University of Colorado
Boulder

EXERCISE #6 – REAL-TIME SOFTWARE SYSTEMS

FINAL PROJECT

REAL TIME EMBEDDED SYSTEMS (ECEN 5623)

SUMMER 2023

STANDARD SYNCHRONOME

Anuhya Kuraparthy

Introduction:

Standard Synchronome: The project enables capturing of the frames of a clock in motion at a designed frequency. The main inspiration behind this project is the timekeeping system developed in the 19th century, synchronome. It was designed to provide highly accurate synchronization in various application where precise time was of utmost importance. The project is intended to concentrate on the design of a soft real-time system. The system makes use of a Raspberry Pi 4 with OS based on Debain, a Linux-based operating system and Logitech C270 Webcam. With this hardware in-place, there is a multi-threaded application to run the system at both 1 Hz and 10 Hz to capture frames from both an analog clock and digital clock. The captured frames are without any glitch ie. skips, repeats or blurs. The project is supported by real time features offered by Linux like preemptive kernel, scheduling policies, and high-resolution timers. The webcam is integrated using the Video4Linux version 2 (V4L2), which is a kernel-mode API designed for managing video capture devices. They provide a low-level interface for more direct control and offer lower overhead.

Functional Requirements:

Functional capabilities requirements for real-time software system:

1. The system should be capable of capturing and processing the acquired frames from the camera at both 1Hz and 10Hz frequencies. The user should be able to choose the system frequency.
2. The system should support scheduling of services and thread priorities to manage the real-time services efficiently. Initial and more critical tasks will have higher priority to make sure the deadlines are not missed. This is to reduce the system jitter and observe a more predicable response.
3. **Frame resolution:** The camera resolution is selected based on the timing and memory constraints to be 640x480. The selected resolution also resolves the image clarity and the clock can be clearly observed.
4. **Timestamping:** A service/thread experiences drift when there an accumulation of request and execution jitter over time such that the latency increases proportionally. This would cause deadlines to be missed and system unpredictability. To ensure the real-time services are not experiencing any drift due to any reasons (execution jitter, response jitter), the timestamp of when each service is scheduled is measured using the clock_gettime() API.
5. **WCET Measurement:** To check if any deadlines are being missed for the scheduled threads, every service call's execution time is measured with a start of service and end of service timestamp. The maximum of all the measured execution times for each service is assigned as the WCET (worst case execution time) for that particular service. Both timestamping and WCET measurement aid in debugging and identifying any timing-related discrepancies.
6. **Time stamps in Image header:** The timestamps are embedded in the PPM/PGM header of every frame as a comment field. This is done so the accuracy of frame acquisition can be verified from timestamps taken.

Real-Time Requirements:

1. **Sequencer:** The Sequencer controls the scheduling of the threads by making use of semaphores. The semaphores are posted at a multiple of the sequencer's frequency to schedule the threads at their respective frequencies. The sequencer runs at a frequency of 100Hz. This is done with a system timer created with the function timer_create which is based on the system's real-time clock (CLOCK_REALTIME). When the timer expires, it generates a 'SIGALARM' signal that will trigger the Sequencer function. When the number of required frames is captured, all the service threads are aborted. The timer function is chosen as it gives more accurate response than the clock APIs such as clock_nanosleep().

2. **Frame Acquisition:** The frame acquisition service runs at 5Hz or 20Hz depending on the user input (1Hz or 10Hz). The frames acquired are stored in the system buffers for processing. The camera, Logitech C270, offers a highest frame capturing rate of 30 fps ie. 30Hz.
    - **C1:** 1 msec (expected)
    - **T1:** 200 msec or 40 msec (based on user input)

3. **Frame Processing:** The captured frames need to be processed into a known pixel format. The service runs at 1Hz or 10Hz based on the user input (1Hz or 10Hz). As an additional feature implementation, the continuous transformation- negative image is selected. This is applied to a real-time stream of image frames acquired.
    - **C2:** 20 msec (expected)
    - **T2:** 1 sec or 100 msec (based on user input)

4. **Frame Storage:** Frame storage is a best-effort service scheduled on a different core as it has the most unpredictable response times. The images are saved as a ppm (Portable Pixmap) format for frame acquisition at 1Hz and pgm (Portable Graymap) format for frame acquisition at 10Hz. This is done to manage system resources and timing constraints.
    - **C3:** 3 sec (expected)
    - **T3:** 1 sec or 100 msec (based on user input)

Functional Design Overview and Diagrams

The high-level design for the system which includes both hardware and software elements is shown below in Figure 1. The main hardware components of the project include – an analog clock to observe the seconds or a digital clock to observe 100 milliseconds. The camera used in the project is C270 LogiTech Webcam. The project aims to make use of the V4L2 driver to capture frames to observe the mentioned clocks at 1Hz and 10Hz with the camera device. The V4L2 provides low-level access to video devices by giving direct control over the hardware features. This significantly reduces the response latency when compared with using OpenCV APIs.

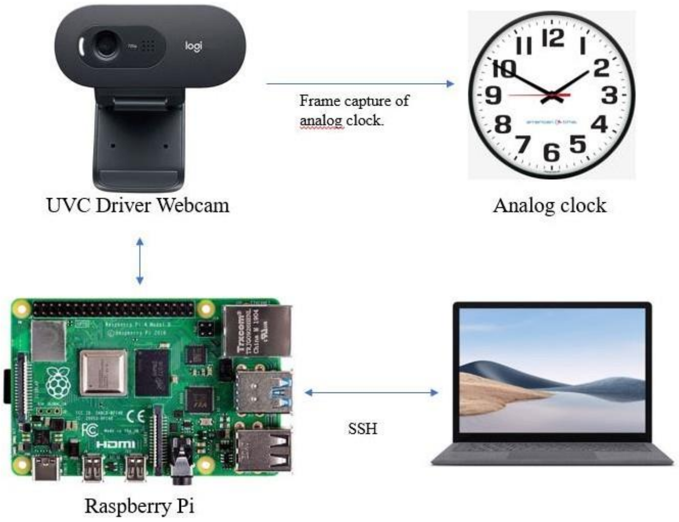


Figure 1: Hardware setup

The user has options to enable the features offered by the program and set the frequency of frame acquisition. Based on the user input, the frame acquisition rate is changed to 1Hz or 10Hz. The project offers feature for negative transform of the acquired frame. These features are to be selected with a key-press during the execution of the program.

The program follows the following execution process:

1. With the V4L2 driver, the camera starts capturing the frames at the specified frame rate.
2. The captured frames are stored in a ring buffer so there is IO decoupling between the services which read and process the frames.
3. The selected frame is processed from the YUYV format received from the webcam into RGB or grayscale format depending on the selected frame acquisition rate.
4. If the feature for frame transformation is enabled, the processed frame is negated to reproduce a negative frame.
5. The processed/transformed frame is saved on the device with the header information-timestamp at the processed time.
6. For every service, the start and stop timestamps and execution times are printed as syslog information.

The following threads are created to perform each of the mentioned functions: Service_1_frame_acquisition, Service_2_frame_process, Service_3_frame_storage. The real-time services wait on its respective semaphore which is posted by the sequencer to control the thread frequencies. While the sequencer runs at the highest frequency, the services/threads run at a frequency that is a factor of the sequencer frequency.

The program reaches the end of its control when the required number of captured frames are saved and with this, the respective abortS<i> flags are set. With this, the threads' execution ends and they are synchronized with the main process. The main thread, real-time threads and best effort services are explained in detail in the next sectionwith their corresponding flow chart diagrams.

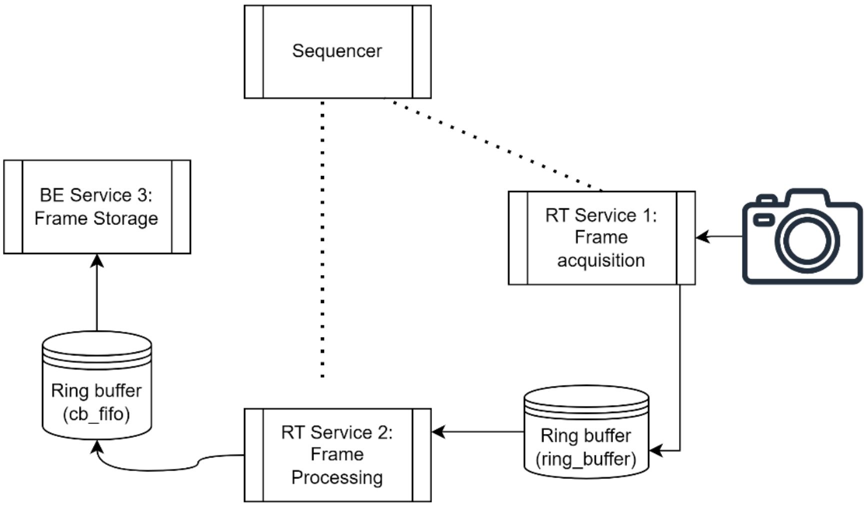

Sequencer
BE Service 3: Frame Storage
RT Service 1: Frame acquisition
Ring buffer (cb_fifo)
RT Service 2: Frame Processing
Ring buffer (ring_buffer)

Figure 2: High-level system overview

The system block diagram is shown below for both 1Hz and 10Hz frame rate.

