**Synchronome**

Rishab Shah

Final Project Report

ECEN 5623 Real Time Embedded System

Aug 10, 2021

1 INTRODUCTION 2

2 FUNCTIONAL REQUIREMENTS 3

3 REAL-TIME REQUIREMENTS 4

4 FUNCTIONAL DESIGN OVERVIEW and DIAGRAMS 6

4.1 System Level Block Diagram: 6

4.2 Software Architecture Diagram for services: 7

5 REAL-TIME ANALYSIS, DESIGN AND TIMING DIAGRAMS 9

5.1 1 Hz Analysis 9

5.1.1 Core 1 Analysis 9

5.1.2 Core 2 Analysis 10

5.2 10 Hz Analysis 12

5.2.1 Core 1 Analysis 12

5.2.2 Core 2 Analysis 13

6 PROOF OF CONCEPT WITH EXAMPLES AND OUTPUT 15

6.1 1 Hz test for 1800 frames (Negative Transformation included) 15

6.2 10 Hz test for 1800 frames (Negative Transformation included) 19

6.3 1 Hz test run time transformation change 23

7 CONCLUSION 25

8 FORMAL REFERENCES 26

9 APPENDICES 26

9.1 CODE BULD AND RUN 26

# INTRODUCTION

Inspiration:

The Shortt–Synchronome free pendulum clock was a complex precision [electromechanical](https://en.wikipedia.org/wiki/Electromechanical) [pendulum clock](https://en.wikipedia.org/wiki/Pendulum_clock) invented in 1921 by British railway engineer [William Hamilton Shortt](https://en.wikipedia.org/wiki/William_Hamilton_Shortt) in collaboration with [horologist](https://en.wikipedia.org/wiki/Horologist) [Frank Hope-Jones](https://en.wikipedia.org/wiki/Frank_Hope-Jones), and manufactured by the Synchronome Co., Ltd. Of [London](https://en.wikipedia.org/wiki/London), UK. They were the most accurate pendulum clocks ever commercially produced, and became the highest standard for timekeeping between the 1920s and the 1940s, after which mechanical clocks were superseded by [quartz](https://en.wikipedia.org/wiki/Quartz_clock) time standards. They were used worldwide in [astronomical observatories](https://en.wikipedia.org/wiki/Astronomical_observatory), [naval observatories](https://en.wikipedia.org/wiki/Naval_observatory), in scientific research, and as a [primary standard](https://en.wikipedia.org/wiki/Primary_standard) for national [time dissemination services](https://en.wikipedia.org/wiki/Time_signal). The Shortt was the first clock to be a more accurate timekeeper than the Earth itself; it was used in 1926 to detect tiny seasonal changes in the Earth's rotation rate. Shortt clocks achieved accuracy of around a second per year, although [a recent measurement indicated](https://en.wikipedia.org/wiki/Shortt%E2%80%93Synchronome_clock#Recent_accuracy_measurement) they were even more accurate. About 100 were produced between 1922 and 1956.

Shortt clocks kept time with two [pendulums](https://en.wikipedia.org/wiki/Pendulum), a primary pendulum swinging in a vacuum tank and a secondary pendulum in a separate clock, which was synchronized to the primary by an electric circuit and [electromagnets](https://en.wikipedia.org/wiki/Electromagnet). The secondary pendulum was attached to the timekeeping mechanisms of the clock, leaving the primary pendulum virtually free of external disturbances. [1]

***i.e to synchronize events.***

***In the project I am trying to achieve to synchronize the image capturing of the camera with the second hand tick of the clock for 1 Hz and capturing the images at every 10 Hz for the stopwatch***

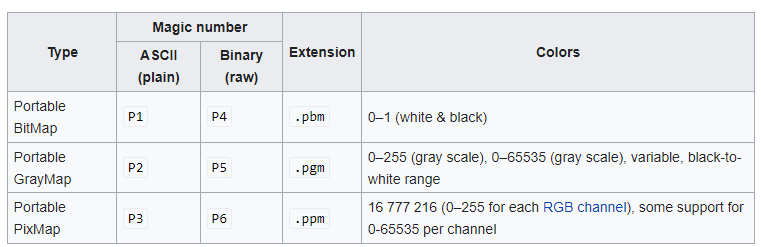
# FUNCTIONAL REQUIREMENTS

**Objective:**

An external clock – Physical or <https://www.visnos.com/demos/clock> would be used to acquire synchronized images where all images would show a unique state of hour, minute and second hands on an ANALOG\_CLOCK using a resolution of VGA 640x840. Based on this resolution, accurate time-lapse images will be encoded. An effort to achieve 1 Hz verification (no skips, blurs, or repeats) with an ANALOG CLOCK is to be completed and full-credit STRETCH goal for 10 Hz verification with a DIGITAL STOPWATCH over 1801 frames (30 minutes for 1 Hz, 3 minute for 10 Hz).

**Detailed list of requirements:**

1. Resolution used will be VGA (640x480)
2. The frames will be stored in the ppm or pgm format and will be used for verification by acquiring individual frames from the camera. The frames will be acquired at 1Hz rate.
3. The camera and interface will acquire frames at 1 Hz.
4. Verification of image file format will be PPM "P3" or "P6" RGB or it should be "P2" or "P5" for Graymap format as documented for [Netpbm format](http://en.wikipedia.org/wiki/Netpbm_format)



1. Your frame acquisition and time-stamping should be accurate so that there is no "observable" errors in 1801 frames (or 1800 seconds of continuous operation over a wall-clock time period of 30 minutes and 0 seconds).
2. The time-stamps must be embedded in the images in the PPM header of EVERY FRAME as a comment field (using "#" prefaced line in your header).
3. The HOST or TARGET platform "uname -a" output must be embedded the PPM header of EVERY FRAME as a comment field (using "#" prefaced line in your header) for identification of the system on which it was run.
4. The 1 Hz verification (no skips, blurs, or repeats) with an ANALOG CLOCK is for minimum goal and the full-credit STRETCH goal is with 10 Hz verification with a DIGITAL STOPWATCH over 1801 frames (30 minutes for 1 Hz, 3 minute for 10 Hz).

In the design, multiple RT services are on one core and the write back service is a best effort service and is on a separate core so that it does not interfere with the working of the RT services.

# REAL-TIME REQUIREMENTS

The project has two types of services categorized as Real- time and Non-Real time:

1. Real-Time functional requirements/Services:

Sequencer: (PIT)

This function is triggered at a 100 Hz frequency by the kernel timer using a Signal and timer\_create combination. Here, we set the value of the trigger frequency by modifying it\_interval and it\_value parameters. This function is also used to disable all the RT services when an abort request is received.

Triggered by the O.S.

1. **Service\_0\_Sequencer**:

This Service controls the execution of all the real time services which are present on the different cores of the system. This thread executes at 100 HZ frequency and gives releases to other services at 1Hz, 10 Hz, and 0.5 Hz as per the count value.

This has the priority of RTMAX-1.

This service runs on core 1 for both 1 and 10 Hz.

This service is called at 100 Hz frequency.

1. **Service\_1\_frame\_acquisition**:

The purpose of this service is to acquire frames from the camera. It acquires frames at 10 Hz for 1HZ verification and 20 Hz for 10 HZ verification. This service involves driver calls made over USB to the UVC camera to get the frames.

This has the priority of RTMAX-2.

This service runs on core 1 for both 1 and 10 Hz.

This service is called at 10 Hz and 20 Hz frequency.

1. **Service\_X\_frame\_diff**:

The purpose of this service is to select the proper images to send it forward for processing YUYV to RGB or YUYV to Negative.

For 1 Hz, an oversampling mechanism is used with shotgun approach.

For 10 Hz, a selection logic is applied.

This has the priority of RTMAX-3.

This service runs on core 1 for 1 Hz and Core 2 for 10 Hz

This service is called at 10 Hz frequency for both the 1 Hz and 10 Hz verification.

1. **Service\_2\_frame\_process**:

The purpose of this service is to process the frames received by the Service\_X\_frame\_diff. It takes frames at 1 Hz frequency for 1 Hz verification and 10 Hz for 10 Hz verification. This service converts the previously selected YUYV image to RGB.

This has the priority of RTMAX-4.

This service runs on core 2 for both 1 and 10 Hz.

This service is called at 1 Hz frequency for 1 Hz, and 10 Hz for 10 Hz verification.

1. **Service\_3\_transformation\_process**:

This service converts the previously transformed RGB image to Negative image if the user has requested for a negative image to be stored. The output is RGB if the user had requested for a colored image either as a permanent service or a run-time change.

This has the priority of RTMAX-5.

This service runs on core 2 for both 1 and 10 Hz.

This service is called at 1 Hz frequency for 1 Hz, and 10 Hz for 10 Hz verification.

