting the stress load on [{cores}] CPU Cores for [{stress_duration}]␣
↪seconds")
    stress_cpu(num_cpus=cores, time=stress_duration)
```

```
    print(f"Idling for {idle_duration} seconds...")
    time.sleep(idle_duration)
```

2.6 Helper function to let the CPU cool down

This function is used to let the CPU cool down first to find a stable baseline.

[7]:
```
#from time import sleep

def cpu_cooldown(interval=60, filename=None):
    """Lets the CPU cool down until the temperature does not change anymore."""
    prev_tmp = get_cpu_temp()
    while True:
        time.sleep(interval)
        tmp = get_cpu_temp()
        print(
            f"Current temperature: {tmp:4.1f}°C - "
            f"Previous temperature: {prev_tmp:4.1f}°C"
        )
        if abs(tmp - prev_tmp) < 0.2:
            break
        prev_tmp = tmp
    return tmp
```

2.7 Helper function for handling dataframes

First, a dataframe is created and at the same time the column headers are set. The function `dataframe_add_row()` is used to add the measured values to the dataframe in the form of new rows.

[8]:
```
#import pandas as pd

# Dataframe for the measuring values
df_meas_values = pd.DataFrame(columns=['Time', 'CPU Temperature', 'CPU Frequency'])
```

[9]:
```
def dataframe_add_row(df=None, row=[]):
    if (df is None):
        return

    # Add a row
    df.loc[-1] = row

    # Shift the index
    df.index = df.index + 1

    # Reset the index of dataframe and avoid the old index being added as a column
    df.reset_index(drop=True, inplace=True)
```

2.8 Main worker function

[10]:
```
#import threading
#import time

# Function for running the stress test in another thread while measuring CPU␣
↪temperature and frequency
```

```python
# Found here: https://github.com/nschloe/stressberry/blob/main/stressberry/cli/run.
↪py
def run(argv=None):
    # Cool down first
    print("Awaiting stable baseline temperature ...")
    cpu_cooldown(interval=60)

    # Start the stress test in another thread
    t = threading.Thread(
        target=lambda: run_stress(stress_duration=900, idle_duration=300, cores=4),␣
↪args=()
    )
    # Init event handler for killing the thread
    t.event = threading.Event()
    # Start the thread
    t.start()

    # Init row array
    values_row = []
    # Get starting time
    start_time = time.time()
    while t.is_alive():
        try:
            # Get time relative to starting time and round to 2 decimals
            timestamp = float("{:.1f}".format(time.time() - start_time))
            # Get CPU temperature and round to 2 decimals
            temperature = float("{:.1f}".format(get_cpu_temp()))
            # Get CPU frequency and round to 1 decimal
            frequency = float("{:.1f}".format(get_cpu_freq()))

            values_row = [ timestamp,
                           temperature,
                           frequency ]

            dataframe_add_row(df_meas_values, values_row)

            print(
                    f"Time: {timestamp} s,\t"
                    f"Temperature: {temperature} °C,\t"
                    f"Frequency: {frequency} MHz"
            )

            # Choose the sample interval such that we have a respectable number of␣
↪data points
            t.join(2.0)

        except:
            print("Keyboard Interrupt ^C detected.")
            print("Bye.")
            # Stop the thread by calling the event
            t.event.set()
            break

    # Normalize times so we are starting at '0 s'
    #time0 = df_meas_values['Time'][0]
    # It's a really fancy oneliner - but not necessary at all ...
    #df_meas_values['Time'] = [tm - time0 for tm in df_meas_values['Time']]
```

3 Run the heating test

[ ]:
```
# Clear all data in dataframe
df_meas_values = df_meas_values.iloc[0:0]

run()
```

[13]:
```
display(df_meas_values)
```

```
       Time  CPU Temperature  CPU Frequency
0       0.0             56.9         1000.0
1       2.1             56.9          700.0
2       4.2             56.9          700.0
3       6.3             56.9          700.0
4       8.4             57.5          800.0
..      ...              ...            ...
701  1491.6             64.5          800.0
702  1493.7             63.9          900.0
703  1495.8             64.5         1200.0
704  1497.9             64.5          600.0
705  1500.0             63.4          600.0