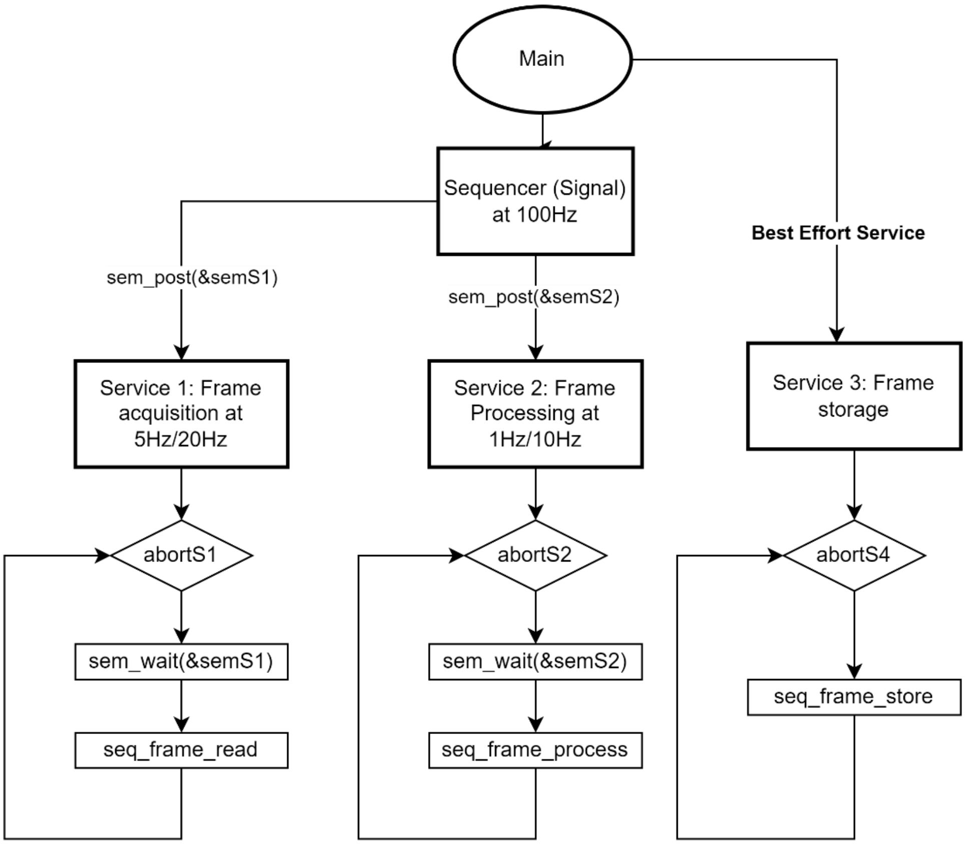


Figure 3: System block diagram

Main thread:

In the main thread, the camera is initialized with V4L2 driver. The device is opened as a file with read and write flags enabled following Linux ideology- ‘Everything is a file’. The device is initialized by setting the method of IO as ‘IO_METHOD_MMAP’ and video format to YUYV. After initializing the video capture buffers, an xioctl call is made to start video streaming. The synchronization mechanism used is semaphores and these are initialized to locked state as well. Based on the user input, the frequencies of the threads are set. The threads are assigned to each core by setting the CPU affinity of their respective pthread attributes. The real-time threads scheduled on Core 2 are: Service_1_frame_acquisition and Service_2_frame_process. Best effort thread is Service_3_frame_storage assigned to a different core. After assigning priorities to each service, the threads are created. Along with this, system timer is created based on the system’s real-time clock. When the timer expires, it generated a ‘SIGALARM’ signal that will trigger the Sequencer function. The sequencer is responsible for releasing (sem_post) the semaphores for every thread execution.

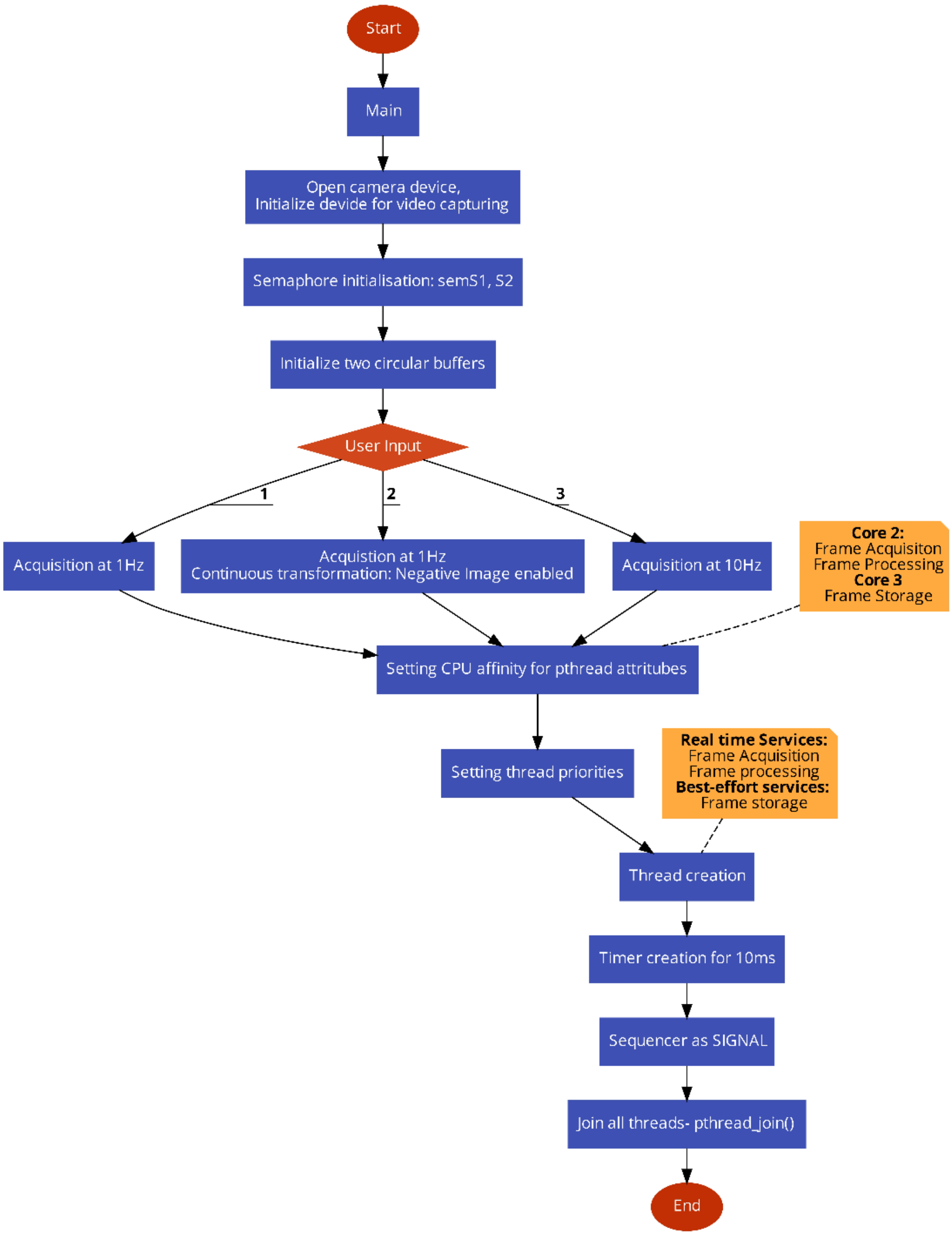


Figure 4: Main function block diagram

Real-time threads:

1. **Service_1_frame_acquisition:** The thread runs at 5Hz or 20Hz based on the user input and requirements. The select() system call is used to wait on the file descriptor for 'timeout' amount of time. Then the thread dequeues the captured buffer in the kernel driver's outgoing queue to read the frame information from the camera. The received buffer is stored at the tail of the ring buffer. A circular buffer is implemented to provide IO decoupling for a more predictable and controlled program [2].

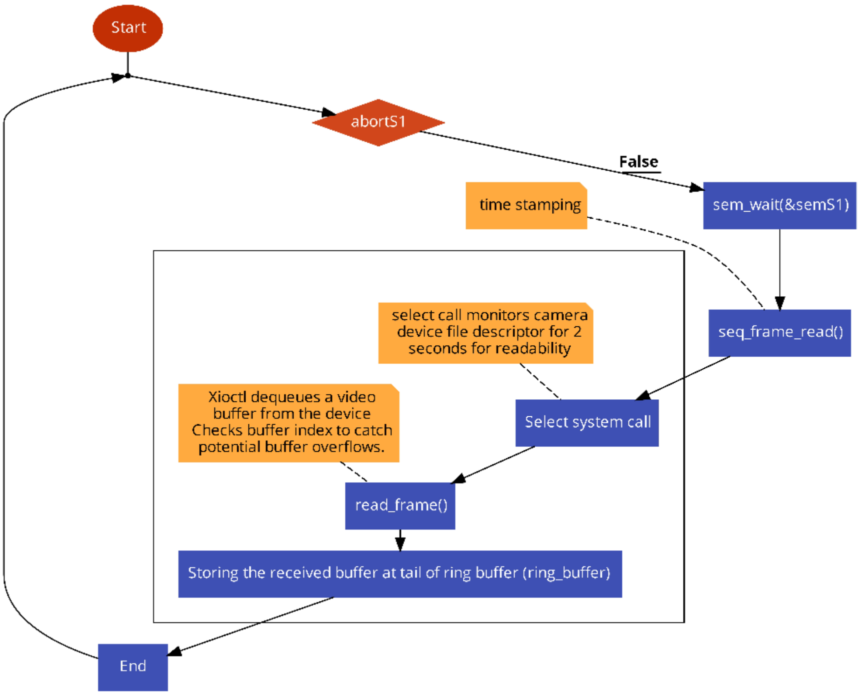

Start
abortS1
False
time stamping
sem_wait(&semS1)
seq_frame_read()
select call monitors camera device file descriptor for 2 seconds for readability
Select system call
Xioctl dequeues a video buffer from the device Checks buffer index to catch potential buffer overflows.
read_frame()
Storing the received buffer at tail of ring buffer (ring_buffer)
End