1. **Service\_4\_transformation\_on\_off**:

The purpose of this service is to make a switch between RGB and Negative images being dumped to the core. The frequency at which this service does not matter as it is blocked by a getchar (). But as a general understanding, in case the user wants to alternate the images continuously, one can possibly do so as the updating of the service happens before the frame\_process is called again.

This has the priority of RTMAX-6.

This service runs on core 2.

This service is called at 2 Hz frequency for both 1 and 10 Hz.

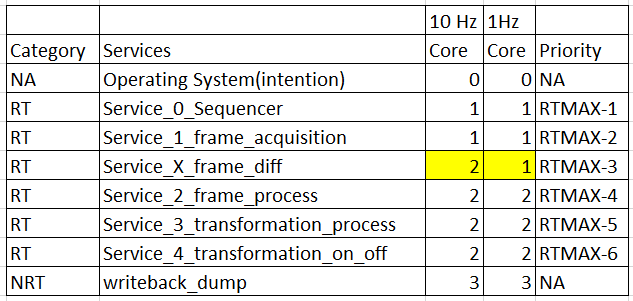
The communication between Service 1, Service X, Service 2, Service 3 and Writeback is done using message queues.

1. Non Real-time functional requirements/Services:
2. **Writeback\_dump**:

This service is used to dump the ppm images received from the Service\_3\_transformation\_process. This would run as a best-effort service. As the verification and analysis would be done post and not in real-time. It is not required to have a real-time priority. Since, dumping is the slowest operation of all the services, it is kept at a separate core as well as best-effort to not cause any problem to other real-time services running.

This service runs on core 3.

Summary of the services for 1 Hz and 10 Hz



# FUNCTIONAL DESIGN OVERVIEW and DIAGRAMS

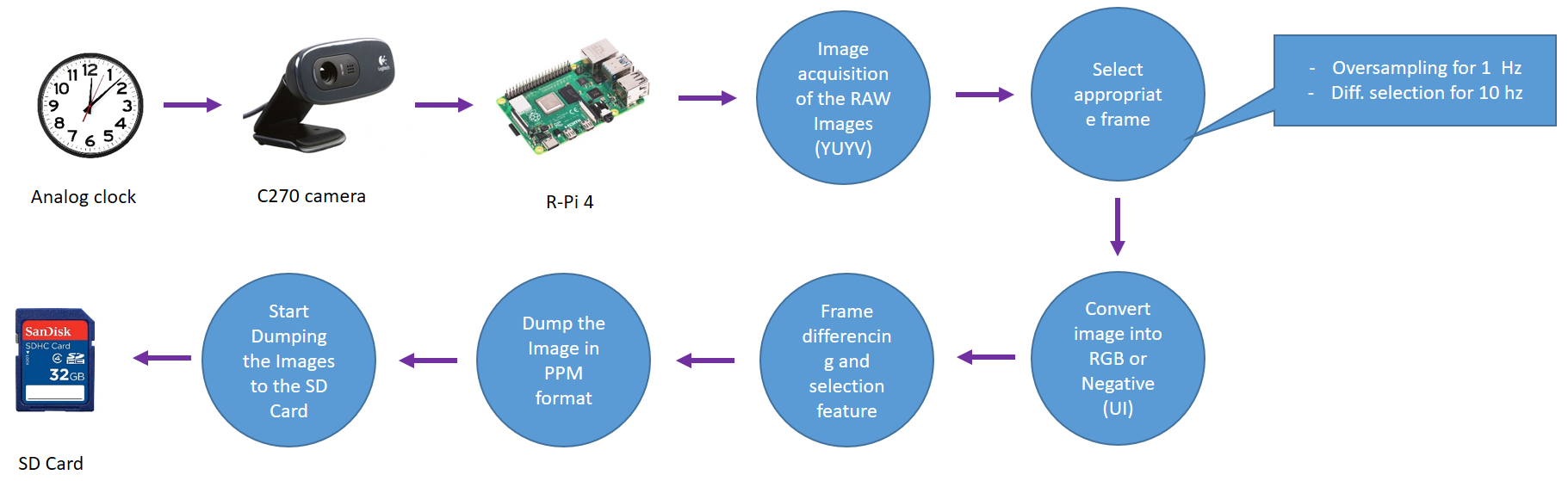
Following are the different diagrams used to explain the system

## System Level Block Diagram:

Following is the high level System Block Diagram of the end-to-end communication involved from getting the images to storing them.

Below diagram involves both the hardware and software components.

The main software components of the project (division steps) are shown as bubbles.

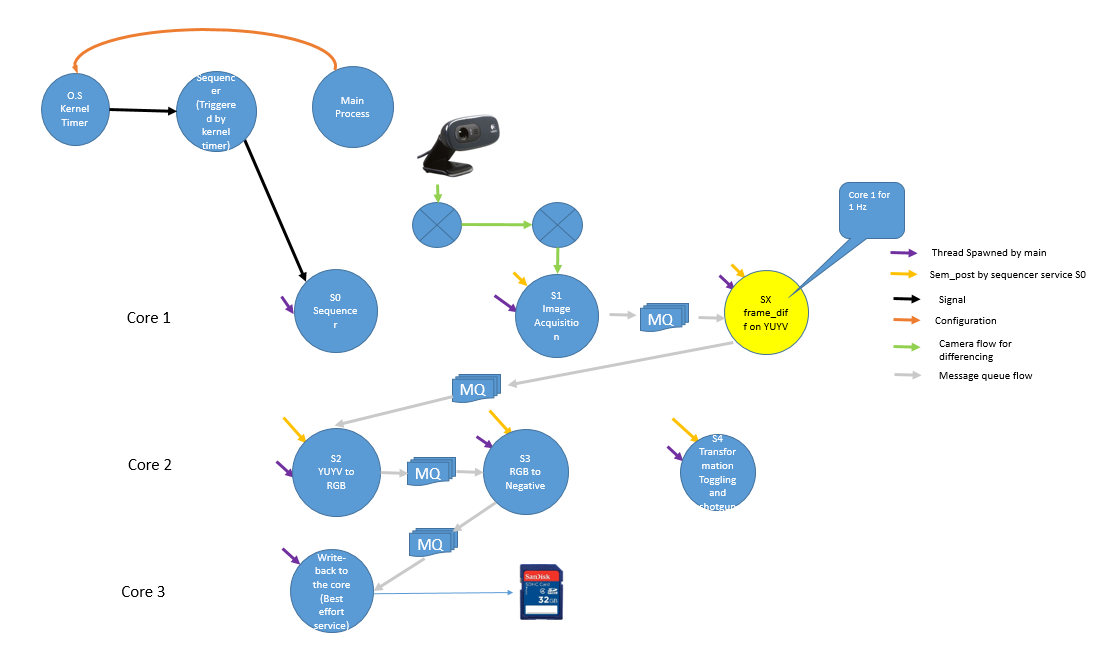


The camera in the above diagram captures the data from the analog clock/stop watch. This data of frames is transmitted to the R-Pi over the USB.

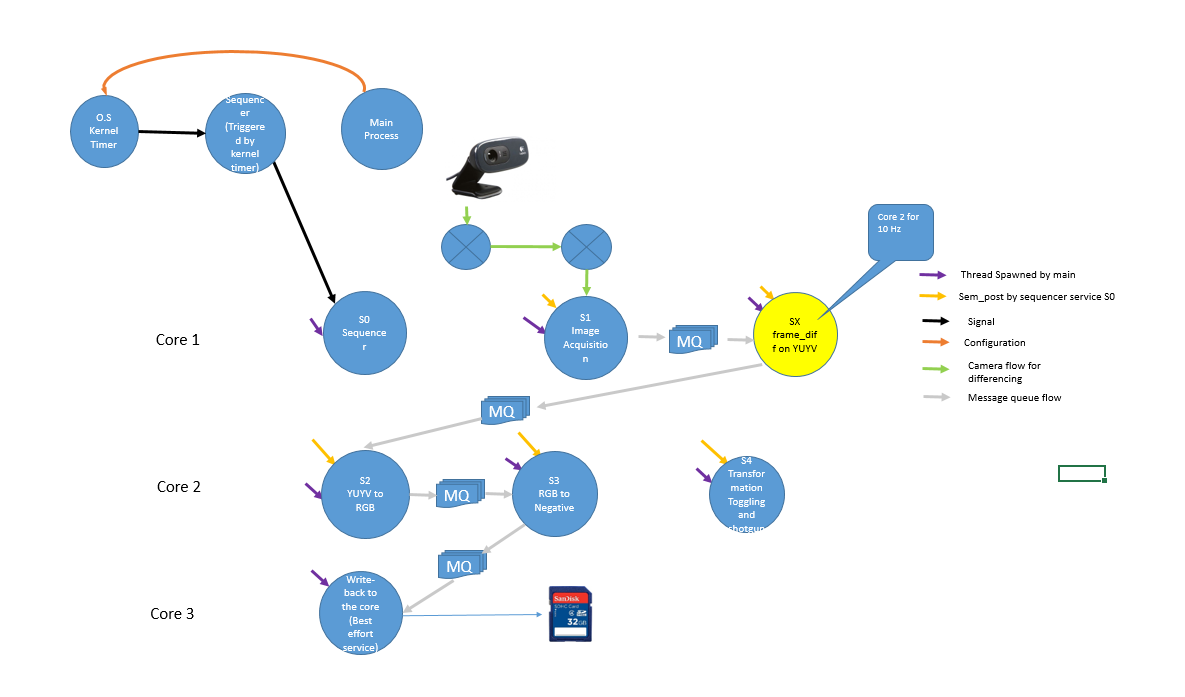
The images which are received are then processed (YUYV to RGB to Negative). These images are then stored on the SD card in PPM format.