[706 rows x 3 columns]
```

4 Save all to CSV files

[14]:
```
# Write dataframe to CSV file
str_file_prefix_b4 = 'RaspiB4JupyterLab_stress_measurement'
str_file_prefix_b3plus = 'RaspiB3plusEPaper_stress_measurement'

#df_meas_values.to_csv(r'./data_files/' + str_file_prefix_b4 +␣
↪'_PlasticCase_woHeatSinks.csv', sep ='\t', index = False, header=True)
#df_meas_values.to_csv(r'./data_files/' + str_file_prefix_b4 +␣
↪'_PlasticCase_wHeatSinks.csv', sep ='\t', index = False, header=True)
#df_meas_values.to_csv(r'./data_files/' + str_file_prefix_b4 +␣
↪'_PlasticCase_wHeatSinksAndFan5V.csv', sep ='\t', index = False, header=True)
#df_meas_values.to_csv(r'./data_files/' + str_file_prefix_b4 +␣
↪'_PlasticCase_wHeatSinksAndFan3V.csv', sep ='\t', index = False, header=True)
#df_meas_values.to_csv(r'./data_files/' + str_file_prefix_b4 +␣
↪'_PlasticCase_wHeatSinksAndFan5Vrev.csv', sep ='\t', index = False, header=True)
#df_meas_values.to_csv(r'./data_files/' + str_file_prefix_b4 +␣
↪'_pinkRaspiCase_wHeatSinks.csv', sep ='\t', index = False, header=True)
#df_meas_values.to_csv(r'./data_files/' + str_file_prefix_b4 +␣
↪'_PlasticCase_wHeatSinksAndNoctuaFan5V.csv', sep ='\t', index = False,␣
↪header=True)
#df_meas_values.to_csv(r'./data_files/' + str_file_prefix_b4 +␣
↪'_PlasticCase_wHeatSinksAndNoctuaFan3V.csv', sep ='\t', index = False,␣
↪header=True)

df_meas_values.to_csv(r'./data_files/' + str_file_prefix_b3plus +␣
↪'_PlasticCase_wHeatSinks.csv', sep ='\t', index = False, header=True)
```

5 Read in the CSV files and display it

5.1 Read in the CSV files in dataframes

[2]:
```
#import pandas as pd

# Helper function for creating dataframes from CSV files
def create_dictionary_from_csv(filename, offset=0, cols_wanted=1):
    my_dataframe = pd.read_csv(filename, sep='\t', index_col=False, decimal='.',␣
 ↪header=offset)

    # Delete all cloumns after the desired ones
    my_dataframe.drop(my_dataframe.columns[cols_wanted:], axis=1, inplace=True)

    return my_dataframe
```

[3]:
```
str_file_prefix_b4 = 'RaspiB4JupyterLab_stress_measurement'
str_file_prefix_b3plus = 'RaspiB3plusEPaper_stress_measurement'

str_file_name_1 = str_file_prefix_b4 + '_PlasticCase_woHeatSinks.csv'
str_file_name_2 = str_file_prefix_b4 + '_PlasticCase_wHeatSinks.csv'
str_file_name_3 = str_file_prefix_b4 + '_PlasticCase_wHeatSinksAndFan5V.csv'
str_file_name_4 = str_file_prefix_b4 + '_PlasticCase_wHeatSinksAndFan3V.csv'
str_file_name_5 = str_file_prefix_b4 + '_PlasticCase_wHeatSinksAndFan5Vrev.csv'
str_file_name_6 = str_file_prefix_b4 + '_pinkRaspiCase_wHeatSinks.csv'
str_file_name_7 = str_file_prefix_b4 + '_PlasticCase_wHeatSinksAndNoctuaFan5V.csv'
str_file_name_8 = str_file_prefix_b4 + '_PlasticCase_wHeatSinksAndNoctuaFan3V.csv'

str_file_name_9 = str_file_prefix_b3plus + '_PlasticCase_wHeatSinks.csv'