Figure 5: Block diagram of frame acquisition service

2. **Service_2_frame_process:** The thread runs at 1Hz or 10Hz based on the user input and requirements. The frame stored at the head of the ring buffer (ring_buffer) is processed. Depending on the user input of frame rate, the YUYV pixels are transformed to RGB or YY ie. black and white. 4 bytes of YUYV pixels are read and transformed into 6 bytes of RGB data. For ppm image requirements, alternating bytes of data ie. Y are taken. On the processed frame, a continuous transformation ie. negative image algorithm is applied. This function is performed if the feature is enabled by the user. The processed/transformed frame is stored at the tail of the ring buffer (cb_fifo). This is done to provide decoupling between the services. The block diagram is shown in Figure 6.

Best-Effort Services:

3. **Service_3_frame_storage:** Based on the user input, the image is stored in a ppm or pgm format. A file to store the image is created using open() system call in create mode and non-blocking mode. The processed/transformed buffer at the head of the ring buffer (cb_fifo) along with header information is written into the file.

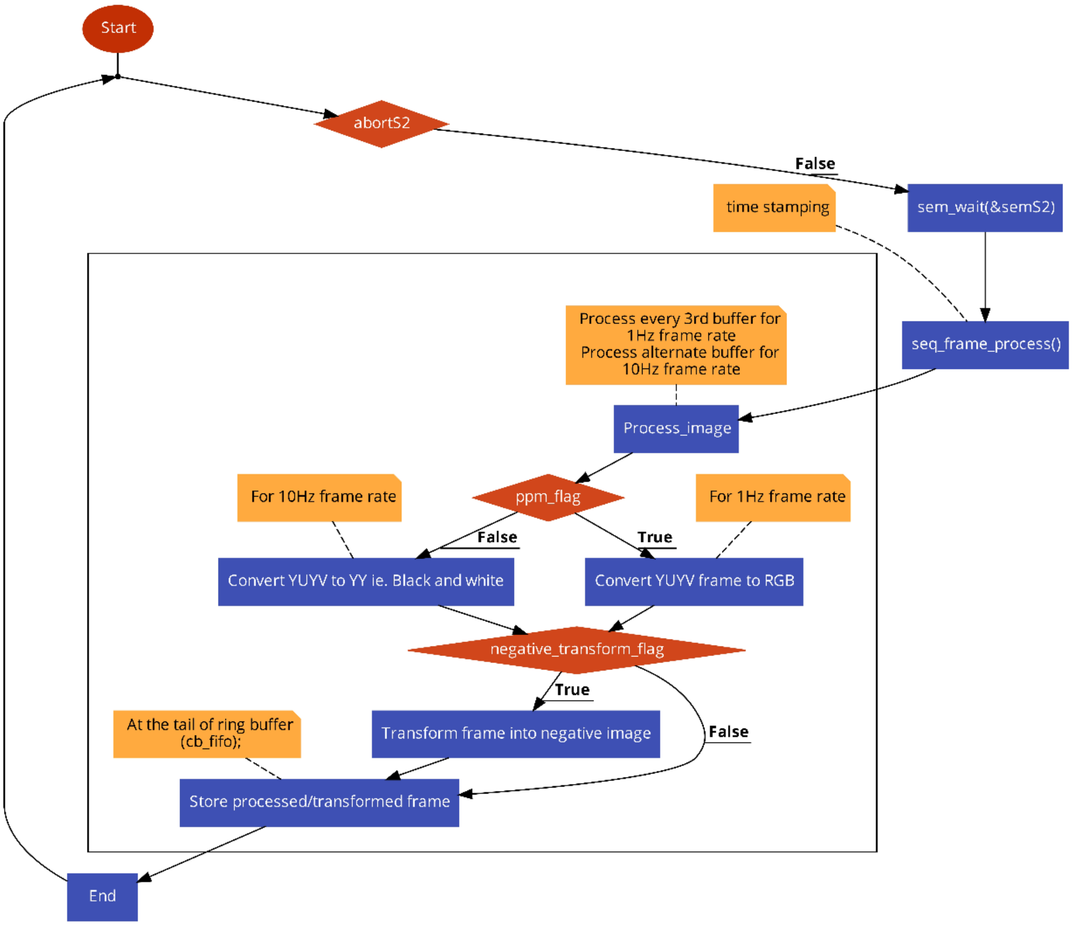

Start
abortS2
False
time stamping
sem_wait(&semS2)
seq_frame_process()
Process every 3rd buffer for 1Hz frame rate Process alternate buffer for 10Hz frame rate
Process_image
For 10Hz frame rate
ppm_flag
For 1Hz frame rate
False
True
Convert YUYV to YY ie. Black and white
Convert YUYV frame to RGB
negative_transform_flag
True
False
At the tail of ring buffer (cb_fifo);
Transform frame into negative image
Store processed/transformed frame
End

Figure 6: Block diagram of frame processing service

Real-Time Service Requirements:

- The system design is made of two real-time services. The threads are scheduled to run at their respective frequencies by setting the scheduling policy to SCHED_FIFO. Threads scheduled with this policy have a fixed priority and they run until they voluntarily yield the CPU or a higher-priority thread is added to the queue in the kernel scheduler. Since the threads have constraints on pre-emption, scheduling is done only after through analysis of timing and memory resources. Any shared memory resources can lead to a potential deadlock or else deadlines could be missed if the CPU is overloaded. A program-based scheduler is designed to control the release of semaphores for each thread. Semaphores are used here as a synchronization mechanism to coordinate the execution of multiple threads.
- The board used is Raspberry Pi 4 which uses the Broadcom BCM2711 SoC. The SoC features a quad-core ARM Cortex-A72 CPU. To control the real-time execution of threads with 4 available cores, all the real-time threads are scheduled to run only on Core 2. The rest of the services are assigned to separate cores as best-effort services. The CPU affinity for each real-time thread is set by assigning the thread attribute to a particular core.
- To perform Rate-Monotonic Analysis to check the schedulability of the system, parameters for each service must be evaluated. These parameters include WCET (worst case execution time), deadline and period. Deadline refers to the time within which the service must be completed and period refers to the time interval between the start of consecutive instances of execution. In RMA, period is assumed to be equal deadline to ensure that the task's execution time is limited to the period.
- The WCET is determined for each service by running the program for the entire duration and calculating the maximum of the execution times. The execution time for each thread call is measured as a difference of the end timestamp and start timestamp. If the system is schedulable with all the service's WCETs and ensures that no deadlines are missed, then the system is determined to be feasible. The parameters- deadline and period for the services vary based on the user input. The threads run at different frequencies to emulate different system architectures to manage system requirements. First the analysis for 1Hz frame rate is described and then the analysis for 10 Hz frame rate is presented.

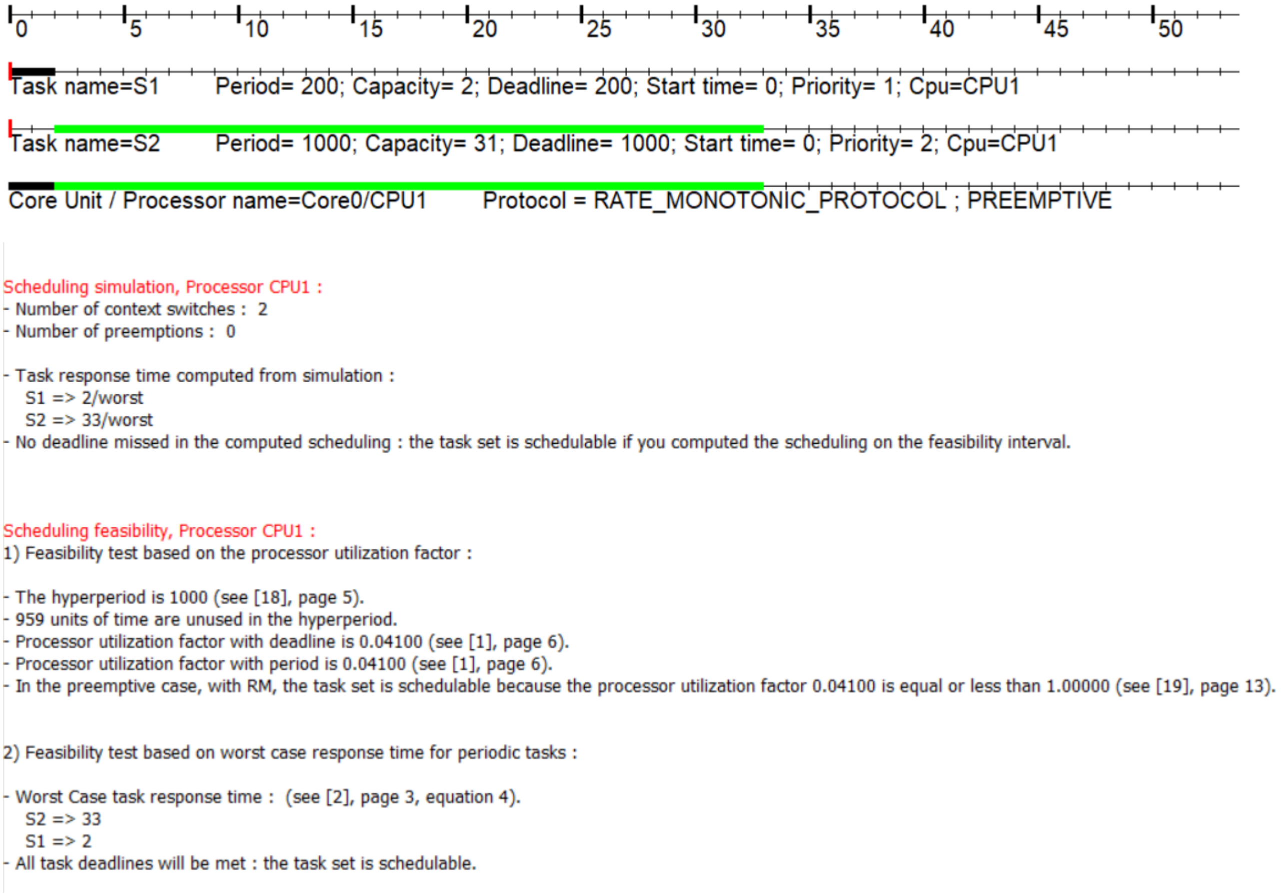
For 1Hz frame acquisition and storage:

1. **Service_1_frame_acquisition:** With the V4L2 driver, the camera starts capturing the frames at the specified frame rate. The captured frames are stored in a ring buffer so there is IO decoupling between the services which read and process the frames. According to the Nyquist theorem, the sampling rate should be at least twice the frequency of the highest frequency. For acquiring frames at 1 Hz, the camera needs to capture frames at at least 2 frames per second. For this reason, the thread frequency is selected to be 5 Hz. Thus, the deadline and period for the service are both 200 milliseconds.
2. **Service_2_frame_process:** The selected frame is processed from the YUYV format received from the webcam into RGB. Only one frame is selected to be processed and thus, it can be processed at the rate of 1 Hz. The processed frame is negated to reproduce a negative frame, when the frame transformation is enabled.

In the table below, all the parameters- C_i, T_i, D_i are listed out for each service, S_i. The execution time listed is the maximum of all the execution times for that particular service and thus, it is the WCET (worst case execution time) for that service. The WCET is observed when the frame transformation is enabled.

Si	Service name (S_i)	Execution time (C_i)	Time period (T_i)	Deadline (D_i)
S1	Service_1_frame_acquisition	1.25963 ms	200 ms	200 ms
S2	Service_2_frame_process	30.141981	1000 ms	1000 ms

Cheddar Analysis: Cheddar analysis is done to emulate Rate Monotonic scheduling policy with the calculated parameters. The program calculates the RM LUB (Least Upper Bound) and schedules the threads based on the priorities assigned and time periods calculated. This is shown in the figure below.



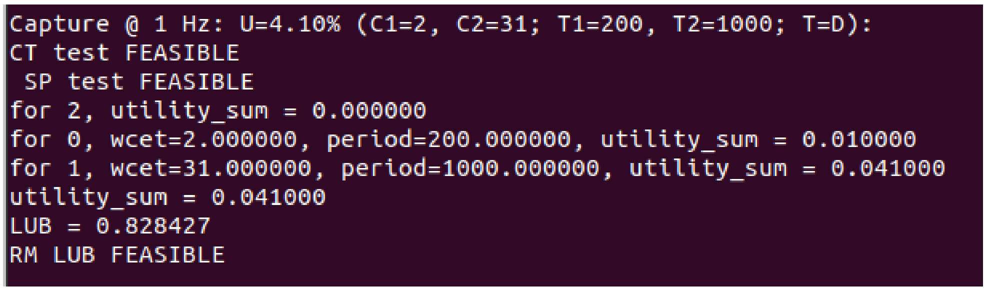
```
Task name=S1      Period= 200; Capacity= 2; Deadline= 200; Start time= 0; Priority= 1; Cpu=CPU1

Task name=S2      Period= 1000; Capacity= 31; Deadline= 1000; Start time= 0; Priority= 2; Cpu=CPU1

Core Unit / Processor name=Core0/CPU1      Protocol = RATE_MONOTONIC_PROTOCOL ; PREEMPTIVE
```

```
Scheduling simulation, Processor CPU1 :
- Number of context switches :  2
- Number of preemptions :  0

- Task response time computed from simulation :
   S1 => 2/worst
   S2 => 33/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.


Scheduling feasibility, Processor CPU1 :
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 1000 (see [18], page 5).
- 959 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.04100 (see [1], page 6).
- Processor utilization factor with period is 0.04100 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.04100 is equal or less than 1.00000 (see [19], page 13).


2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :  (see [2], page 3, equation 4).
   S2 => 33
   S1 => 2
- All task deadlines will be met : the task set is schedulable.
```

Figure 7: Cheddar analysis for 1Hz frame rate

With the scheduled threads, the processor utilization is 4.1%, compared to the RM LUB of 82.84%. Thus, the system provides a margin of error of 95.9% while to still enable the feasibility of the system, the margin is 78.16%. In the case for varying execution times (C_i), the system holds the same margin of error as the schedulability is designed with the worst-case execution time. Thus, the system is still feasible and safe for varying C_i times [3].

The feasibility tests program checks the feasibility of the service set by running the completion time and scheduling points tests and also compares the CPU utilization with the RM LUB [4]. The competition time and scheduling point tests will precisely pass the service set. When the program is run with the parameters for the three services, it can be seen that the service set passes both the completion test and scheduling point test. The program output is shown in the figure below, Figure 7.



```
Capture @ 1 Hz: U=4.10% (C1=2, C2=31; T1=200, T2=1000; T=D):
CT test FEASIBLE
 SP test FEASIBLE
for 2, utility_sum = 0.000000
for 0, wcet=2.000000, period=200.000000, utility_sum = 0.010000
for 1, wcet=31.000000, period=1000.000000, utility_sum = 0.041000
utility_sum = 0.041000
LUB = 0.828427
RM LUB FEASIBLE
```

Figure 8: Feasibility tests output for 1Hz frame rate.

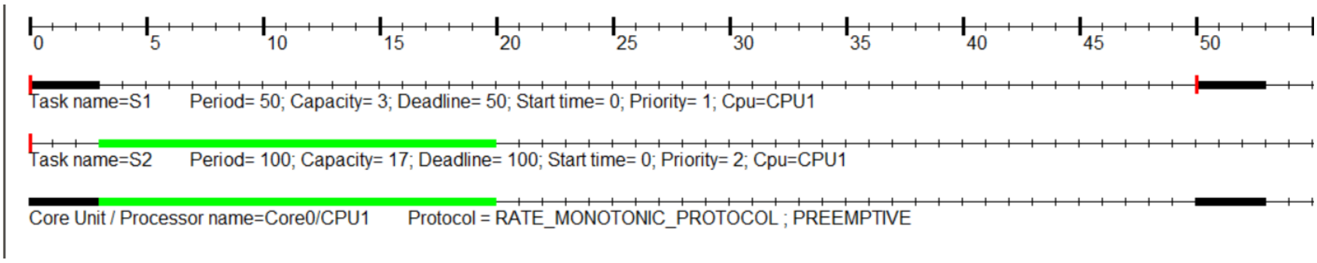
For 10Hz frame acquisition and storage:

1. **Service_1_frame_acquisition:** For acquiring frames at 10 Hz, the camera needs to capture frames at at least 2 frames per 100 milliseconds. For this reason, the thread frequency is selected to be 20 Hz. Thus, the deadline and period for the service are both 50 milliseconds.
2. **Service_2_frame_process:** The selected frame is processed from the YUYV format received from the webcam into graymap. Only one frame is selected to be processed and thus, it is processed at the rate of 10 Hz.

In the table below, all the parameters- C_i, T_i, D_i are listed out for each service, S_i. The execution time listed is the maximum of all the execution times for that particular service and thus, it is the WCET (worst case execution time) for that service. The WCET is observed when the frame transformation is enabled.

Service name (S_i)	Execution time (C_i)	Time period (T_i)	Deadline (D_i)
Service_1_frame_acquisition	2.711889 ms	50 ms	50 ms
Service_2_frame_process	16.782055 ms	100 ms	100 ms

Cheddar Analysis: With the scheduled threads, the processor utilization is 23%, compared to the RM LUB of 82.84%. Thus, the system provides a margin of error of 77% while to still enable the feasibility of the system, the margin is 59.84%. In the case for varying execution times (C_i), the system holds the same margin of error as the schedulability is designed with the worst-case execution times. Thus, the system is still feasible and safe for varying C_i times.