## Software Architecture Diagram for services:

1 Hz



10 Hz



In the above diagram, appropriate labels are used to indicate between signals, configuration, thread spawning and sem\_post for release of services.

All RT services are SCHED\_FIFO.

**Core 0:**

Threads are not assigned to this core. It was left free for OS to schedule its services so that it does not interfere with our services by deferring to schedule any tasks on our selected cores.

**Core 1:**

Image acquisition and Image selection based upon YUYV image. Both are on same core to run in RMA such that Image acquisition can run on 20 Hz and Image selection on 10 Hz so as to properly extract the message queue data. The sequencer with highest priority is run on this core to schedule all other services.

**Note: Post demo, Image diff was used on core 2 for 10 Hz to get better results so that interference is reduced to the acquisition task. For 1 Hz, it was on core 1**

**Core 2:**

The Core 2 is used for Image processing operations (YUYV to RGB – S2) and (RGB to Negative – S3). The transformation on/off selection feature is also added here at 2 Hz. The diff is part of core 1 for 1 he and on core 2 for 10 Hz.

**Core 3:**

Writing to the SD card is a heavy operation. At 1 he it can be a real-time thread as per the jitter analysis but if it made to write at 10 Hz then it will impact deadline of other services as the time taken by writeback sometimes is in seconds.

Writeback is a heavy operation and hence, it was decided to put it as a Best effort service and on a separate core 3. This ensures that the deadline of other RT services is not impacted.

Message queues are used to communicate the image, timestamp and frame number over the services S1, SX, S2, S3 and Writeback. This provides elasticity in design.

# REAL-TIME ANALYSIS, DESIGN AND TIMING DIAGRAMS

## 1 Hz Analysis

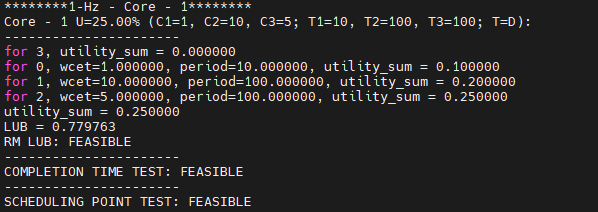
While finding the WCET, the time for CPU is considered after a data is dequeue and before the data is enqueue into the message queue. This time in between is the processing time which involves calling of function as well as processing. The value spent here would correspond to the CPU utilization time.

### Core 1 Analysis

Following three RT services were running on a single core - 1

1. Sequencer
2. Image Acquisition Service
3. Image Selection and Differencing Service

Scheduling Point Test and Completion Point Test for Core 1

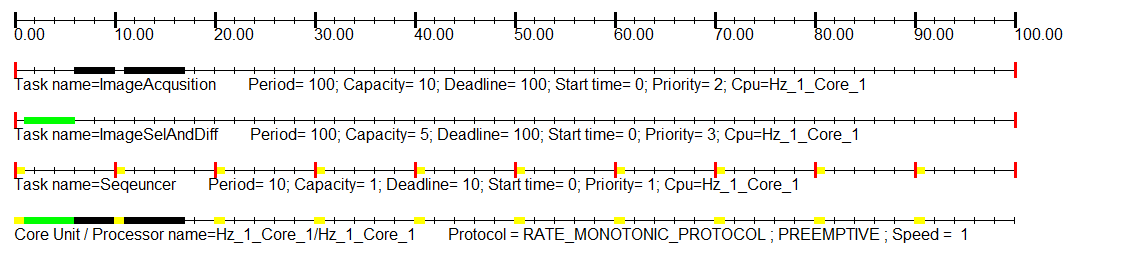


As Completion Point and Scheduling Point both are feasible, it shows that the given set of services can be scheduled by Rate Monotonic Theory. This is in concurrent with the output obtained for 1 Hz.

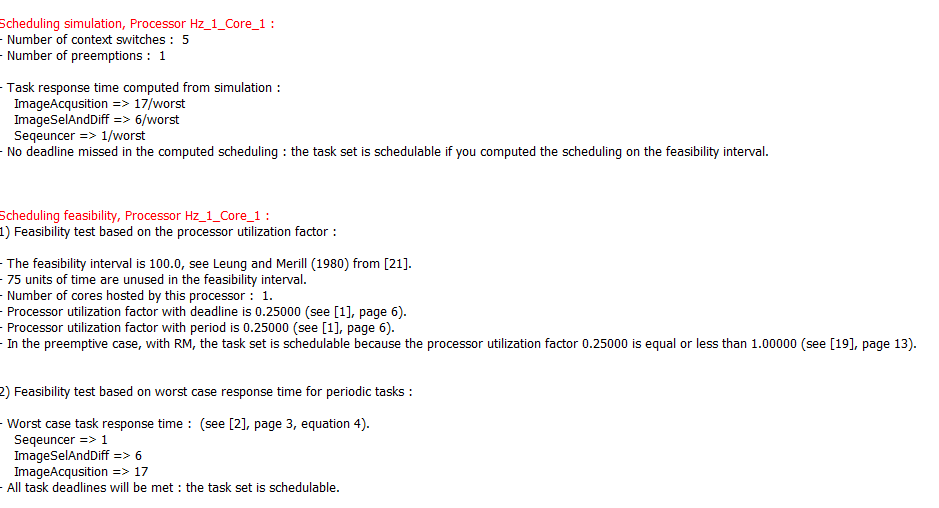
The safety margin in the current system is quite large: There is no need to worry regarding the task scheduled to fail the execution.

The margin is 1- 0.25 = 0.75 i.e. 75 percent for Core 1.

**Timing Diagram:**



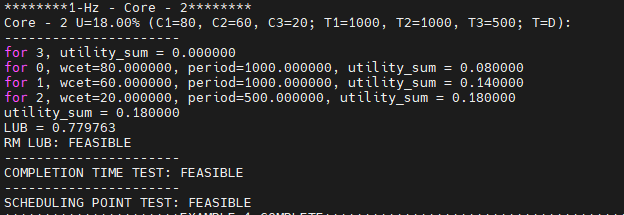
**Schedulability Analysis:**



### Core 2 Analysis

Following three RT services were running on a single core - 2

1. Image Processing Service: (YUYV to RGB)
2. Image Transformation Service: (RGB to Negative)
3. User Input Service:

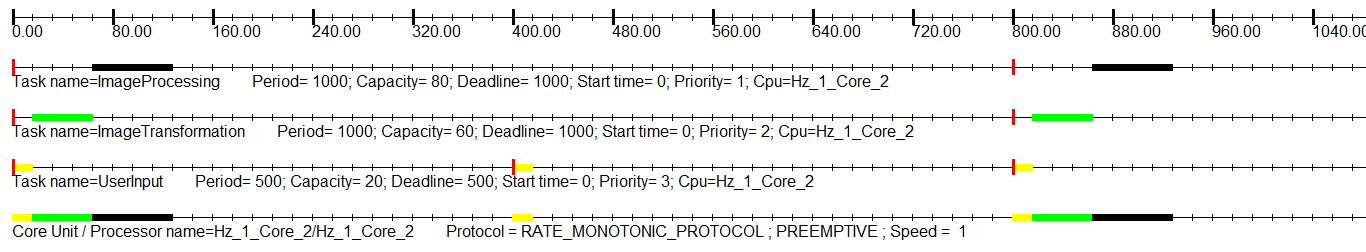


As Completion Point and Scheduling Point both are feasible, it shows that the given set of services can be scheduled by Rate Monotonic Theory. This is in concurrent with the output obtained for 1 Hz.