df_1_PC_woHeatSinks = create_dictionary_from_csv(filename="./data_files/" +␣
 ↪str_file_name_1, offset=0, cols_wanted=3)
df_2_PC_wHeatSinks = create_dictionary_from_csv(filename="./data_files/" +␣
 ↪str_file_name_2, offset=0, cols_wanted=3)
df_3_PC_wHeatSinksAndFan5V = create_dictionary_from_csv(filename="./data_files/" +␣
 ↪str_file_name_3, offset=0, cols_wanted=3)
df_4_PC_wHeatSinksAndFan3V = create_dictionary_from_csv(filename="./data_files/" +␣
 ↪str_file_name_4, offset=0, cols_wanted=3)
df_5_PC_wHeatSinksAndFan5Vrev = create_dictionary_from_csv(filename="./data_files/"␣
 ↪+ str_file_name_5, offset=0, cols_wanted=3)
df_6_RC_wHeatSinks = create_dictionary_from_csv(filename="./data_files/" +␣
 ↪str_file_name_6, offset=0, cols_wanted=3)
df_7_PC_wHeatSinksAndNoctuaFan5V = create_dictionary_from_csv(filename="./
 ↪data_files/" + str_file_name_7, offset=0, cols_wanted=3)
df_8_PC_wHeatSinksAndNoctuaFan3V = create_dictionary_from_csv(filename="./
 ↪data_files/" + str_file_name_8, offset=0, cols_wanted=3)

df_9_PC_wHeatSinks = create_dictionary_from_csv(filename="./data_files/" +␣
 ↪str_file_name_9, offset=0, cols_wanted=3)
```

[4]:
```
#df_1_PC_woHeatSinks.head(6)
#df_2_PC_wHeatSinks.head(6)
#df_3_PC_wHeatSinksAndFan5V.head(6)
#df_4_PC_wHeatSinksAndFan3V.head(6)
#df_5_PC_wHeatSinksAndFan5Vrev.head(6)
#df_6_RC_wHeatSinks.head(6)
#df_7_PC_wHeatSinksAndNoctuaFan5V.head(6)
```

```
#df_8_PC_wHeatSinksAndNoctuaFan3V.head(6)

df_9_PC_wHeatSinks.head(6)
```

[4]:

	Time	CPU Temperature	CPU Frequency
0	0.0	56.9	1000.0
1	2.1	56.9	700.0
2	4.2	56.9	700.0
3	6.3	56.9	700.0
4	8.4	57.5	800.0
5	10.5	56.4	800.0

[5]:

```
#df_1_PC_woHeatSinks.dtypes
#df_2_PC_wHeatSinks.dtypes
#df_3_PC_wHeatSinksAndFan5V.dtypes
#df_4_PC_wHeatSinksAndFan3V.dtypes
#df_5_PC_wHeatSinksAndFan5Vrev.dtypes
#df_6_RC_wHeatSinks.dtypes
#df_7_PC_wHeatSinksAndNoctuaFan5V.dtypes
#df_8_PC_wHeatSinksAndNoctuaFan3V.dtypes

df_9_PC_wHeatSinks.dtypes
```

[5]:

```
Time               float64
CPU Temperature    float64
CPU Frequency      float64
dtype: object
```

5.2 Smoothing with a moving average filter

[6]:

```
# Smooth temperature column only!
df_1_PC_woHeatSinks['CPU Temperature'] = df_1_PC_woHeatSinks['CPU Temperature'].
 ↪rolling(window=3).mean()
df_2_PC_wHeatSinks['CPU Temperature'] = df_2_PC_wHeatSinks['CPU Temperature'].
 ↪rolling(window=3).mean()
df_3_PC_wHeatSinksAndFan5V['CPU Temperature'] = df_3_PC_wHeatSinksAndFan5V['CPU␣
 ↪Temperature'].rolling(window=3).mean()
df_4_PC_wHeatSinksAndFan3V['CPU Temperature'] = df_4_PC_wHeatSinksAndFan3V['CPU␣
 ↪Temperature'].rolling(window=3).mean()
df_5_PC_wHeatSinksAndFan5Vrev['CPU Temperature'] =␣
 ↪df_5_PC_wHeatSinksAndFan5Vrev['CPU Temperature'].rolling(window=3).mean()
df_6_RC_wHeatSinks['CPU Temperature'] = df_6_RC_wHeatSinks['CPU Temperature'].
 ↪rolling(window=3).mean()
df_7_PC_wHeatSinksAndNoctuaFan5V['CPU Temperature'] =␣
 ↪df_7_PC_wHeatSinksAndNoctuaFan5V['CPU Temperature'].rolling(window=3).mean()
df_8_PC_wHeatSinksAndNoctuaFan3V['CPU Temperature'] =␣
 ↪df_8_PC_wHeatSinksAndNoctuaFan3V['CPU Temperature'].rolling(window=3).mean()