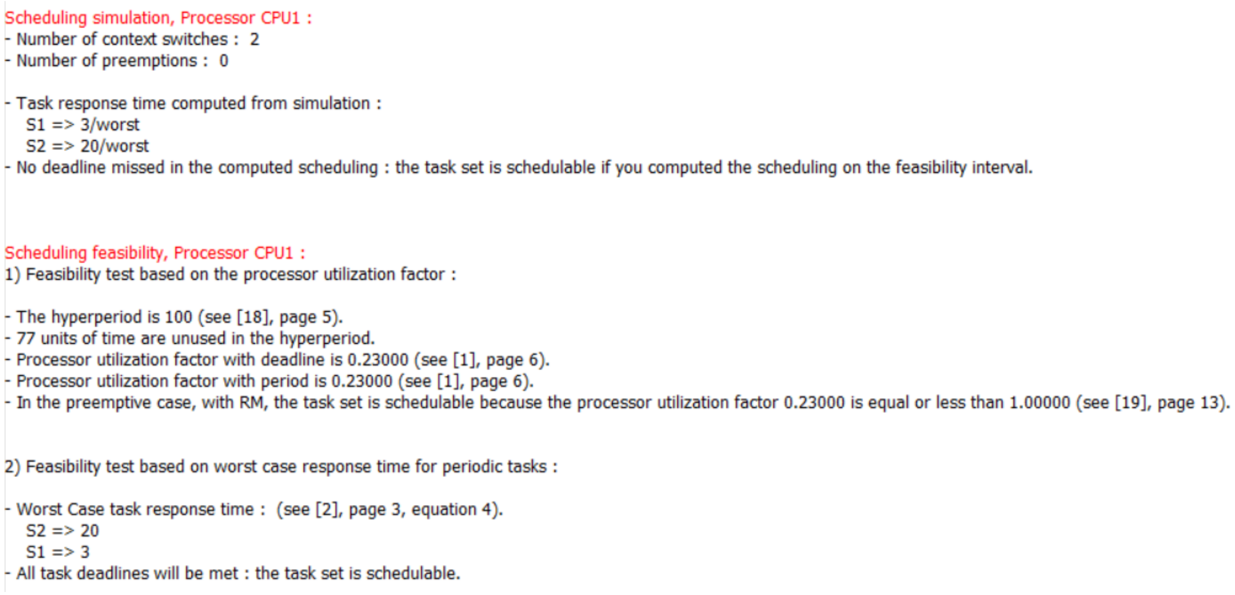


Task name=S1 Period= 50; Capacity= 3; Deadline= 50; Start time= 0; Priority= 1; Cpu=CPU1

Task name=S2 Period= 100; Capacity= 17; Deadline= 100; Start time= 0; Priority= 2; Cpu=CPU1

Core Unit / Processor name=Core0/CPU1 Protocol = RATE_MONOTONIC_PROTOCOL ; PREEMPTIVE

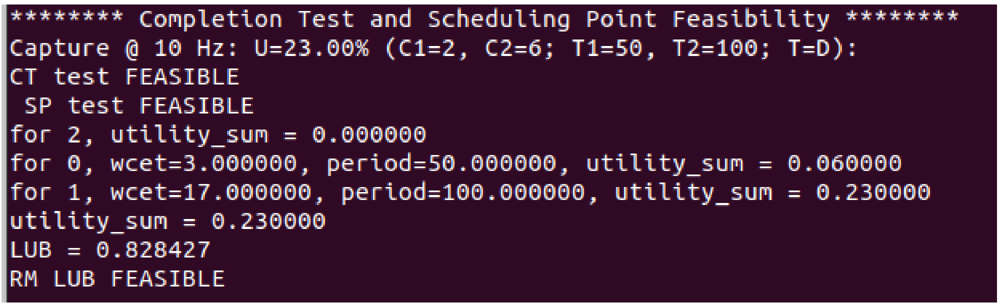
Figure 9: Cheddar simulation for 10Hz frame rate



```
Scheduling simulation, Processor CPU1 :
- Number of context switches :  2
- Number of preemptions :  0

- Task response time computed from simulation :
    S1 => 3/worst
    S2 => 20/worst
- No deadline missed in the computed scheduling : the task set is schedulable if you computed the scheduling on the feasibility interval.


Scheduling feasibility, Processor CPU1 :
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 100 (see [18], page 5).
- 77 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.23000 (see [1], page 6).
- Processor utilization factor with period is 0.23000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.23000 is equal or less than 1.00000 (see [19], page 13).


2) Feasibility test based on worst case response time for periodic tasks :

- Worst Case task response time :  (see [2], page 3, equation 4).
    S2 => 20
    S1 => 3
- All task deadlines will be met : the task set is schedulable.
```

Figure 10: Scheduling feasibility in Cheddar analysis for 10Hz frame rate

The feasibility tests program checks the feasibility of the service set by running the completion time and scheduling points tests and also compares the CPU utilization with the RM LUB. The competition time and scheduling point tests will precisely pass the service set. When the program is run with the parameters for the two services, it can be seen that the service set passes both the completion test and scheduling point test.



```
******** Completion Test and Scheduling Point Feasibility ********
Capture @ 10 Hz: U=23.00% (C1=2, C2=6; T1=50, T2=100; T=D):
CT test FEASIBLE
 SP test FEASIBLE
for 2, utility_sum = 0.000000
for 0, wcet=3.000000, period=50.000000, utility_sum = 0.060000
for 1, wcet=17.000000, period=100.000000, utility_sum = 0.230000
utility_sum = 0.230000
LUB = 0.828427
RM LUB FEASIBLE
```

Figure 11: Feasibility tests output for 10Hz

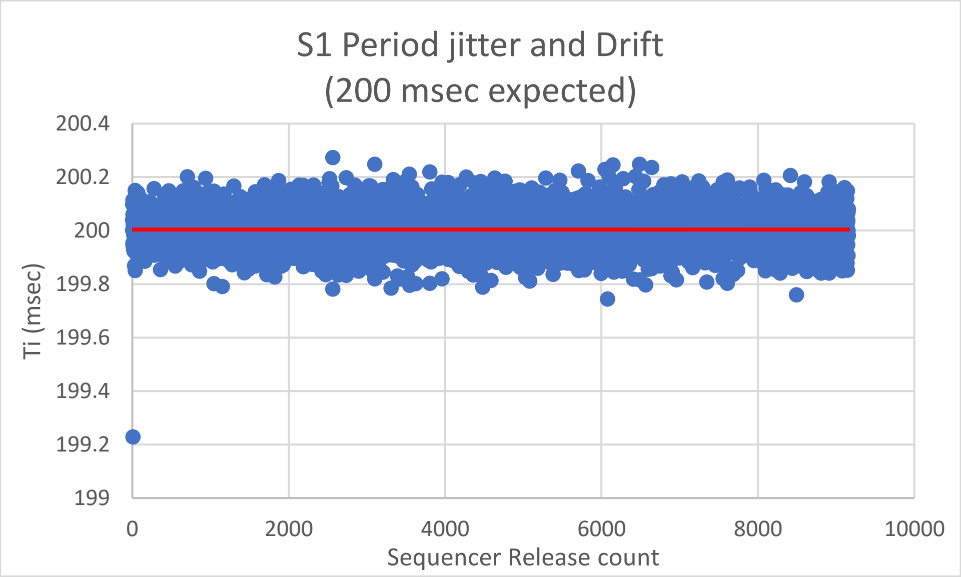
Real-Time Analysis (Observed):

To check the system accuracy and if all the real-time services are meeting the deadlines, the syslog file is parsed with the grep command. Only timestamps related to that particular service are analyzed for jitter and drift [5].

1 Hz frame rate analysis: The following analysis is conducted with the frame transformation enabled so that the system can be checked with WCET timings.

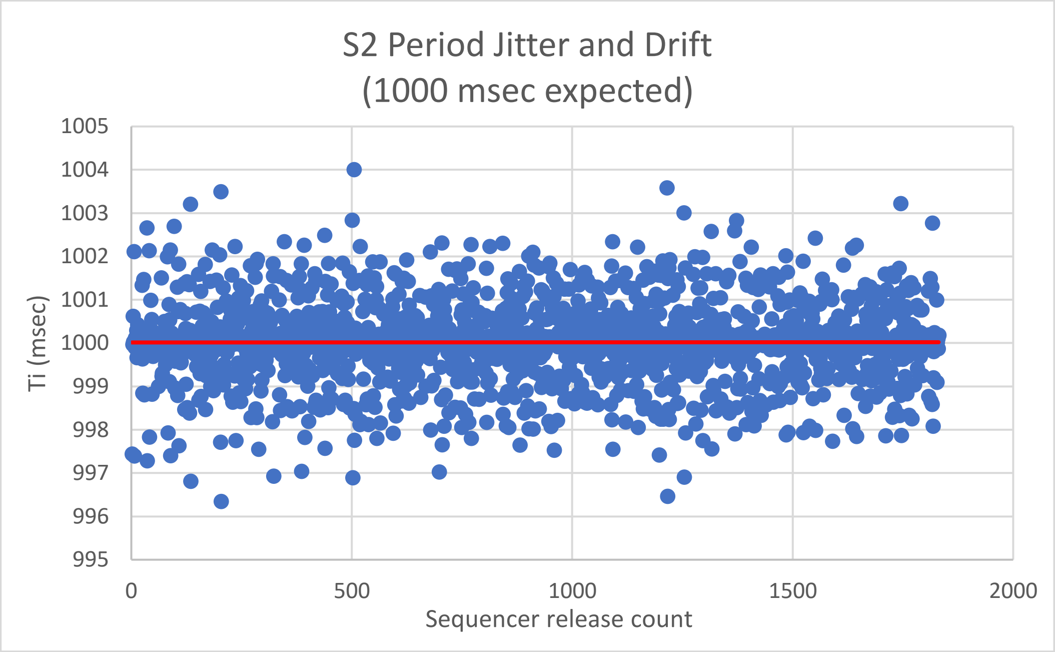
1. **Service_1_frame_acquisition (S1):** With the V4L2 driver, the camera starts capturing the frames at the specified frame rate. The captured frames are stored in a ring buffer so there is IO decoupling between the services which read and process the frames. The thread jitter and drift are shown in the graph below.

C1	1.25963 msec
T1	200 msec
D1	200 msec
Core	Core 2



2. **Service_2_frame_process (S2):** The selected frame is processed from the YUYV format received from the webcam into RGB. Only one frame is selected to be processed and thus, it can be processed at the rate of 1 Hz. The thread jitter and drift are shown in the graph below. The analysis is done with the frame transformation enabled so the worst case execution can be observed.

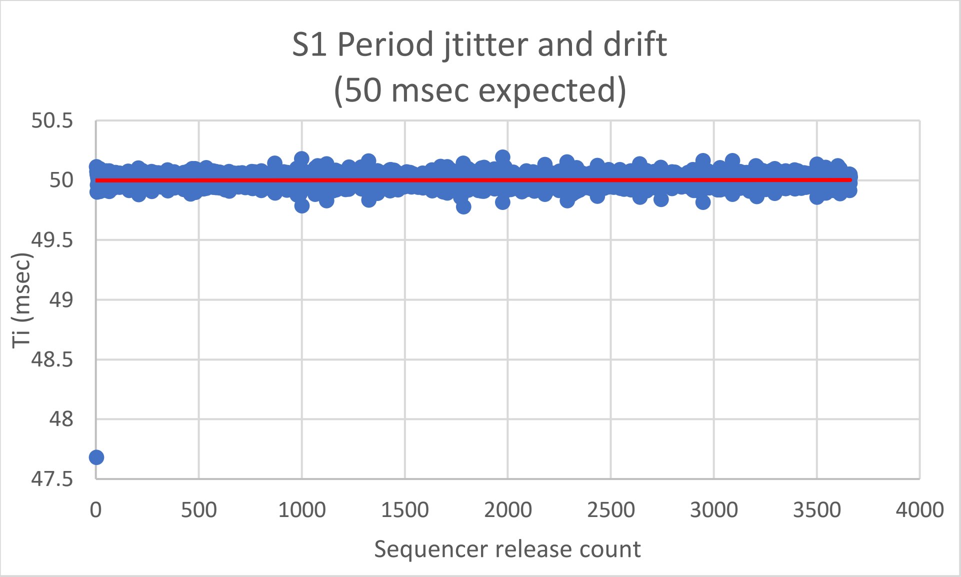
C2	30.14198 ms
T2	1000 ms
D2	1000 ms
Core	Core 2



<u>10 Hz frame rate analysis:</u> The following analysis is conducted with the frame transformation enabled so that the system can be checked with WCET timings.

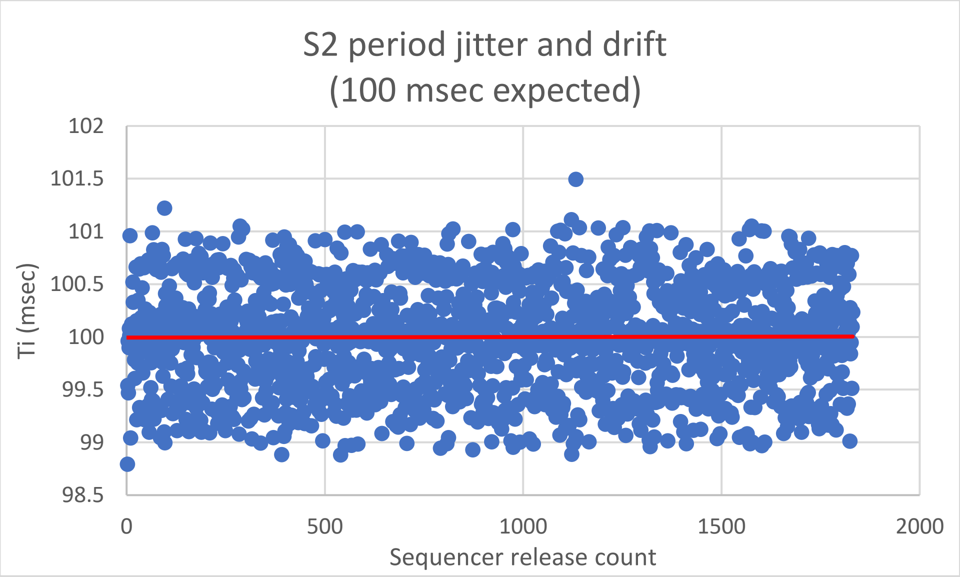
1. Service_1_frame_acquisition (S1): With the V4L2 driver, the camera starts capturing the frames at the specified frame rate. The captured frames are stored in a ring buffer so there is IO decoupling between the services which read and process the frames. The thread jitter and drift are shown in the graph below.

C1	2.711889 msec
T1	50 msec
D1	50 msec
Core	Core 2



2. **Service_2_frame_process (S2):** The selected frame is processed from the YUYV format received from the webcam into RGB. Only one frame is selected to be processed and thus, it can be processed at the rate of 1 Hz. The thread jitter and drift are shown in the graph below. The analysis is done with the frame transformation enabled so the worst case execution can be observed.

C2	16.78206 ms
T2	100 ms
D2	100 ms
Core	Core 2



Proof-of-Concept:

1 Hz test: The program is run with the frame transformation feature enabled at a selection of 1Hz for 30 minutes to capture 1800 frames. The images shown below are captured during the test to verify the proof-of-concept.

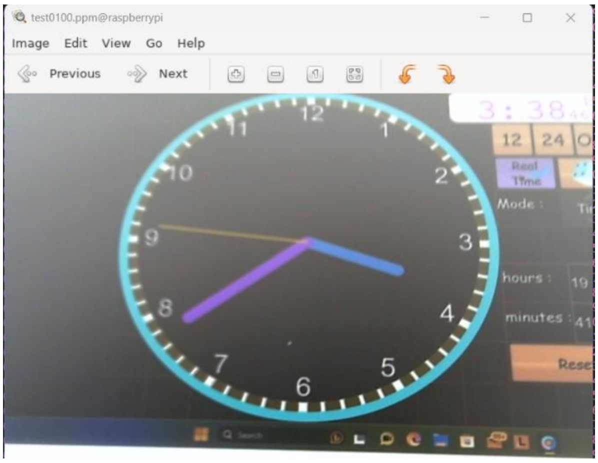


Figure 12: 100th frame opened with the command eom test0100.ppm

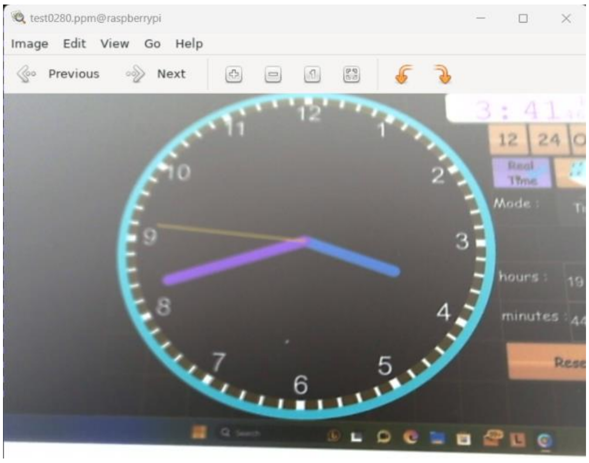


Figure 13: 280th frame opened with the command eom test0280.ppm

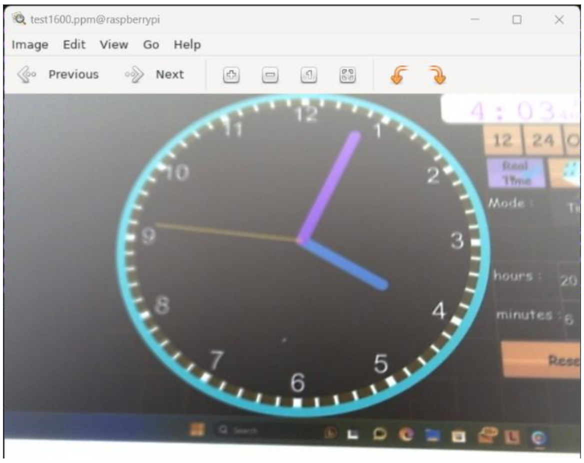
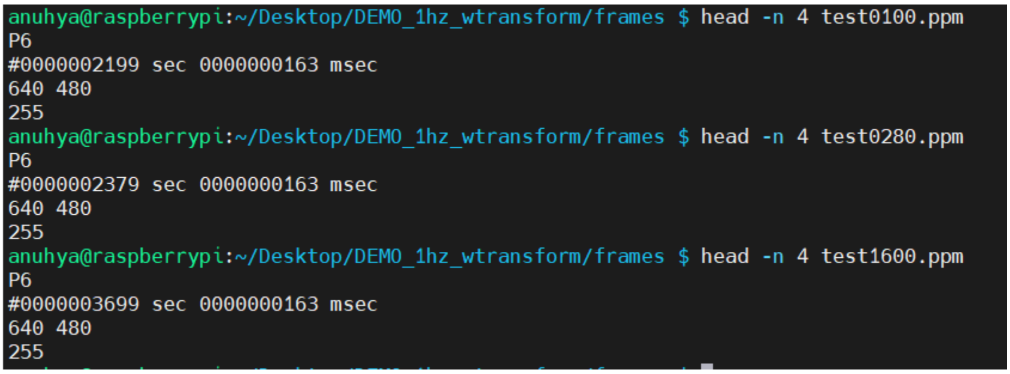


Figure 14: 1600th frame opened with the command eom test1600.ppm

The program frequency is set to be 1 Hz with negative transformation enabled. There is 180 seconds time difference between the 100th frame and 280th frame. Thus, the program is fully functional for 1Hz acquisition rate. This is also verified with the header information stored in each file. This is shown in the figure below.



```
anuhya@raspberrypi:~/Desktop/DEMO_1hz_wtransform/frames $ head -n 4 test0100.ppm
P6
#0000002199 sec 0000000163 msec
640 480
255
anuhya@raspberrypi:~/Desktop/DEMO_1hz_wtransform/frames $ head -n 4 test0280.ppm
P6
#0000002379 sec 0000000163 msec
640 480
255
anuhya@raspberrypi:~/Desktop/DEMO_1hz_wtransform/frames $ head -n 4 test1600.ppm
P6
#0000003699 sec 0000000163 msec
640 480
255
```

10 Hz test: The program program is run with the frame transformation feature enabled at a selection of 10Hz for 3 minutes to capture 1800 frames. The images shown below are captured during the test to verify the proof-of-concept.

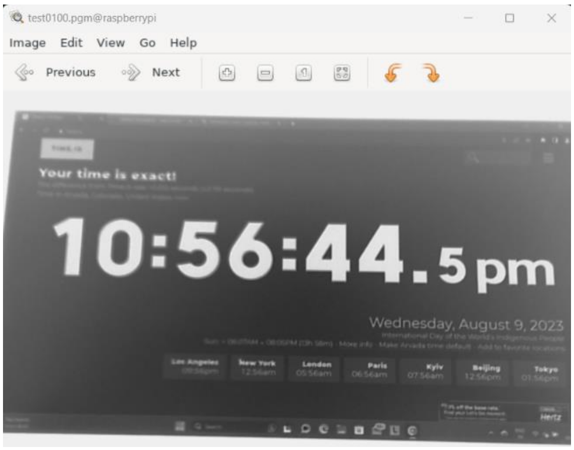


Figure 15: 100th frame opened with the command eom test0100.pgm. This shows the frame acquisition 10 seconds after starting the program.

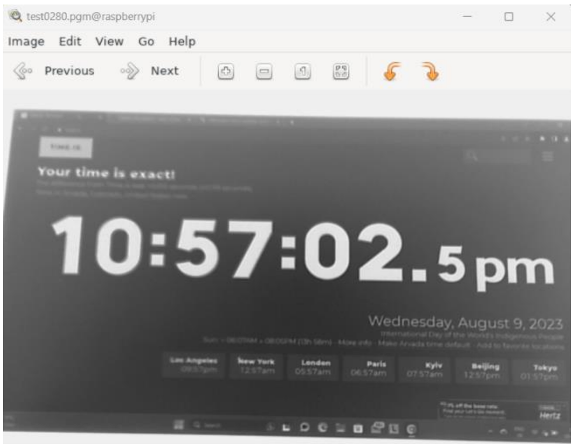


Figure 16: 280th frame opened with the command eom test0280.pgm. The time elapsed from the 100th frame to 280th frame is 18 seconds as observed.

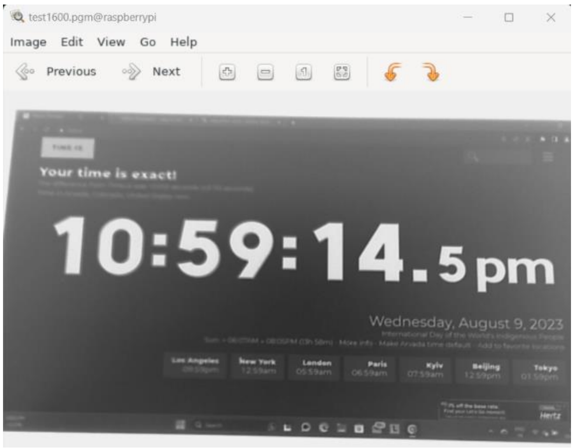
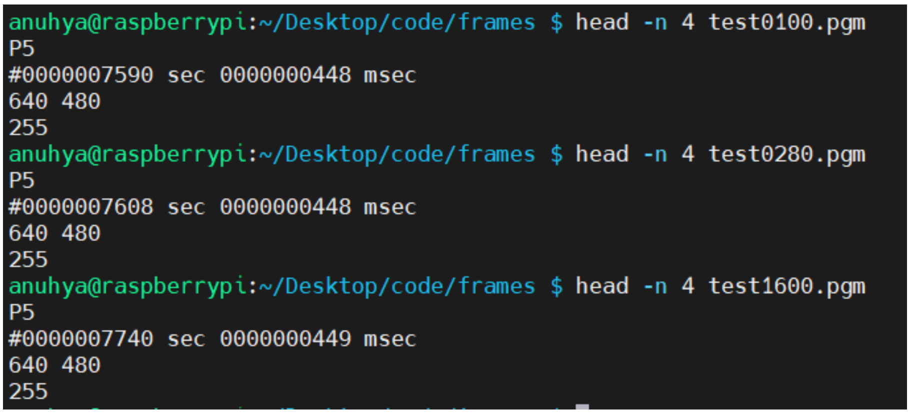


Figure 17: 1600th frame opened with the command eom test1600.pgm. The time elapsed from the 100th frame to the 1600th frame is 2.5 minutes as observed.

There is 18 seconds time difference between the 100th frame and 280th frame. Thus, the program is functional for 10Hz acquisition rate. This is also verified with the header information stored in each file. This is shown in the figure below.



```
anuhya@raspberrypi:~/Desktop/code/frames $ head -n 4 test0100.pgm
P5
#0000007590 sec 0000000448 msec
640 480
255
anuhya@raspberrypi:~/Desktop/code/frames $ head -n 4 test0280.pgm
P5
#0000007608 sec 0000000448 msec
640 480
255
anuhya@raspberrypi:~/Desktop/code/frames $ head -n 4 test1600.pgm
P5
#0000007740 sec 0000000449 msec
640 480
255
```

Conclusion

With this project, the design and execution of a soft real-time system is thoroughly observed. Soft real-time services are often found in application like multimedia streaming, and web services, where meeting response deadlines is crucial but not critical to avoid catastrophic consequences. The board, Raspberry Pi is capable of capturing images at frame rates of 1Hz and 10Hz when integrated with the Webam, Logitech C270. With frame transformation features like sharpening and negative image, the project has applications in object detection, image analysis to observe medical scans, and computer vision. POSIX threads and SCHED_FIFO scheduling policy are valuable tools for developing such soft real-time services and enable implementation of Rate Monotonic policy. With through analysis of task deadlines and jitter experienced by each service, the system is made to be more deterministic.

References

[1] Siewert, Sam & Pratt, John. (2015). Real-Time Embedded Components and Systems with Linux and RTOS.
[2] S. Siewert, "IO latency hiding in pipelined architectures," 2005 IEEE Region 5 and IEEE Denver Section Technical, Professional and Student Development Workshop, Boulder, CO, USA, 2005, pp. 39-45, doi: 10.1109/TPSD.2005.1614345.
[3] C. L. Liu and James W. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. J. ACM 20, 1 (Jan. 1973), 46–61.
[4] Scheduling Point and completion test code referenced from RTES files written by Professor Sam Siewert.
[5] Reghenzani, Federico & Massari, Giuseppe & Fornaciari, William. (2019). The real-time linux kernel: A survey on Preempt_RT. ACM Computing Surveys. 52. 1-36. 10.1145/3297714.

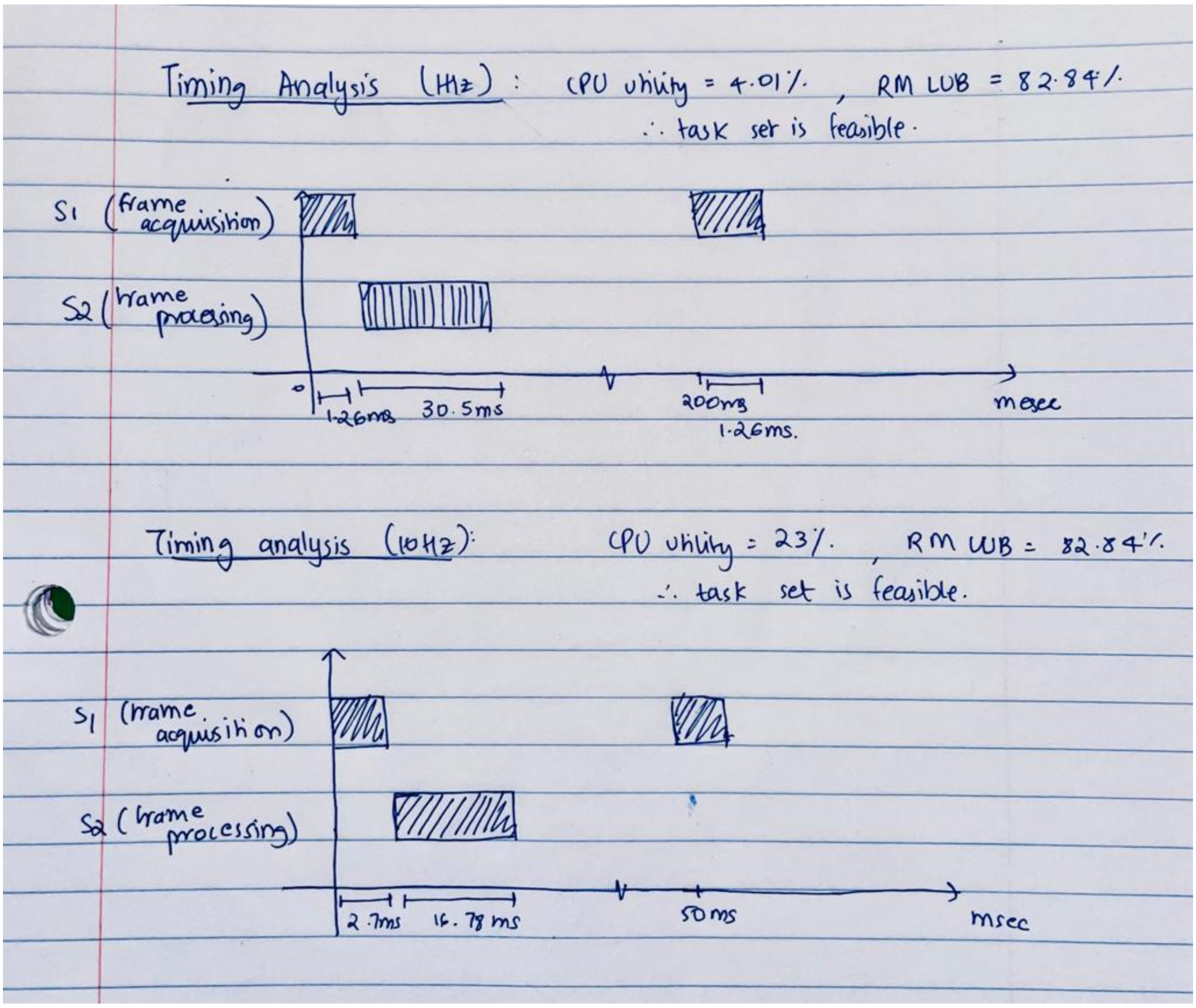
Acknowledgements

I am extremely grateful to Professor Sam Siewart for his support and guidance throughout the course. His efforts in creating this course and imparting knowledge are immensely commendable. Further, I am also thankful to our teaching assistants, Daanish Shariff, Chinmay Shalawadi and Swastik for their responsiveness and patience in clarifying our doubts.

Appendix:

1. **Source code and make file:** 'capture.c' in the folder **'code'** in the folder '**kuraparthy_final_project_code'** in 'kuraparthy_project_material'.
2. **Jitter analysis for 1 Hz:** Excel sheet '**RMA_1hz_wtransform'** in the folder '**jitter analysis**' contains jitter analysis of all services for the 1 Hz frame rate.
3. **Jitter analysis for 10 Hz:** Excel sheet '**RMA_10hz_wtransform'** in the folder '**jitter analysis**' contains jitter analysis of all services for the 1 Hz frame rate.
4. **Captured Frames for 1Hz:** The folder '**DEMO_1hz_wtransform.zip'** contains all the 1800 captured frames with the frame transformation enabled. It can be found in the link: https://drive.google.com/drive/folders/1ZUtYysp-TFOYojI4X-BT7Y4oyymrt11C?usp=drive_link
5. **Captured Frames for 10Hz:** The folder '**DEMO_10hz_wtransform.zip'** contains all the 1800 captured frames with the frame transformation enabled. It can be found in the link: https://drive.google.com/drive/folders/1ZUtYysp-TFOYojI4X-BT7Y4oyymrt11C?usp=drive_link
6. **Hand-drawn timing analysis for both the system architectures (1Hz and 10Hz)-** Page 19.
7. **Monotonicity of Request periods for the services (1Hz and 10Hz)-** Page 19,20.

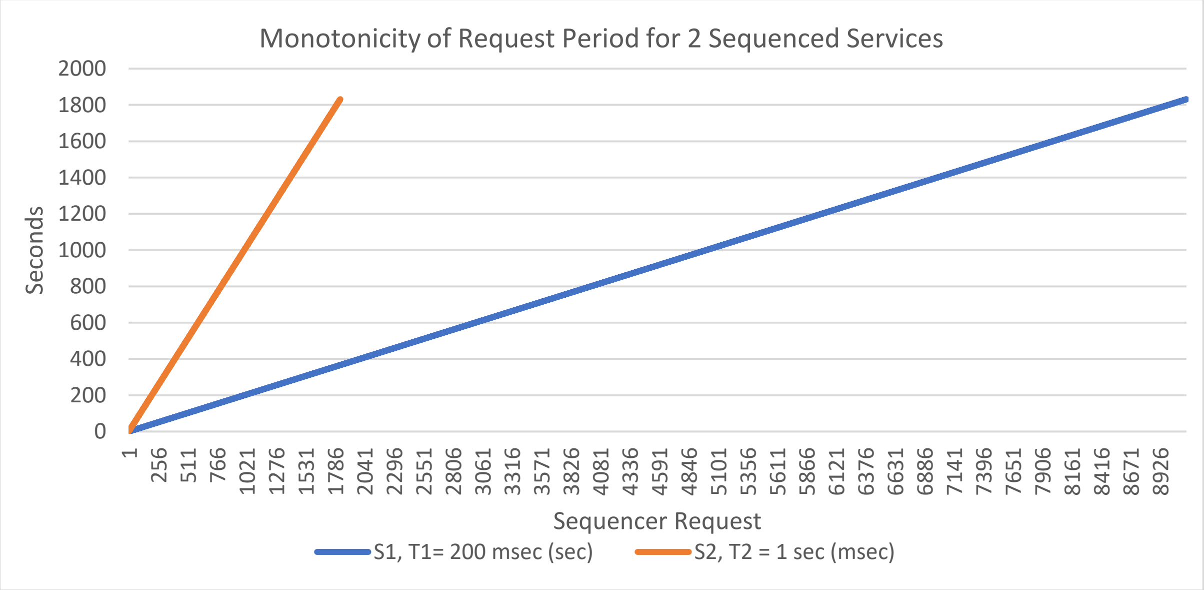
Hand-drawn Timing Analysis:



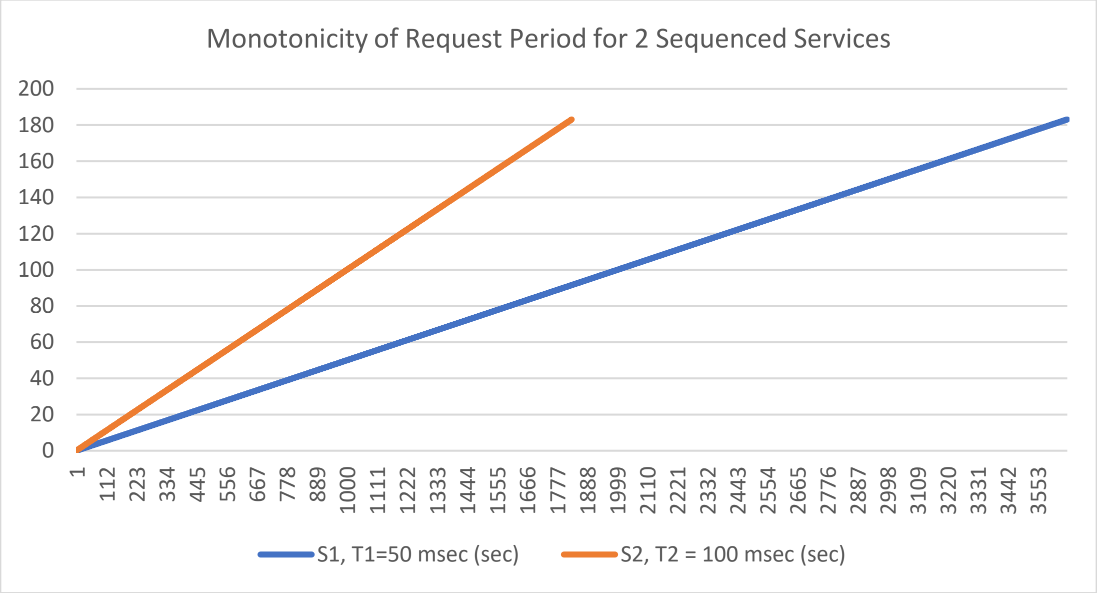
Timing Analysis (1Hz) : CPU utility = 4.01%, RM LUB = 82.84%.
∴ task set is feasible.

Timing analysis (10Hz): CPU utility = 23%, RM LUB = 82.84%.
∴ task set is feasible.

Monotonicity of Request periods for 2 services (1Hz frame rate)



Monotonicity of Request periods for 2 services (10Hz frame rate)


Monotonicity of Request Period for 2 Sequenced Services
S1, T1=50 msec (sec)
S2, T2 = 100 msec (sec)