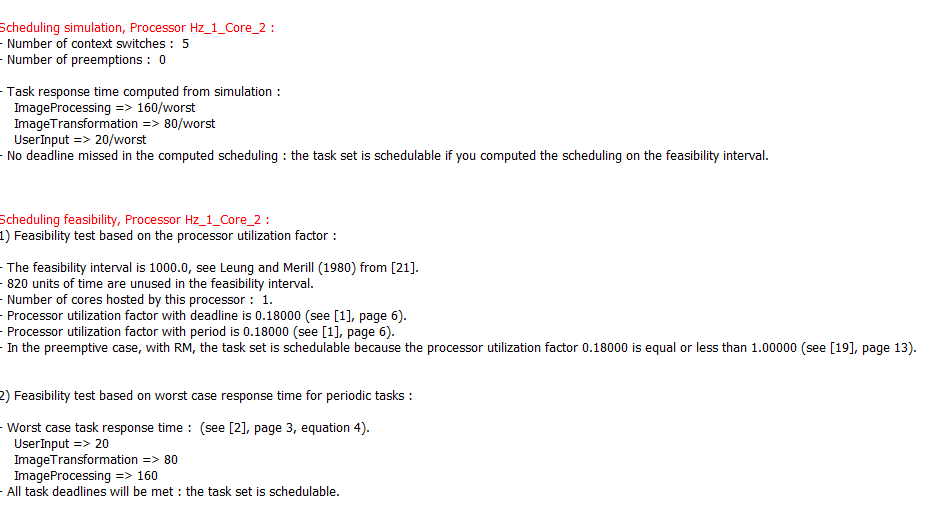
The safety margin in the current system is quite large: There is no need to worry regarding the task scheduled to fail the execution.

The margin is 1- 0.18 = 0.82 i.e. 82 percent for Core 2.

**Timing Diagram:**



**Schedulability Analysis:**



## 10 Hz Analysis

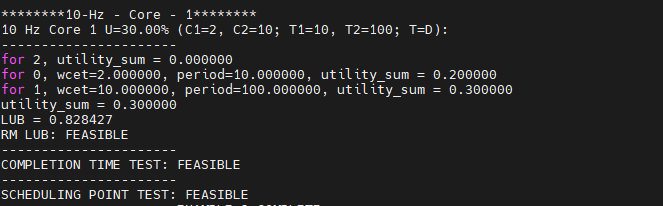
While finding the WCET, the time for CPU is considered after a data is dequeue and before the data is enqueue into the message queue. This time in between is the processing time which involves calling of function as well as processing. The value spent here would correspond to the CPU utilization time.

### Core 1 Analysis

Following two RT services were running on a single core - 1

1. Sequencer
2. Image Acquisition Service

**Scheduling and Completion Test**

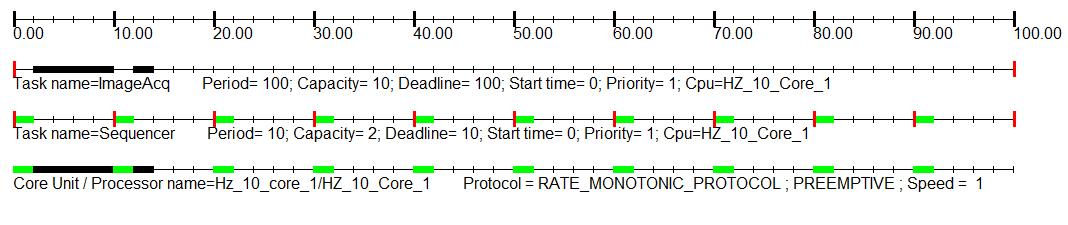


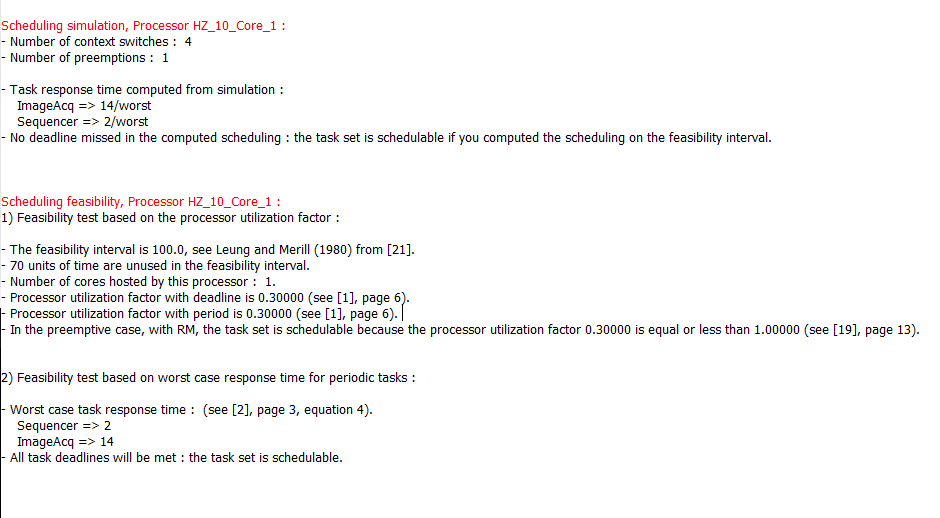
As Completion Point and Scheduling Point both are feasible, it shows that the given set of services can be scheduled by Rate Monotonic Theory.

The safety margin in the current system is quite large: There is no need to worry regarding the task scheduled to fail the execution.

The margin is 1- 0.30 = 0.70 i.e. 70 percent for Core 1

**Timing Diagram:**



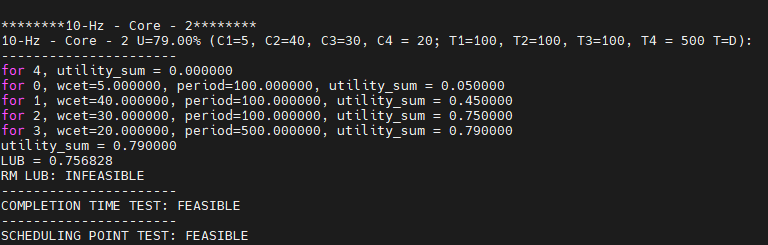


### Core 2 Analysis

Following three RT services were running on a single core - 2

1. Image Selection and Differencing Service
2. Image Processing Service (YUYV to RGB)
3. Image Transformation Service (RGB to Negative)
4. User Input Service

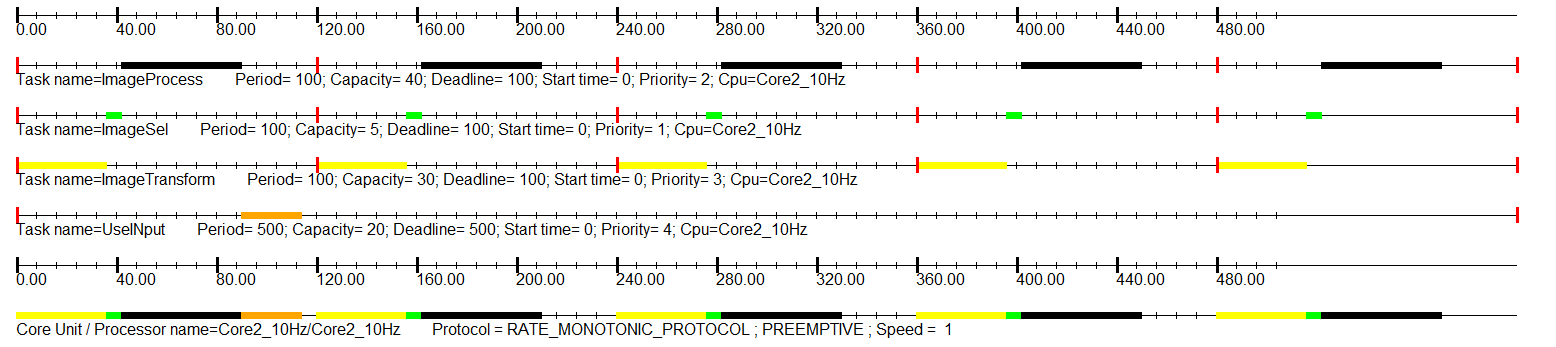
**Scheduling and Completion test**

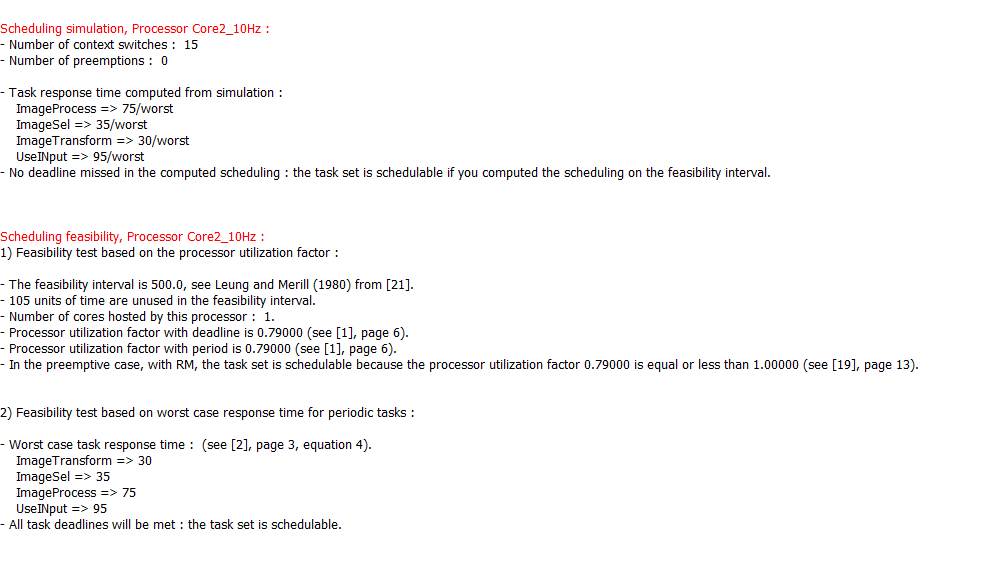


As Completion Point and Scheduling Point both are feasible, it shows that the given set of services can be scheduled not by LUB but these tests. It is better to rely on these tests and not on LUB value.