df_9_PC_wHeatSinks['CPU Temperature'] = df_9_PC_wHeatSinks['CPU Temperature'].
 ↪rolling(window=3).mean()
```

[7]:

```
df_9_PC_wHeatSinks
```

[7]:

	Time	CPU Temperature	CPU Frequency
0	0.0	NaN	1000.0

```
1       2.1            NaN        700.0
2       4.2      56.900000        700.0
3       6.3      56.900000        700.0
4       8.4      57.100000        800.0
..      ...            ...          ...
701  1491.6      64.300000        800.0
702  1493.7      64.100000        900.0
703  1495.8      64.300000       1200.0
704  1497.9      64.300000        600.0
705  1500.0      64.133333        600.0

[706 rows x 3 columns]
```

5.3 Display / Plot data from dataframes

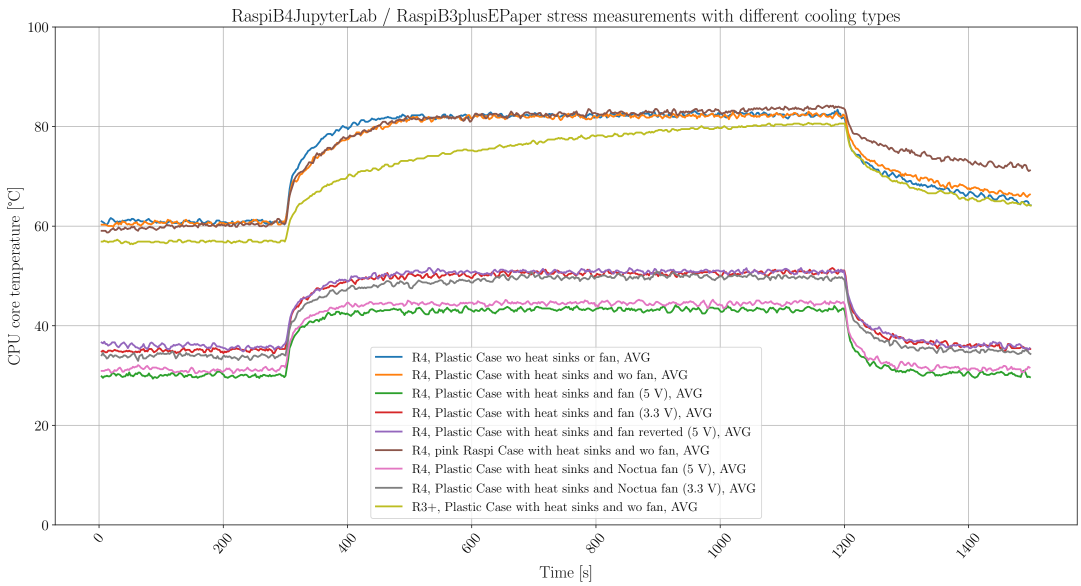
5.3.1 Comparative representation of the temperature curves

This is a comparative representation of the temperature curves over all examined cooling variants for Raspberry Pi B4 and 3B+.

[27]:

```
# figsize: a tuple (width, height) in inches
plt.figure(num=0, figsize=(20, 10), dpi=80, facecolor='w', edgecolor='k')
axes = plt.gca()

#xfmt = mdates.DateFormatter('%H:%M:%S')
#axes.xaxis.set_major_formatter(xfmt)

plt.title('RaspiB4JupyterLab / RaspiB3plusEPaper stress measurements with different␣
 ↪cooling types')

#plt.plot(df_1_PC_woHeatSinks['Time'], df_1_PC_woHeatSinks['CPU Temperature'], '-',␣
 ↪label='Plastic Case wo heat sinks or fan')
#plt.plot(df_2_PC_wHeatSinks['Time'], df_2_PC_wHeatSinks['CPU Temperature'], '-',␣
 ↪label='Plastic Case with heat sinks and wo fan')
#plt.plot(df_3_PC_wHeatSinksAndFan5V['Time'], df_3_PC_wHeatSinksAndFan5V['CPU␣
 ↪Temperature'], '-', label='Plastic Case with heat sinks and fan (5 V)')
#plt.plot(df_4_PC_wHeatSinksAndFan3V['Time'], df_4_PC_wHeatSinksAndFan3V['CPU␣
 ↪Temperature'], '-', label='Plastic Case with heat sinks and fan (3.3 V)')
#plt.plot(df_5_PC_wHeatSinksAndFan5Vrev['Time'], df_5_PC_wHeatSinksAndFan5Vrev['CPU␣
 ↪Temperature'], '-', label='Plastic Case with heat sinks and fan reverted (5 V)')
#plt.plot(df_6_RC_wHeatSinks['Time'], df_6_RC_wHeatSinks['CPU Temperature'], '-',␣
 ↪label='pink Raspi Case with heat sinks and and wo fan')

plt.plot(df_1_PC_woHeatSinks['Time'], df_1_PC_woHeatSinks['CPU Temperature'], '-',␣
 ↪label='R4, Plastic Case wo heat sinks or fan, AVG')
plt.plot(df_2_PC_wHeatSinks['Time'], df_2_PC_wHeatSinks['CPU Temperature'], '-',␣
 ↪label='R4, Plastic Case with heat sinks and wo fan, AVG')
plt.plot(df_3_PC_wHeatSinksAndFan5V['Time'], df_3_PC_wHeatSinksAndFan5V['CPU␣
 ↪Temperature'], '-', label='R4, Plastic Case with heat sinks and fan (5 V), AVG')
plt.plot(df_4_PC_wHeatSinksAndFan3V['Time'], df_4_PC_wHeatSinksAndFan3V['CPU␣
 ↪Temperature'], '-', label='R4, Plastic Case with heat sinks and fan (3.3 V), AVG')
plt.plot(df_5_PC_wHeatSinksAndFan5Vrev['Time'], df_5_PC_wHeatSinksAndFan5Vrev['CPU␣
 ↪Temperature'], '-', label='R4, Plastic Case with heat sinks and fan reverted (5␣
 ↪V), AVG')
plt.plot(df_6_RC_wHeatSinks['Time'], df_6_RC_wHeatSinks['CPU Temperature'], '-',␣
 ↪label='R4, pink Raspi Case with heat sinks and wo fan, AVG')
```

```
plt.plot(df_7_PC_wHeatSinksAndNoctuaFan5V['Time'],␣
 ↪df_7_PC_wHeatSinksAndNoctuaFan5V['CPU Temperature'], '-', label='R4, Plastic Case␣
 ↪with heat sinks and Noctua fan (5 V), AVG')
plt.plot(df_8_PC_wHeatSinksAndNoctuaFan3V['Time'],␣
 ↪df_8_PC_wHeatSinksAndNoctuaFan3V['CPU Temperature'], '-', label='R4, Plastic Case␣
 ↪with heat sinks and Noctua fan (3.3 V), AVG')

plt.plot(df_9_PC_wHeatSinks['Time'], df_9_PC_wHeatSinks['CPU Temperature'], '-',␣
 ↪label='R3+, Plastic Case with heat sinks and wo fan, AVG')

plt.xlabel('Time [s]')
plt.ylabel('CPU core temperature [°C]')

plt.ylim(0, 100)

plt.grid(True)

plt.setp(plt.gca().xaxis.get_majorticklabels(), 'rotation', 50)

plt.legend()