The margin is 1- 0.79= 0.21 i.e. 21 percent for Core 2

Timing Diagram:

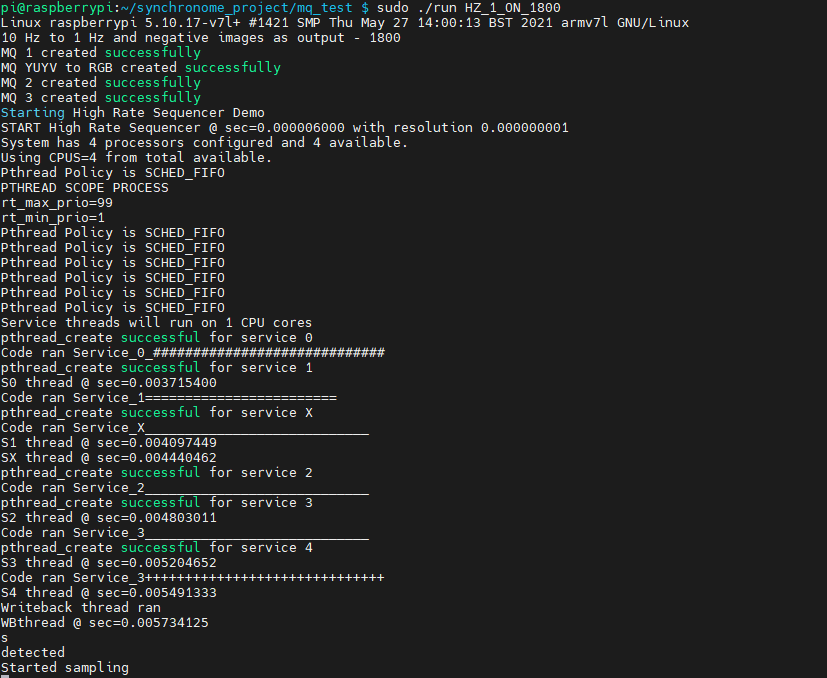




# PROOF OF CONCEPT WITH EXAMPLES AND OUTPUT

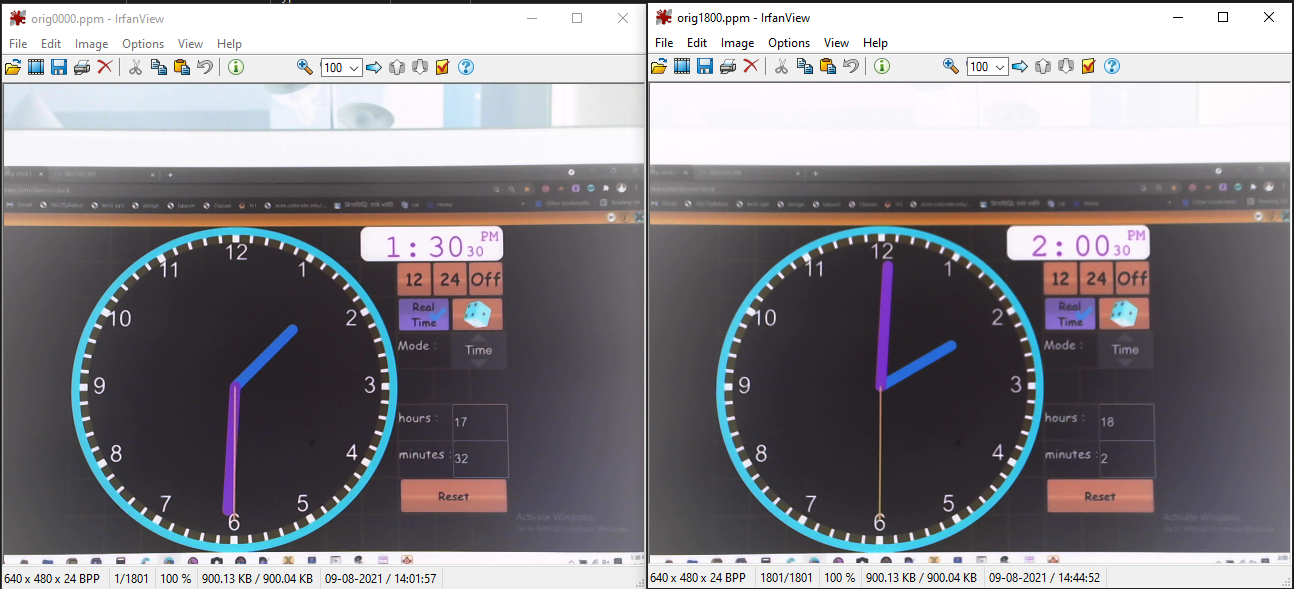
## 1 Hz test for 1800 frames (Negative Transformation included)

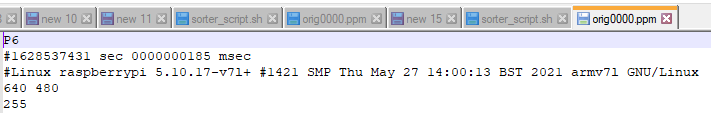
Following are the various outputs taken to demonstrate the working of the code:



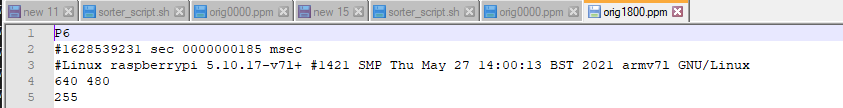
Following is the output when the system is run at 1Hz.

1. First frame and last frame. For Negative transformation the frames match.

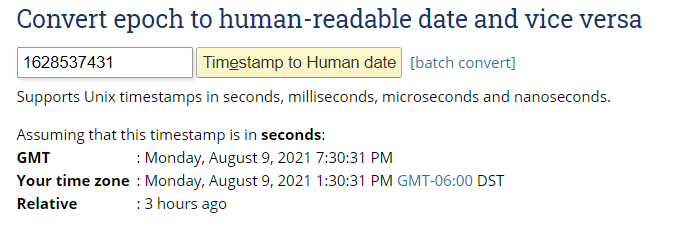


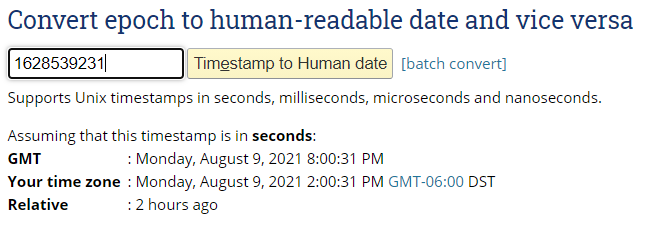
Header Information: Orig0000.ppm

Header Information: Orig1800.ppm



Time difference:





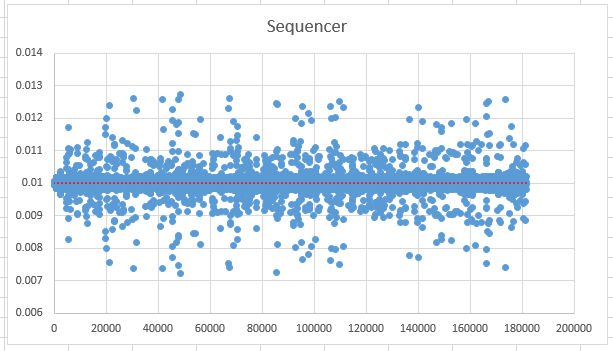
The time-stamps are registered properly.

Jitter Analysis: 

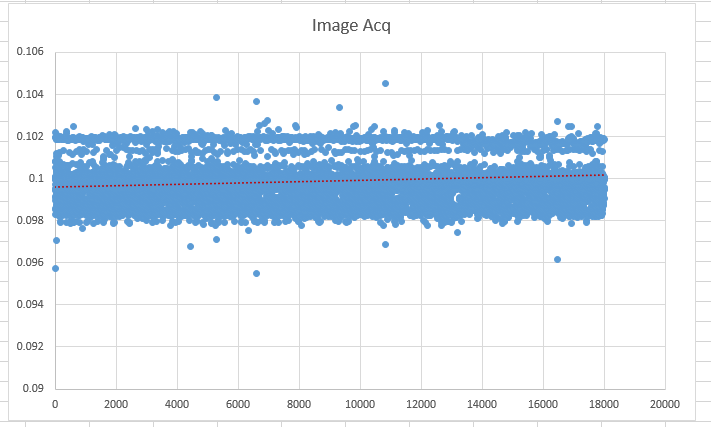
Following are the various plots:

* 1. Core 1:

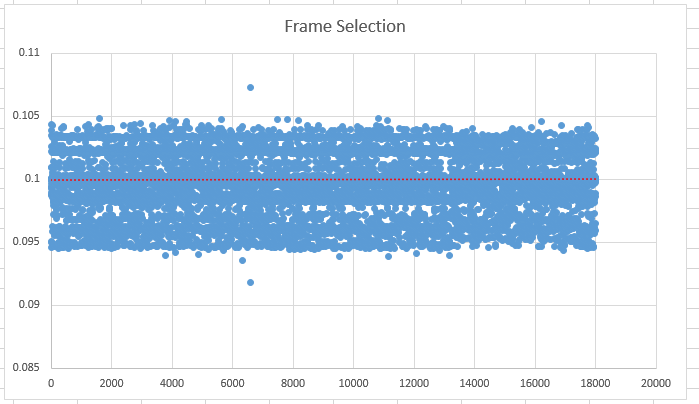
Service 0: Sequencer:



Service 1: Image acquisition:

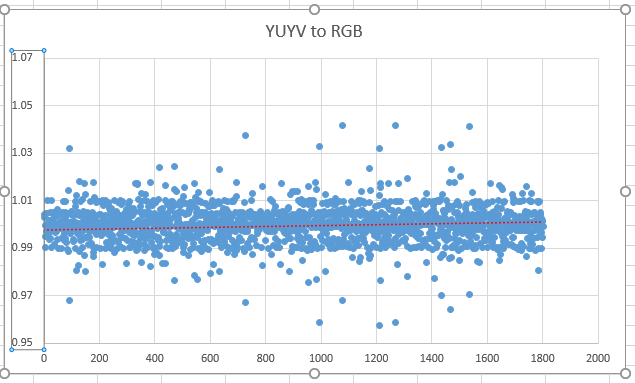


Service X: Frame Selection Service:

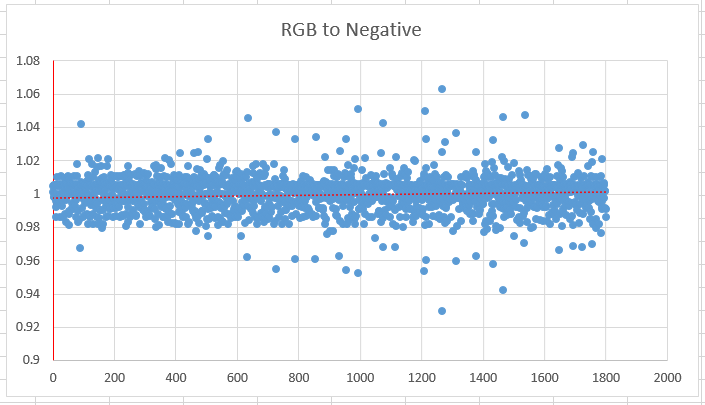


* 1. Core 2:

Service 2: YUYV to RGB

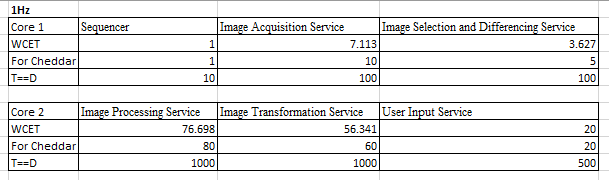


Service 3: RGB to Negative

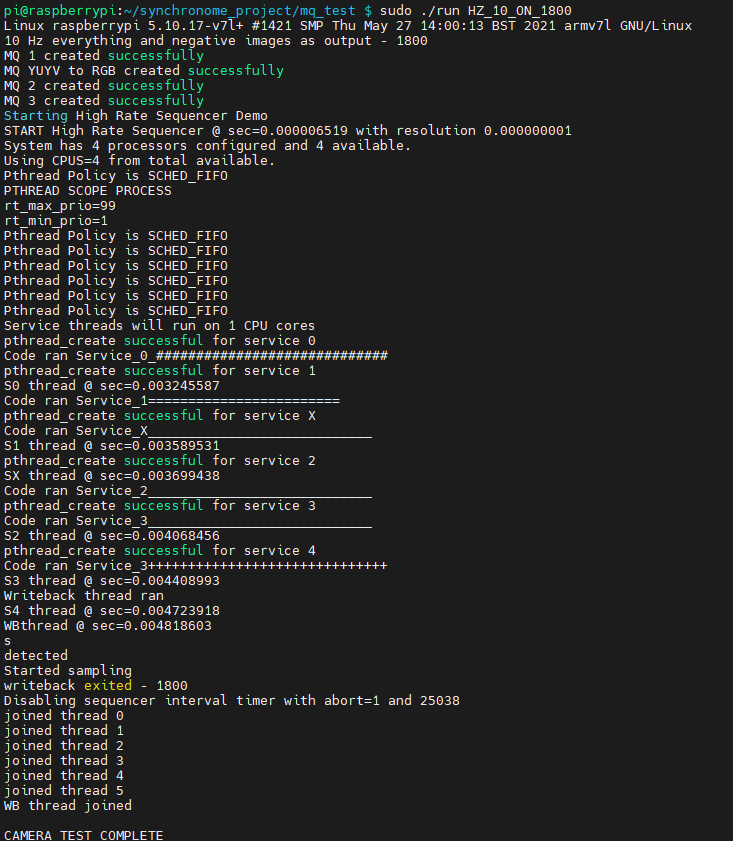


Another service is User input as Service 4, it is a getchar () service which remains blocked and does not play any role.

Following is the WCET analysis arrived practically:

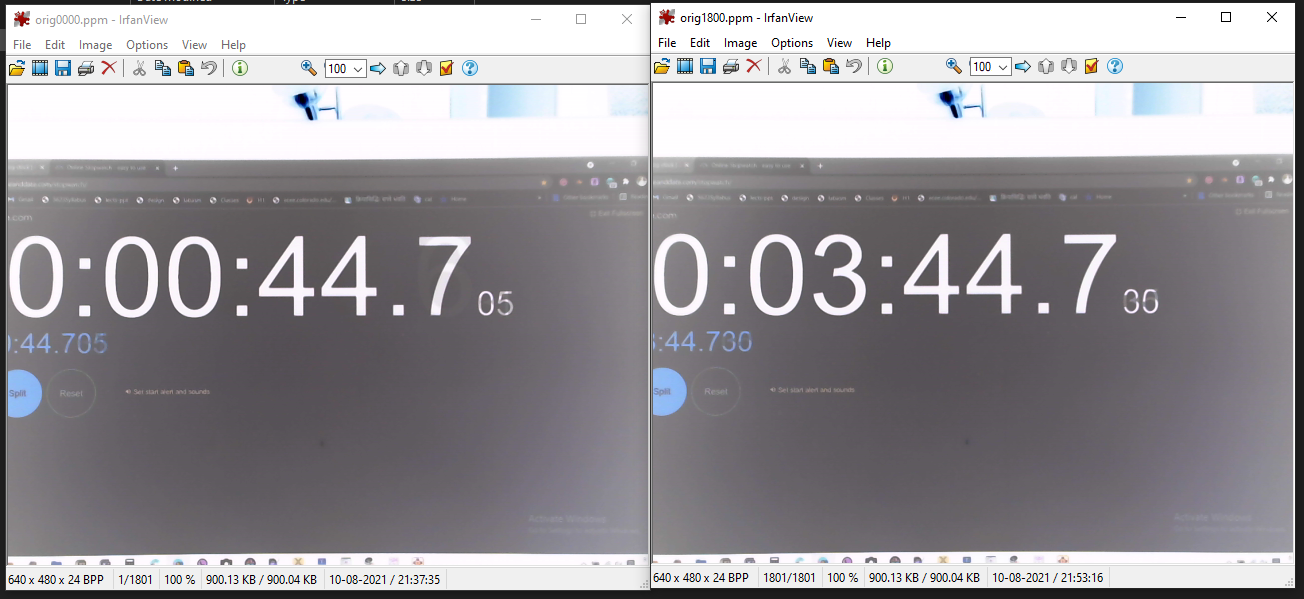


## 10 Hz test for 1800 frames (Negative Transformation included)

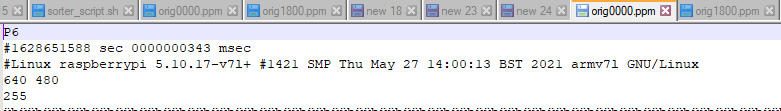


Following is the output when the system is run at 1Hz.

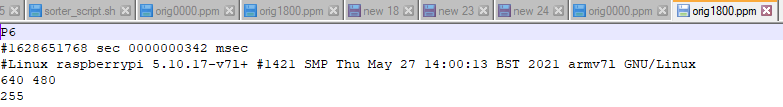
1. First frame and last frame. For Negative transformation the frames match.



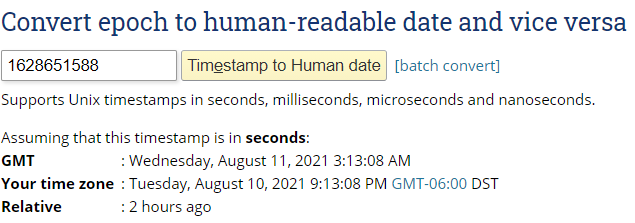
Header Information Orig0000.ppm

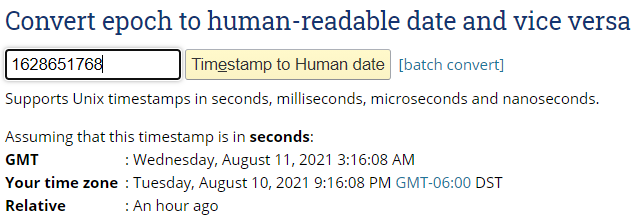


Header Information: Orig1800.ppm



Time difference:





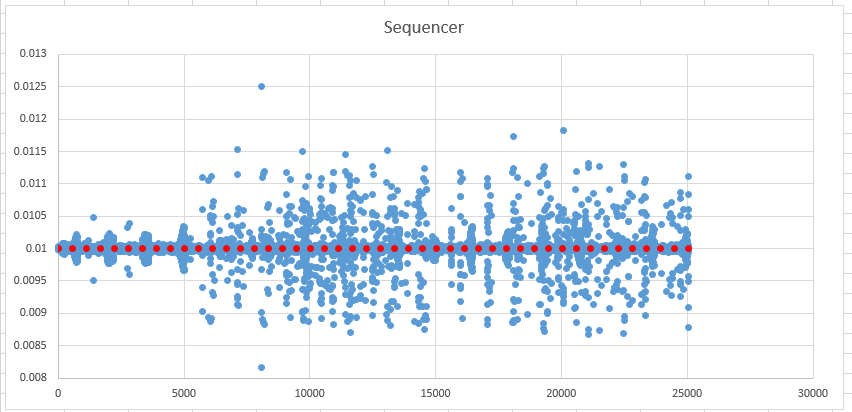
The time-stamps are registered properly.

Jitter Analysis: 

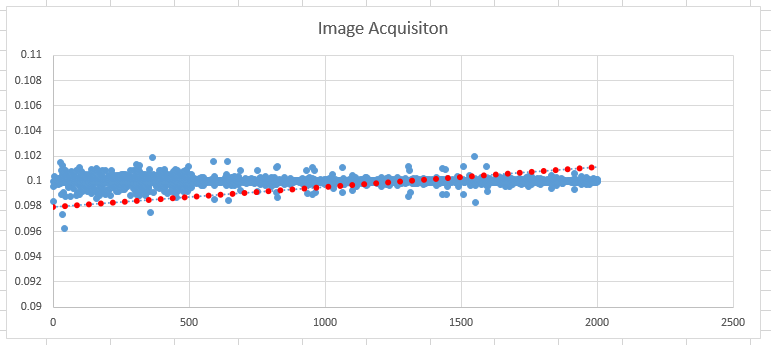
Following are the various plots:

1. Core 1:

Service 0: Sequencer:



Service 1: Image acquisition:

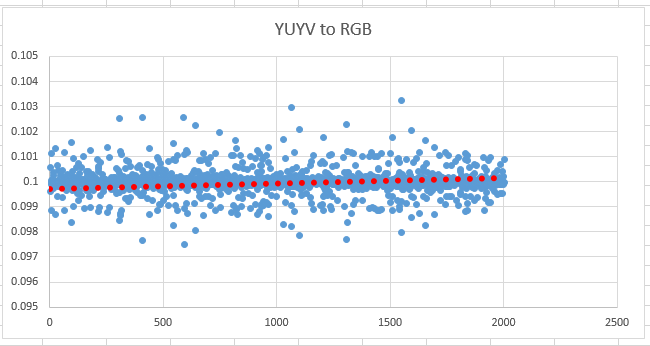


Service X: Frame Selection Service:

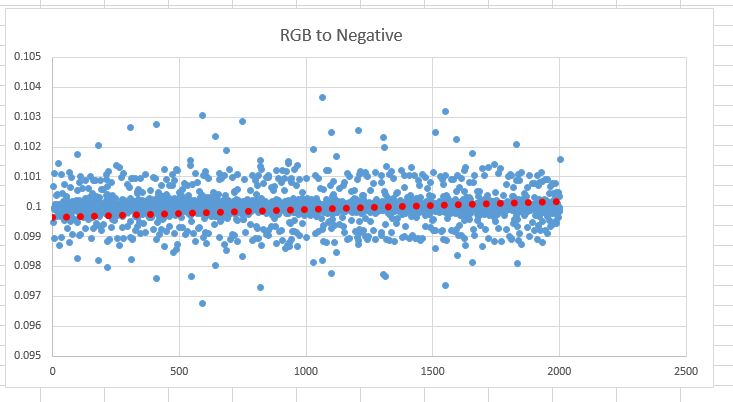


1. Core 2:

Service 2: YUYV to RGB

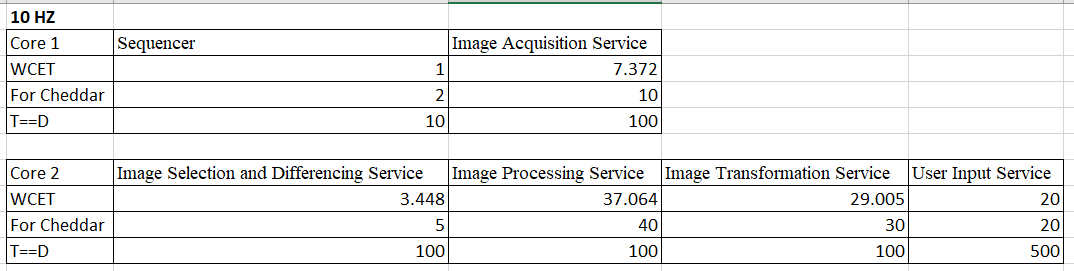


Service 3: RGB to Negative



Another service is User input as Service 4, it is a getchar () service which remains blocked and does not play any role.

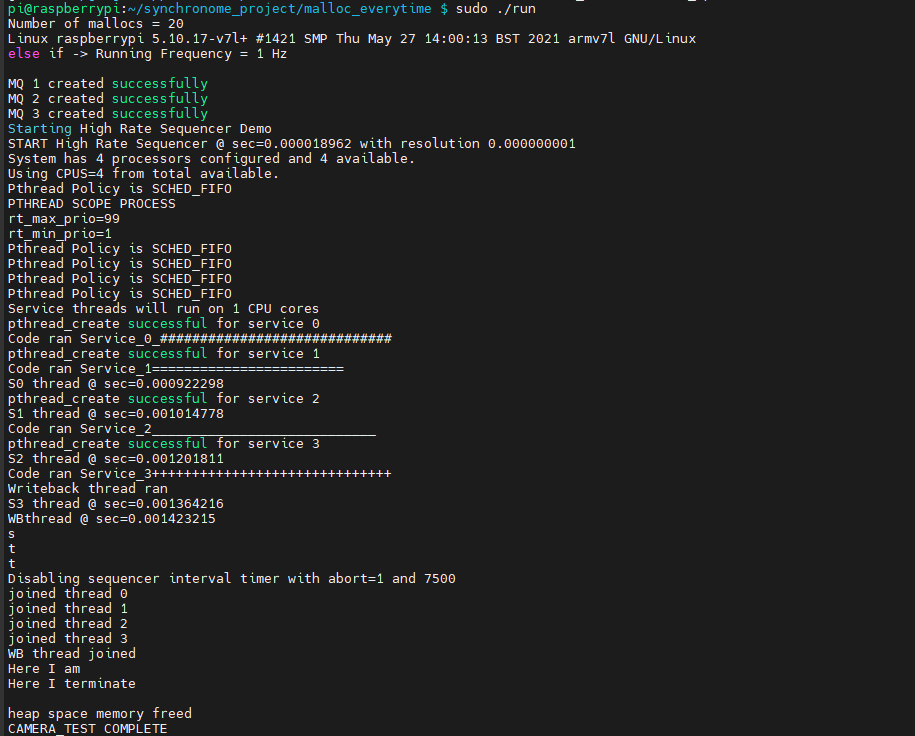
Following is the WCET analysis arrived practically:

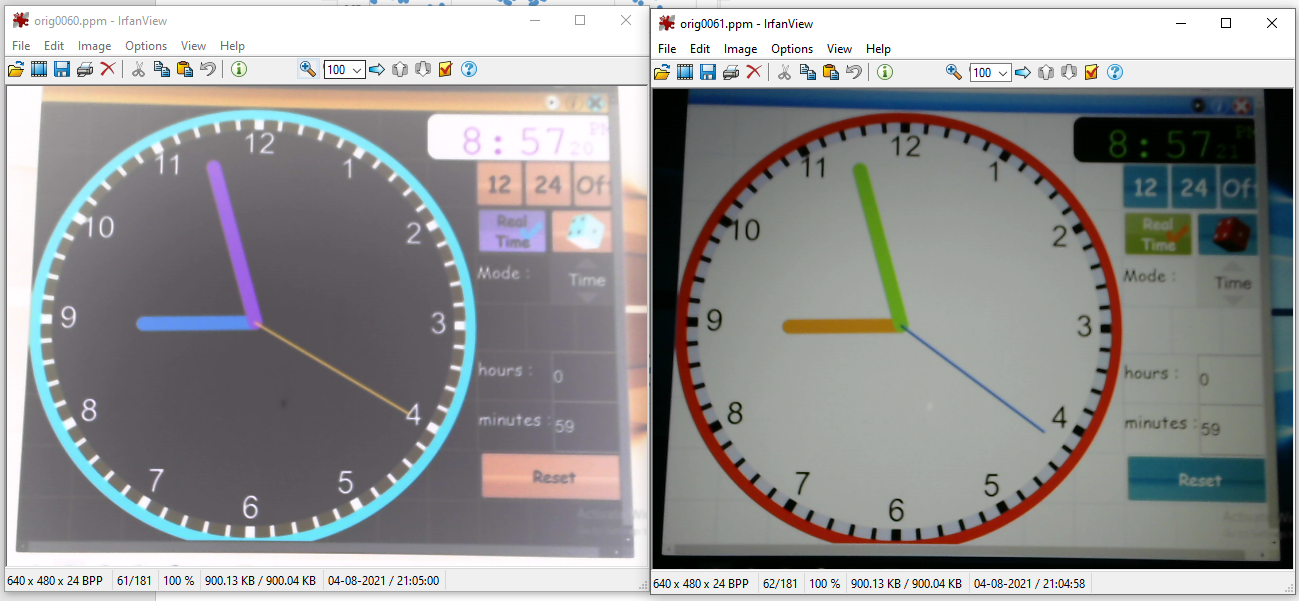


*Of the frames uploaded on google drive, orig0547.ppm, orig0554.ppm, orig0767.ppm, orig0774.ppm, orig1244.ppm, orig1294.ppm and orig1754.ppm have a blur whereas orig00817.ppm is a miss. Rest other frames are clear. This output was achieved when everything was run in single user mode and at 10 Hz.*

## 1 Hz test run time transformation change

The ‘t’ is used to toggle between color and negative image to be dumped





**Constant is the request frequency of the design**

From the jitter analysis, it is clear that my system can handle if everything is run at 1Hz.

It is clearly visible from the above section’s jitter analysis that the system has less or not Jitter.

The service is executed at the rate which it is expected to run.

In the code, Programmable Interrupt Timer (PIT) is used. This is quite reliable. This was the theory.

Now, it is also quite clear from the jitter analysis, that the jitter is quite within the deviation.

Inspire the jitter being less, the code would work find with buffers implemented as any of the overshoot in values due to write back service will get handled in that case. The code would fail miserably even with very smooth almost jitter free graphs if any high CPU demanding service gets scheduled.

# CONCLUSION

**1 Hz:**

The System for 1 Hz sampling with shotgun approach is working with accuracy and functionally correct as well as reliable. There are no glitches or skips or repeats. I have used input circular buffer to take images in a rotation sequence. I am using sampling at 10 Hz frequency for 1 Hz (oversampling) to always collect the correct frame. This is combined with the shotgun approach. This approach is easy as well. Whenever feasible, we should oversample to increase reliability. There was unreliable output when message queues or circular buffer were not used. Using that improved the system reliability and output was more reliable.

I can change the transformation and start the code during run-time for 1 Hz. I have understood how to schedule services. It is also noted that a slight jitter can affect the deadlines or cause skips. When distribution is done of services as per their needs, margin, etc. a better response is obtained.

The implementation submitted as Exercise 6 was very confined and the system was not modular as well as flexible. Hence, the design was modified and is now flexible.

***I can definitely say that my design is ready to deliver for 1 Hz as it is reliable due to oversampling as long as the user using shotgun manages to register the first tick properly.***

**10 Hz:**

For 10 Hz, there were two approaches tried. In on approach, I sampled everything at 10 Hz and processed as well as everything at 10 Hz. This approach is very dependent on the status of the system more specifically RAM. If the system was just rebooted, it works reliably most of the times with very few glitches or errors whereas when the system is loaded, it does not behave properly. This approach is not correct for designing an embedded system especially real time embedded systems (applicable for both soft and hard) as the output is not deterministic as it is in the 1Hz case.

To ensure that I have a functionally correct and reliable embedded system an attempt was made to add a frame differentiation logic which would account for the change in the pixel values. If the change crosses a particular threshold, then it should consider as a change in the number. This logic has flaws in the terms to determine the exact threshold and would vary from daylight to night light conditions. A more approach would be to monitor the pixels for a particular target location and look for changes in pixel values on a percentage basis.

***I cannot commit on the reliability to capture images at 10 Hz rate.***

The 10 Hz log achieved has one skip and 7 ghosts. This was achieved with single user mode. The reason being a lot of background processes do not run allowing the OS to schedule all its services on CPU core leaving less interference to core 1,2 and 3. Thus, better results are achieved. Everything was run on 10 Hz to achieve this output.

***RMA does definitely tells if a given set of services are schedulable or not but it won’t give any idea if the services would achieve a functionally correct output. It is quite clear that the design also has to be very strong to achieve good results.***

***If I got to redo, I would use a circular buffer for capturing images instead of message queues and run the acquisition service at whatever rate it could and not restrict it to 20 hz and use a more better differentiation logic to detect change in the relevant frames.***

# FORMAL REFERENCES

I sincerely express my gratitude to Prof. Sam Siewert for guiding and helping me throughout the coursework and the project.

**References:**

[1] <https://en.wikipedia.org/wiki/Shortt%E2%80%93Synchronome_clock>

[2]The starter code <http://ecee.colorado.edu/~ecen5623/ecen/ex/Linux/code/Std-Project-Starter-Code/> was used for the project (V4L2)

[3]The following video and links were consulted to get an understanding of the software design.

[4]<http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Coursera-rough-cuts/RTES-Example-RT-Synchronome-1-Hz-and-10-Hz-Demo/>

<http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Coursera-rough-cuts/RTES-Demo-Full-Final-Cut/>

[5]<http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Coursera-rough-cuts/RTES-Demo-10-Hz-Demo-Only-Final-Cut/>

[6]<http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Coursera-rough-cuts/RTES-Demo-1-OpenCV-Diff-Interactive/>

[7]<http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Coursera-rough-cuts/RTES-Demo-1-Hz-Demo-Only/>

[8] <https://www.epochconverter.com/>

# APPENDICES

## CODE BULD AND RUN

1. Github link: <https://github.com/Rishab-Shah/SynchronomeTimeLapse_RTES.git>
2. Open the terminal
3. Go into the project directory
4. Do make clean
5. Do make all
6. Inside logs folder, run store the syslog in analysis.txt file
7. Do sudo ./run help for all the menu options
8. Wait for the program to end execution
9. Run the sorter\_script.sh inside the log file to segregate the files
10. Plot the graphs in excel and analyze

Google drive link for Images:

<https://drive.google.com/drive/folders/1s03-oqraZKBGPjPS0gGVOPSrOvaZhS-O?usp=sharing>