# Save plot to PNG and PDF
str_image_name = 'RaspiB4JupyterLab_stress_measurement'
plt.savefig(r'./data_files/' + str_image_name + '.png')
plt.savefig(r'./data_files/' + str_image_name + '.pdf')

plt.show()
```

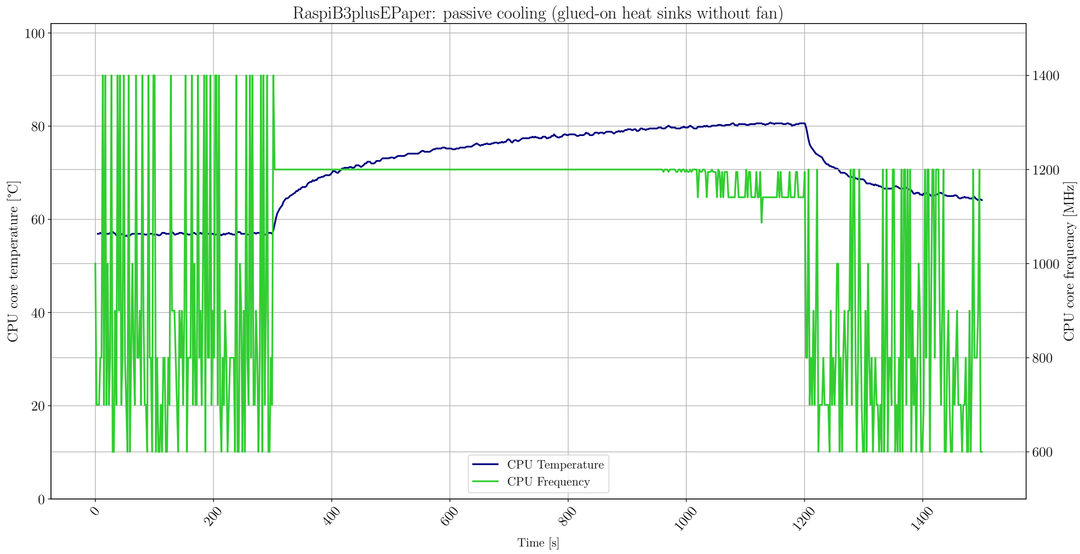


5.3.2 RaspiB3plusEPaper: Temperature curve compared with the curve of the CPU frequency (passive cooling)

This is the plot of the temperature curve compared with the CPU frequency curve for the Raspberry Pi 3B+ with the cooling variant “glued-on heat sinks without fan”.

[26]:

```python
# Figsize: a tuple (width, height) in inches
# Create figure and axis objects with subplots()
fig, ax1 = plt.subplots(num=0, figsize=(20, 10), dpi=80, facecolor='w',␣
↪edgecolor='k')

axes = plt.gca()

plt.title('RaspiB3plusEPaper: passive cooling (glued-on heat sinks without fan)')

line1 = ax1.plot(df_9_PC_wHeatSinks['Time'], df_9_PC_wHeatSinks['CPU Temperature'],␣
↪color='navy', label='CPU Temperature')

# Set x-axis label
ax1.set_xlabel('Time [s]', fontsize=14)
# Set y-axis label
ax1.set_ylabel('CPU core temperature [°C]', fontsize=16)
ax1.set_ylim(0, 102)
ax1.grid(True)
plt.xticks(rotation=50)

# Twin object for two different y-axis on the same plot
ax2 = ax1.twinx()

line2 = ax2.plot(df_9_PC_wHeatSinks['Time'], df_9_PC_wHeatSinks['CPU Frequency'],␣
↪color='limegreen', label='CPU Frequency')

# Set y-axis label
ax2.set_ylabel('CPU core frequency [MHz]', fontsize=16)
ax2.set_ylim(500, 1510)
ax2.grid(True)

# Add all lines to the same legend box
lines_all = line1+line2
labels = [l.get_label() for l in lines_all]
ax1.legend(lines_all, labels, loc='lower center')

# Save plot to PNG and PDF
str_image_name = 'RaspiB4JupyterLab_stress_measurement'
plt.savefig(r'./data_files/' + str_image_name + '.png')
plt.savefig(r'./data_files/' + str_image_name + '.pdf')

plt.show()
```

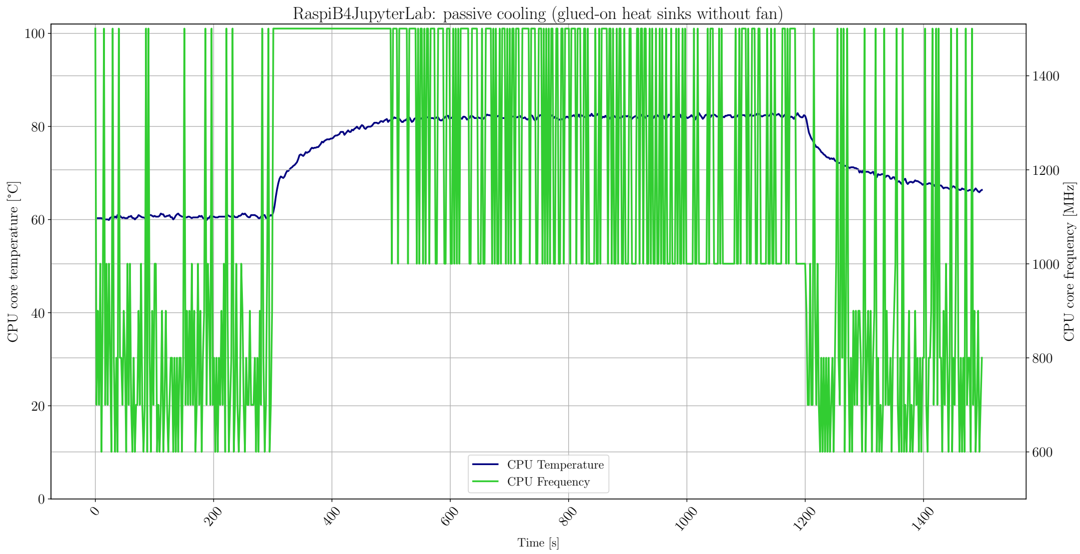


5.3.3 RaspiB4JupyterLab: Temperature curve compared with the curve of the CPU frequency (passive cooling)

This is the plot of the temperature curve compared with the CPU frequency curve for the Raspberry Pi B4 with the cooling variant "glued-on heat sinks without fan".

[25]:
```
# Figsize: a tuple (width, height) in inches
# Create figure and axis objects with subplots()
fig, ax1 = plt.subplots(num=0, figsize=(20, 10), dpi=80, facecolor='w',␣
↪edgecolor='k')

axes = plt.gca()

plt.title('RaspiB4JupyterLab: passive cooling (glued-on heat sinks without fan)')

line1 = ax1.plot(df_2_PC_wHeatSinks['Time'], df_2_PC_wHeatSinks['CPU Temperature'],␣
↪color='navy', label='CPU Temperature')

# Set x-axis label
ax1.set_xlabel('Time [s]', fontsize=14)
# Set y-axis label
ax1.set_ylabel('CPU core temperature [°C]', fontsize=16)
ax1.set_ylim(0, 102)
ax1.grid(True)
plt.xticks(rotation=50)

# Twin object for two different y-axis on the same plot
ax2 = ax1.twinx()

line2 = ax2.plot(df_2_PC_wHeatSinks['Time'], df_2_PC_wHeatSinks['CPU Frequency'],␣
↪color='limegreen', label='CPU Frequency')

# Set y-axis label
ax2.set_ylabel('CPU core frequency [MHz]', fontsize=16)
ax2.set_ylim(500, 1510)
ax2.grid(True)
```

```
# Add all lines to the same legend box
lines_all = line1+line2
labels = [l.get_label() for l in lines_all]
ax1.legend(lines_all, labels, loc='lower center')

# Save plot to PNG and PDF
#str_image_name = 'RaspiB4JupyterLab_stress_measurement'
#plt.savefig(r'./data_files/' + str_image_name + '.png')
#plt.savefig(r'./data_files/' + str_image_name + '.pdf')

plt.show()
```

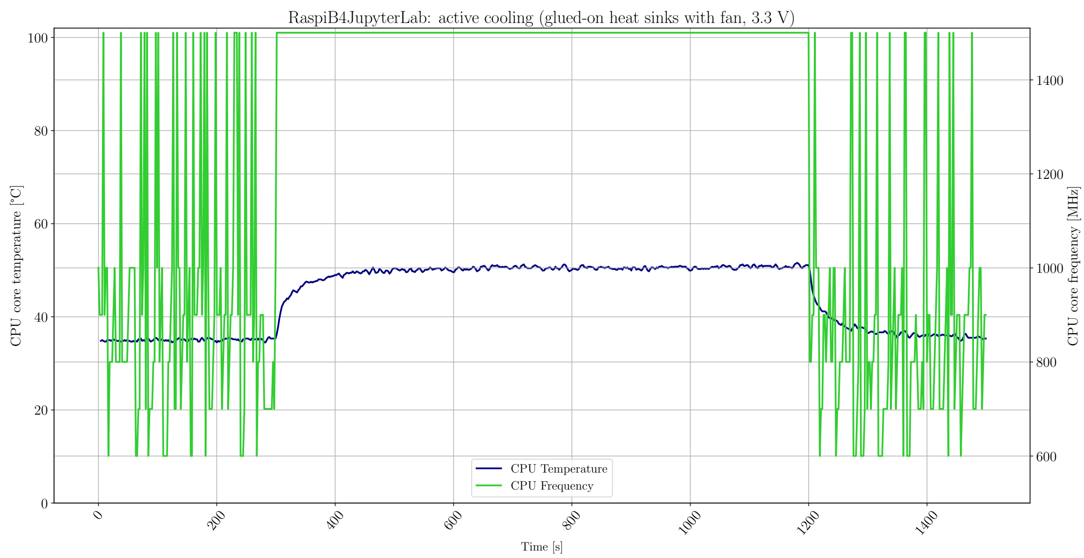


5.3.4 RaspiB4JupyterLab: Temperature curve compared with the curve of the CPU frequency (active cooling)

This is the plot of the temperature curve compared with the CPU frequency curve for the Raspberry Pi B4 with the cooling variant "glued-on heat sinks with fan (driven by 3.3 V)".

[24]:
```
# Figsize: a tuple (width, height) in inches
# Create figure and axis objects with subplots()
fig, ax1 = plt.subplots(num=0, figsize=(20, 10), dpi=80, facecolor='w',␣
 ↪edgecolor='k')

axes = plt.gca()

plt.title('RaspiB4JupyterLab: active cooling (glued-on heat sinks with fan, 3.3 V)')

line1 = ax1.plot(df_4_PC_wHeatSinksAndFan3V['Time'],␣
 ↪df_4_PC_wHeatSinksAndFan3V['CPU Temperature'], color='navy', label='CPU␣
 ↪Temperature')
#plt.plot(df_4_PC_wHeatSinksAndFan3V['Time'], df_4_PC_wHeatSinksAndFan3V['CPU␣
 ↪Temperature'], '-', label='R4, Plastic Case with heat sinks and fan (3.3 V),␣
 ↪AVG')

# Set x-axis label
```

```python
ax1.set_xlabel('Time [s]', fontsize=14)
# Set y-axis label
ax1.set_ylabel('CPU core temperature [°C]', fontsize=16)
ax1.set_ylim(0, 102)
ax1.grid(True)
plt.xticks(rotation=50)

# Twin object for two different y-axis on the same plot
ax2 = ax1.twinx()

line2 = ax2.plot(df_4_PC_wHeatSinksAndFan3V['Time'],␣
↪df_4_PC_wHeatSinksAndFan3V['CPU Frequency'], color='limegreen', label='CPU␣
↪Frequency')

# Set y-axis label
ax2.set_ylabel('CPU core frequency [MHz]', fontsize=16)
ax2.set_ylim(500, 1510)
ax2.grid(True)

# Add all lines to the same legend box
lines_all = line1+line2
labels = [l.get_label() for l in lines_all]
ax1.legend(lines_all, labels, loc='lower center')

# Save plot to PNG and PDF
#str_image_name = 'RaspiB4JupyterLab_stress_measurement'
#plt.savefig(r'./data_files/' + str_image_name + '.png')
#plt.savefig(r'./data_files/' + str_image_name + '.pdf')

plt.show()
```



[ ]: