ECEN 5623 – Real-Time Embedded System

Final Project Report

Exercise #6 – Real-Time Software Systems

Project & Report by -

Deepak E Kapure

University of Colorado Boulder

Department of Electrical, Computer & Energy Engineering

Under the Guidance of:

Prof. Sam Siewert

Table of Content

Introduction

Real-time embedded systems are crucial in applications where precise timing and reliability are essential. In this project, I designed and implemented a real-time image capture system using a Raspberry Pi and a Logitech C270 camera. The primary goal was to capture frames at a consistent rate of 1 frame at 1 Hz and 10 Hz to ensure there are no skips or blurs. This performance was tested by capturing the movement of the second hand of an analog clock.

To achieve this, I leveraged the Video4Linux2 (V4L2) driver, which facilitated the connection between the camera and the Raspberry Pi. The V4L2 driver was instrumental in ensuring clear and consistent image capture. Each frame captured included timestamps and system information to verify the accuracy of the timing, a critical requirement for applications such as surveillance and scientific observation where precise timing and clear imagery are imperative.

Synchronization techniques were employed to maintain reliable frame capture over extended periods. These techniques ensured that the frames were captured consistently, without any loss or degradation in quality, demonstrating the system's robustness in real-time applications. The project not only highlights the practical application of real-time embedded systems but also provides a solid foundation for future improvements and uses.

This report provides a detailed overview of the entire project, covering the hardware and software design, system testing, and analysis of the results. The comprehensive coverage ensures a complete understanding of the project's scope, methodologies, and outcomes, offering valuable insights for future endeavors in the field of real-time embedded systems.

Major Functional Capability Requirements

List of the major functional capability requirements for my real-time image capture system:

1. **Run at least 2 services on a single AMP core**:
   This requirement is realized to demonstrate the real time system and is implemented by running the **Differencing and marking service S3** alongside the **frame selection service S4** on core 2 of the project.

   The service S3 runs at 20Hz and performs frame comparison and thresholding to mark the useful frames. S4 is run at 1 or 10Hz, depending upon the mode set, to pick the right timestamp images to be dumped to memory.

2. **Dump PPM images with appropriate timestamps**:
   This is realized in the **write-back service S5** which is a best effort service running independently on core3.
   It records the unique timestamps taken during the capture stage and dumps them to memory as .ppm files.

3. **Capture unique "second hand" locations (60 per 1 minute cycle +1 to return to starting state) without glitches**:
   This requirement is fulfilled by the **differencing and marking service S3**, which checks for unique frames with no glitches. This is done by performing a progressive differencing of images captured and measuring how many of these have a pixel change significantly higher, for them to be discarded.
   Also, the **frame selection service S4** makes sure that the successive images dumped are all carefully selected at the required rate from the circular buffer.

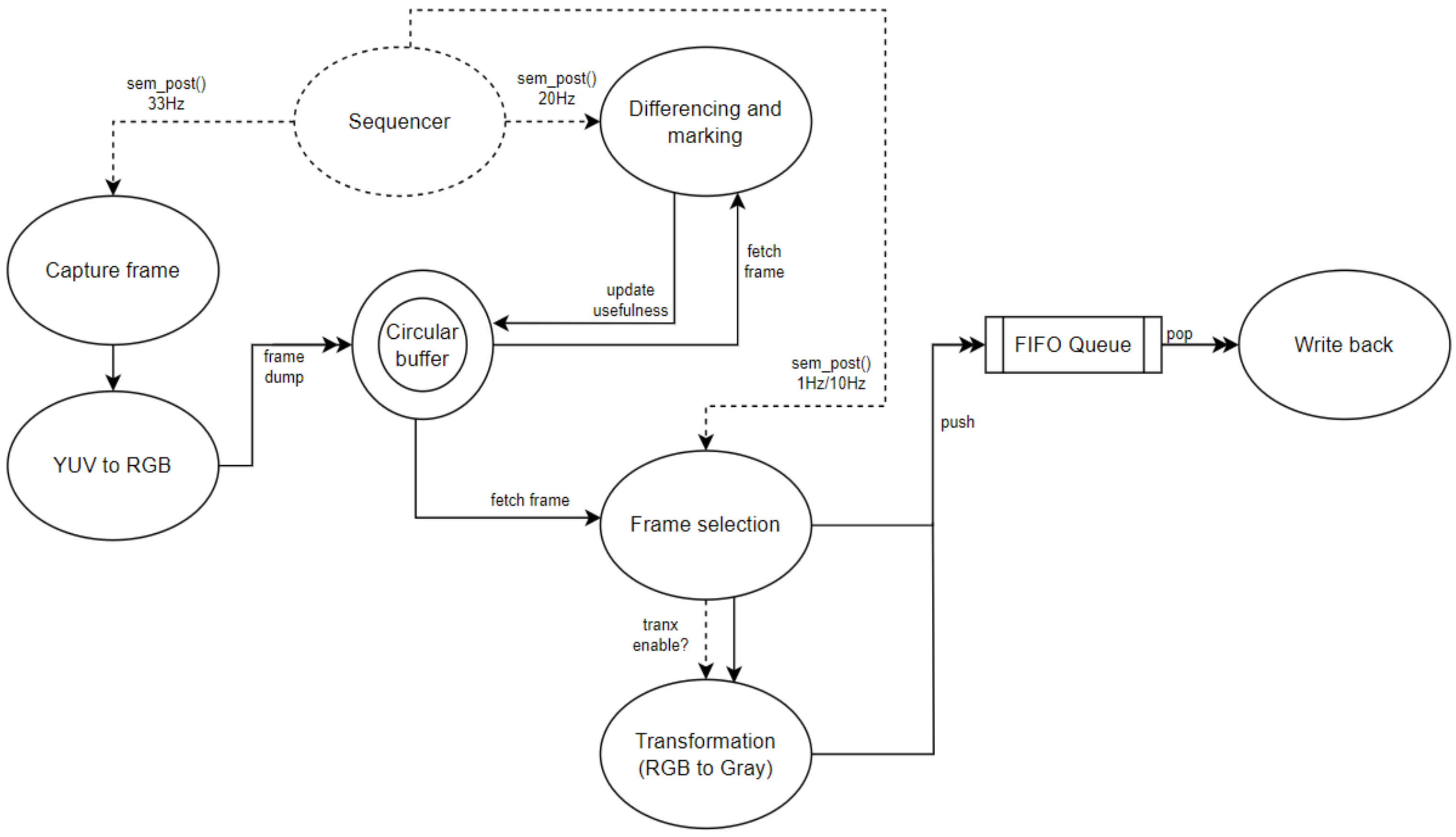
4. **Add a transformation on the frames which could be enabled/disabled**:

An **RGB to grayscale transformation** is added as an extension to the frame selection service S4, which converts the RGB to grayscale before pushing it to the FIFO queue.

5. **Cheddar analysis:**
   To validate and verify the if the timing requirements are met, with rate monotonic feasibility and schedulable service executions

6. **(Stretch goal) Capture unique "tenths of a second digits" (10 per second 0…9 +1 to return to 0) without glitches**:
   This is attempted to be achieved in the similar way as the requirement 3 above.
   This requirement is fulfilled by the **differencing and marking service S3**, which checks for unique frames with no glitches. This is done by performing a progressive differencing of images captured and measuring how many of these have a pixel change significantly higher, for them to be discarded.
   Also, the **frame selection service S4** makes sure that the successive images dumped are all carefully selected at the required rate from the circular buffer.

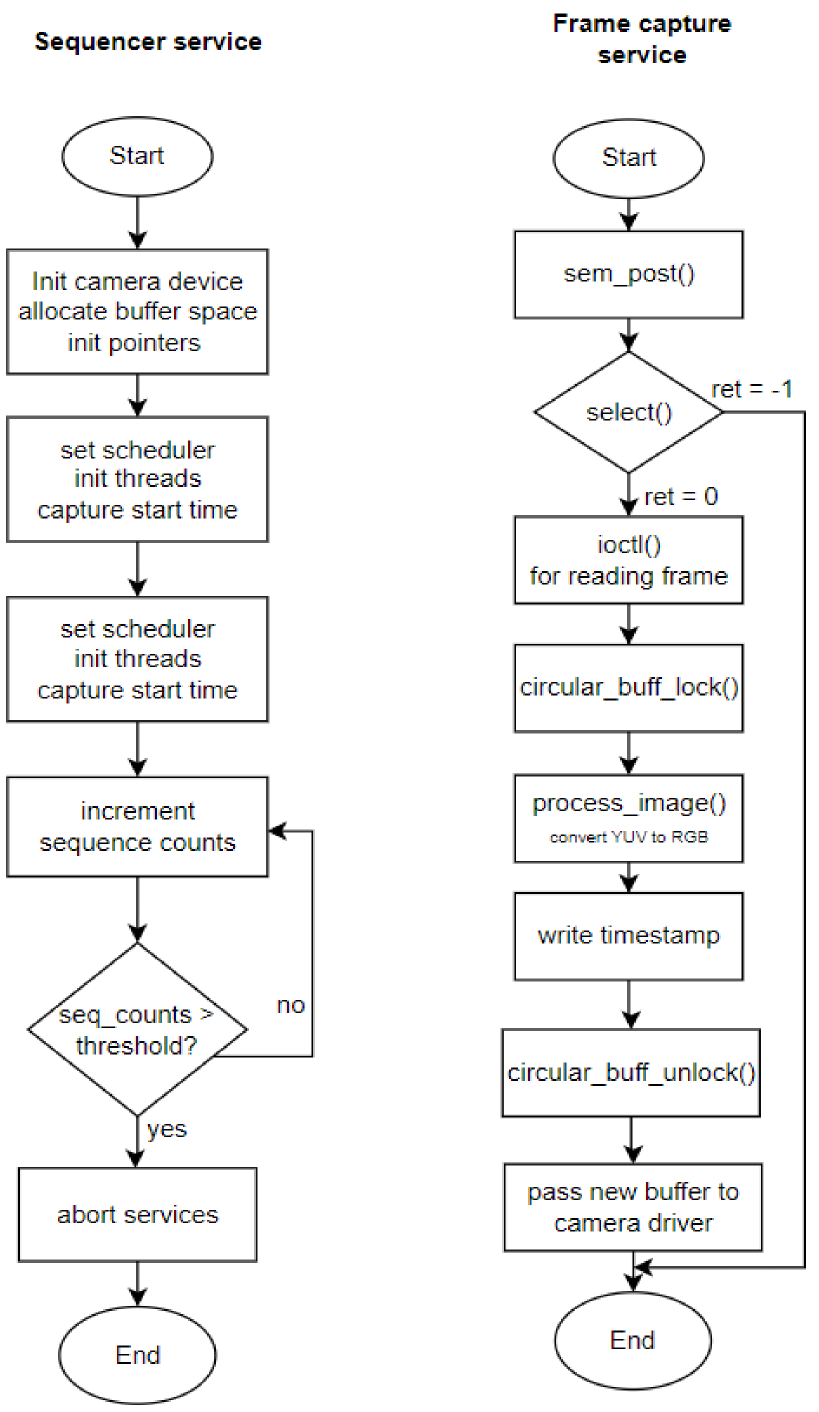
SYSTEM ARCHITECTURE



The system is divided into 5 services:

1. Sequencer service S1
2. Frame capture service S2
3. Differencing and marking service S3
4. Frame selection service S4
5. Write back service S5

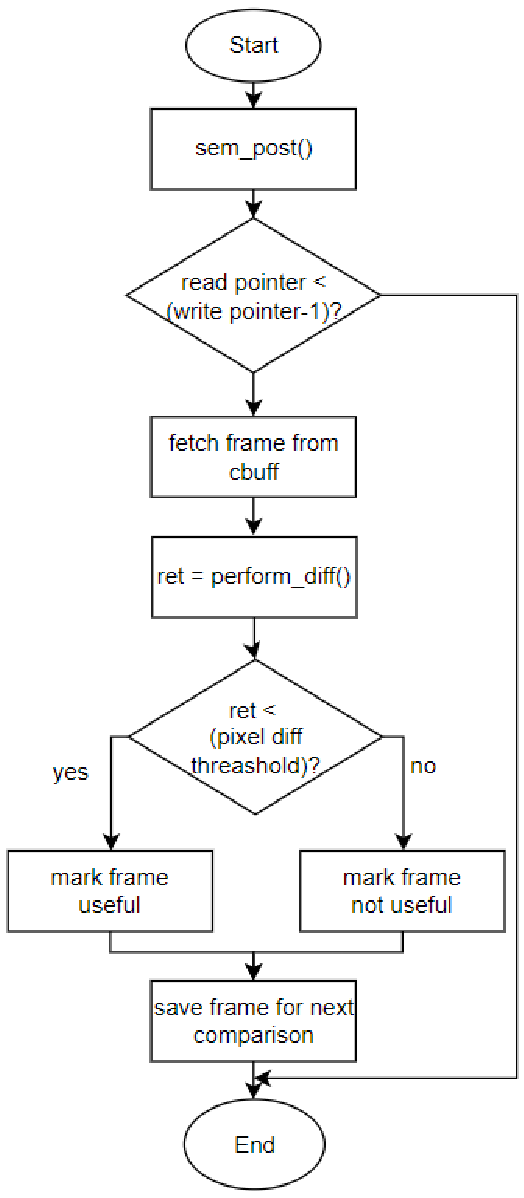
The services are segregated int cores 1, 2 and 3 as core 0 runs the kernel and the design wanted to avoid any possible conflict with the kernel. All the services have SCHED_FIFO configured except S5 which is best effort. The camera device is connected to /dev/video0 device and the application interfaces with the UVC camera driver for reading frames.


Sequencer service
Start
Init camera device allocate buffer space init pointers
set scheduler init threads capture start time
set scheduler init threads capture start time
increment sequence counts
seq_counts > threshold?
no
yes
abort services
End
Frame capture service
Start
sem_post()
select()
ret = -1
ret = 0
ioctl() for reading frame
circular_buff_lock()
process_image()
convert YUV to RGB
write timestamp
circular_buff_unlock()
pass new buffer to camera driver
End

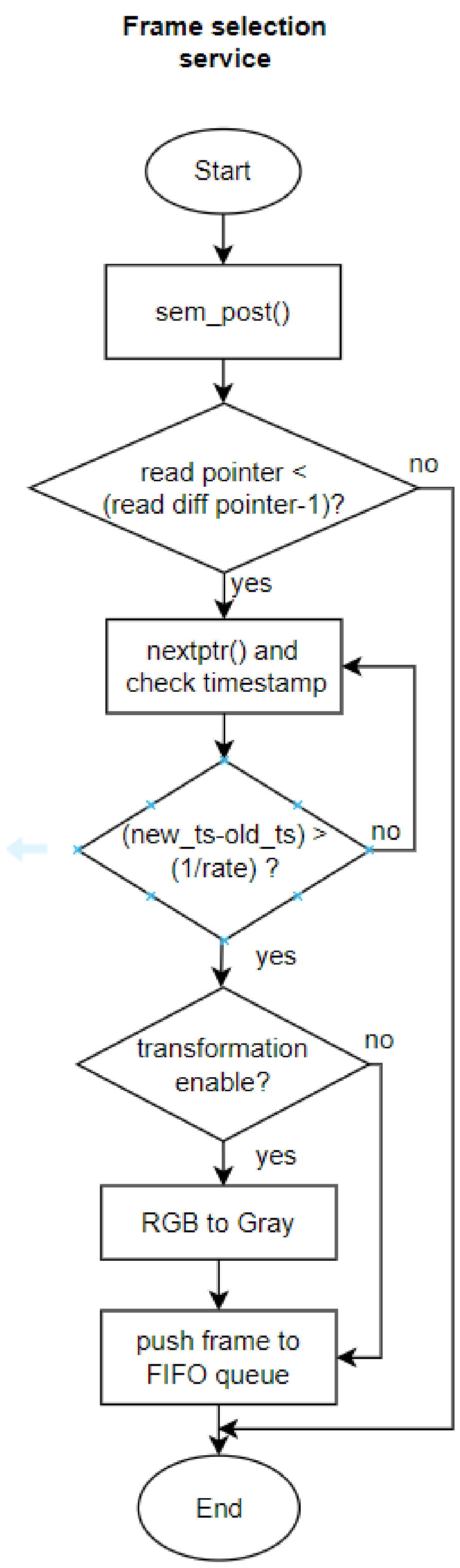
The service S1 runs on core 1 and is responsible for providing the real time execution triggers for the rest of the services except S5, which is a best effort service. It runs an interval timer to measure ticks at 100Hz rate. The service also initializes the semaphores used to dispatch services at their scheduled times. There are 3 rates which the sequencer triggers services at – 33Hz for frame capture S2, 20Hz for differencing and marking S3 and 1Hz or 10Hz for the frame selection service S4. It also provides abort signals to the other service threads to exit.

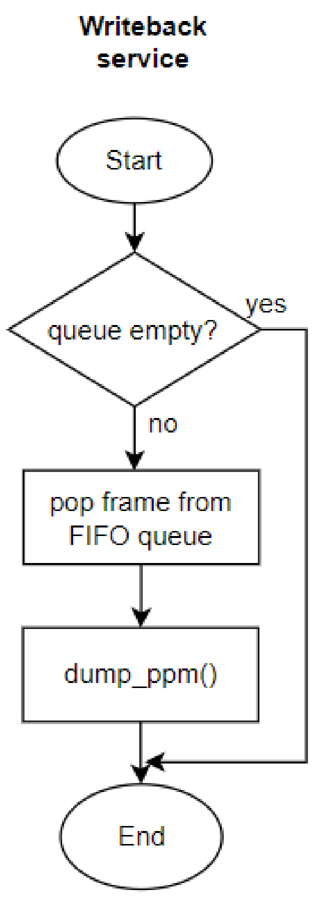
Service S2 also runs on core 1 and shares the core with S1 on a RM real time basis. This service is responsible for communicating with the camera driver and capturing frames at triggered intervals of 33Hz. This rate is set in accordance with the camera's maximum frame rate. The service interfaces using the ioctl system call for the driver and fills the v4l2 buffer assigned to it. It further performs the conversion to RGB from YUV as well and then finally pushes the data to the circular buffer with the appropriate timestamp. This operation is designed to be non-blocking type, except for the SGL accesses during runtime.

Differencing and marking service



Service S3 runs on core 2 and does the differencing and marking of frames suitable to be dumped if selected. The service runs at 20Hz and tries to access and mark frames that have differences in pixels for consecutive frames below a preset threshold. The service runs till the circular buffer has frames or processes a minimum number of frames, whichever is less. Marking writes the usefulness parameter of the frame's entry in the buffer.


Frame selection service
Start
sem_post()
read pointer < (read diff pointer-1)?
no
yes
nextptr() and check timestamp
(new_ts-old_ts) > (1/rate) ?
no
yes
transformation enable?
no
yes
RGB to Gray
push frame to FIFO queue
End


Writeback service
Start
queue empty?
yes
no
pop frame from FIFO queue
dump_ppm()
End

The service S4 also runs on core 2 and shares the core with S3 on an RM real time basis. It is responsible for selecting the previously marked useful frames at the triggered rate of 1Hz or 10Hz. Once found, the service pushes the frame to the FIFO queue.

Service 5 runs on core 3 and is the write back service for dumping frames to memory. It is a best-effort service and has a simple task of fetching frames from FIFO queue and writing back to memory.

Apart from the above, there are 2 essential data structures used. They are the circular buffer and the FIFO queue. The circular buffer is used to decouple the frame capture with the rest of the services. It has a depth of 90 frames/entries and hence gives sufficient time to dump and process frames for the other services. This buffer has one write and two read pointers, used by S1, S2 and S3 respectively. The FIFO queue is used for storing the frames ready to be written to memory by S5 and decouples the costly write back service S5 from the rest of the system.

Real-Time Service Execution, Scheduling and Timings

Following are the real-time requirements for each service

1. Sequencer service S1:

   Sequencer shares the core with S2 and has a higher priority than S2. Being a service that triggers the other, the design decision of having request rate of at least 3 times than the next highest service. Hence, by design, request rate of 100Hz (3*33Hz) was made and thus the completion time required is under 1ms.

   C1 = 0.5 msec

   T1 = D1 = 1msec

2. Frame capture service S2

   Frame capture has a 33Hz refresh rate, which is set in accordance with the highest capture rate for the camera.

   C2 = 28 msec

   T2 = D2 = 33.3 msec

3. Differencing and marking service S3

   The design request rate for S3 was decided based on the highest rate required by the frame selection service S4 and S2 as well. S2 should have a rate higher than 10Hz (highest rate for S4) and less than 33Hz (S2's rate) for maximum efficiency. Thus, a request rate of 20Hz was selected

   C3 = 20 msec

   T3 = D3 = 50 msec

4. <u>Frame selection service S4</u>

   Frame selection is based on the synchronome rate requested, i.e. 10Hz.

   C4 = 40 msec

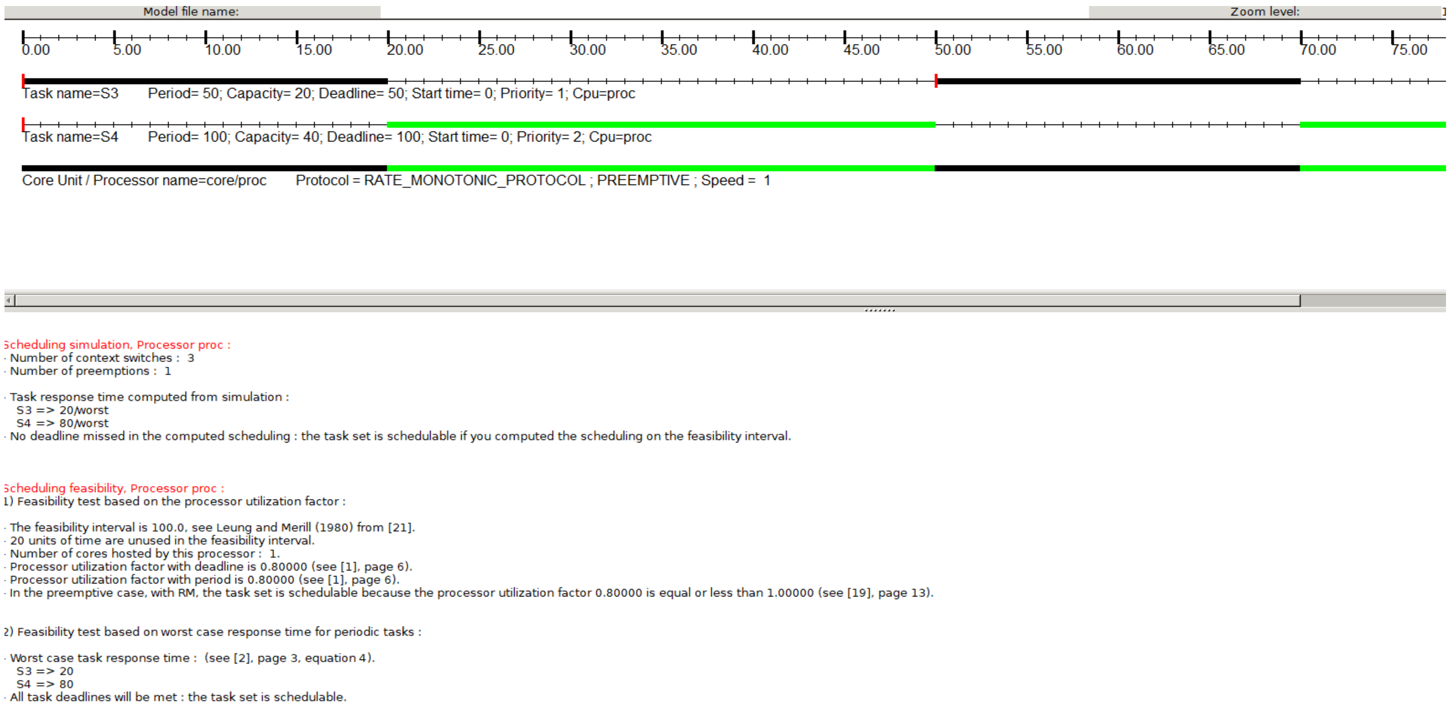
   T4 = D4 = 100msec

5. <u>Write back service S5</u>

   Write back has no real-time requirements as it is a best effort service.

Cheddar Analysis

Cheddar analysis for core 2 which has S3 and S4 running was analyzed in cheddar for RM policy. The solution obtained was schedulable and feasible with utilization factor 80% and a LUB of 82.4% for 2 services.



Task name=S3 Period= 50; Capacity= 20; Deadline= 50; Start time= 0; Priority= 1; Cpu=proc

Task name=S4 Period= 100; Capacity= 40; Deadline= 100; Start time= 0; Priority= 2; Cpu=proc

Core Unit / Processor name=core/proc Protocol = RATE_MONOTONIC_PROTOCOL ; PREEMPTIVE ; Speed = 1

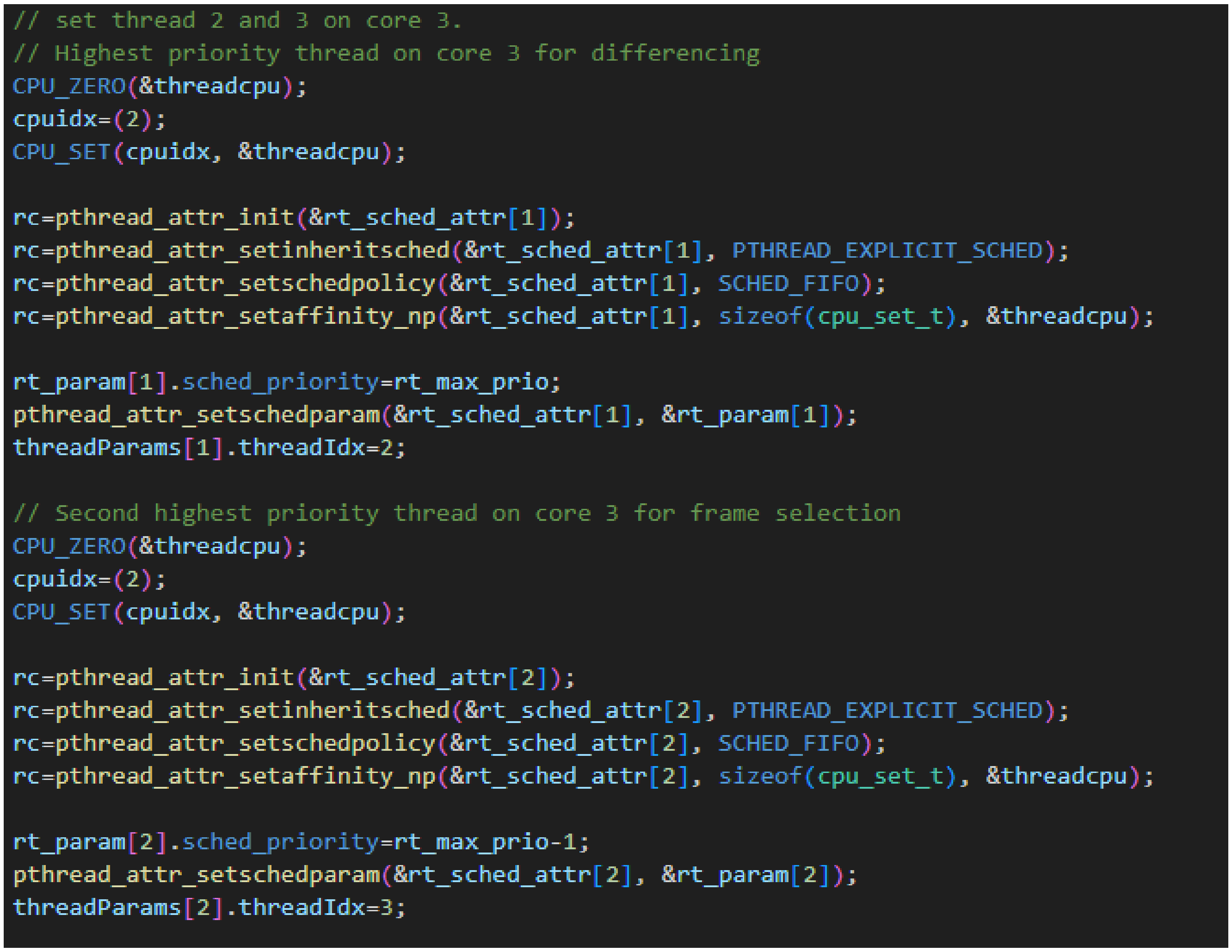
```
Scheduling simulation, Processor proc :
- Number of context switches :  3
- Number of preemptions :  1

- Task response time computed from simulation :
   S3 => 20/worst
   S4 => 80/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 proc :
1) Feasibility test based on the processor utilization factor :

- The feasibility interval is 100.0, see Leung and Merill (1980) from [21].
- 20 units of time are unused in the feasibility interval.
- Number of cores hosted by this processor :  1.
- Processor utilization factor with deadline is 0.80000 (see [1], page 6).
- Processor utilization factor with period is 0.80000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.80000 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).
   S3 => 20
   S4 => 80
- All task deadlines will be met : the task set is schedulable.
```

The safety margins considering the variation in completion time are consistent with the expected requirements. There could be an increase in the utilization for S4 with increasing the C4 for S4.

ANALYSIS OF SYSTEM

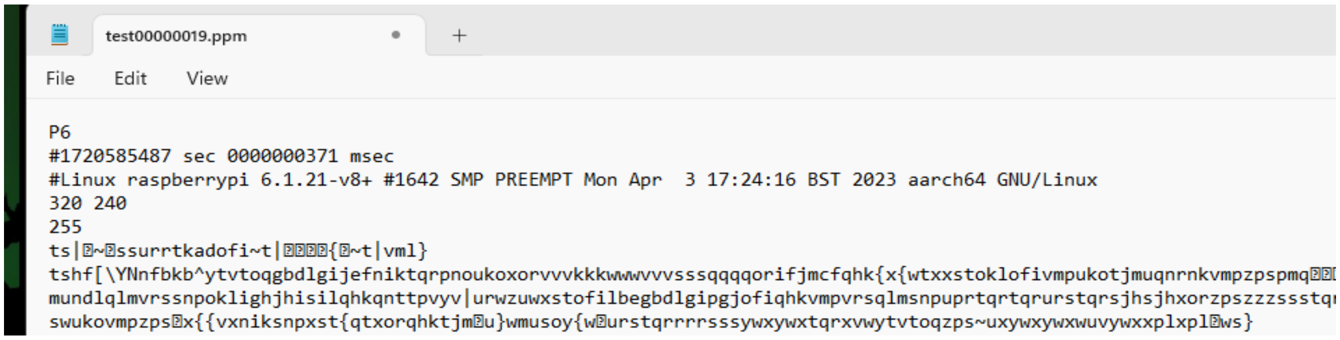
Proof-of-concept

1. Core 2 hosts 2 real-time services



```
// set thread 2 and 3 on core 3.
// Highest priority thread on core 3 for differencing
CPU_ZERO(&threadcpu);
cpuidx=(2);
CPU_SET(cpuidx, &threadcpu);

rc=pthread_attr_init(&rt_sched_attr[1]);
rc=pthread_attr_setinheritsched(&rt_sched_attr[1], PTHREAD_EXPLICIT_SCHED);
rc=pthread_attr_setschedpolicy(&rt_sched_attr[1], SCHED_FIFO);
rc=pthread_attr_setaffinity_np(&rt_sched_attr[1], sizeof(cpu_set_t), &threadcpu);

rt_param[1].sched_priority=rt_max_prio;
pthread_attr_setschedparam(&rt_sched_attr[1], &rt_param[1]);
threadParams[1].threadIdx=2;

// Second highest priority thread on core 3 for frame selection
CPU_ZERO(&threadcpu);
cpuidx=(2);
CPU_SET(cpuidx, &threadcpu);

rc=pthread_attr_init(&rt_sched_attr[2]);
rc=pthread_attr_setinheritsched(&rt_sched_attr[2], PTHREAD_EXPLICIT_SCHED);
rc=pthread_attr_setschedpolicy(&rt_sched_attr[2], SCHED_FIFO);
rc=pthread_attr_setaffinity_np(&rt_sched_attr[2], sizeof(cpu_set_t), &threadcpu);

rt_param[2].sched_priority=rt_max_prio-1;
pthread_attr_setschedparam(&rt_sched_attr[2], &rt_param[2]);
threadParams[2].threadIdx=3;
```

2. Ppm timestamps



test00000019.ppm

File Edit View

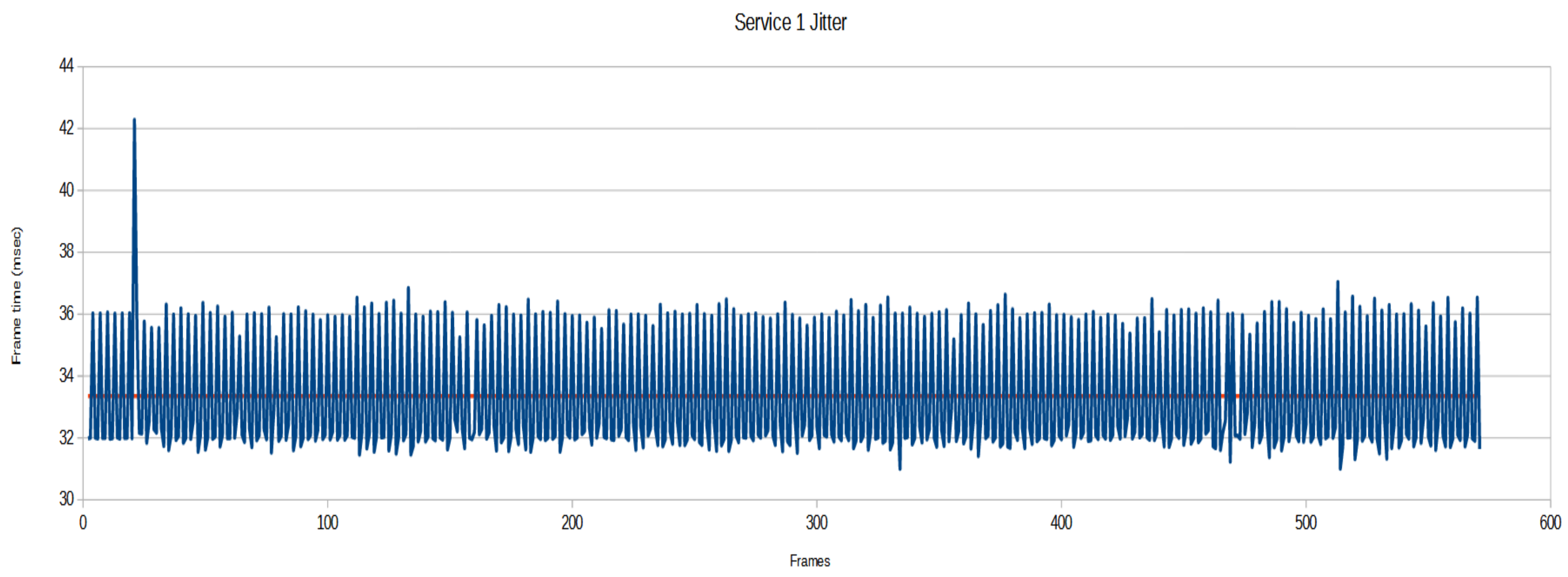
```
P6
#1720585487 sec 0000000371 msec
#Linux raspberrypi 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64 GNU/Linux
320 240
255
ts|□~□ssurrtkadofi~t|□□□□{□~t|vml}
tshf[\YNnfbkb^ytvtoqgbdlgijefniktqrpnoukoxorvvvkkkwwwvvvsssqqqorifjmcfqhk{x{wtxxstoklofivmpukotjmuqnrnkvmpzpspmq□□
mundlqlmvrssnpoklighjhisilqhkqnttpvyv|urwzuwxstofilbegbdlgipgjofiqhkvmpvrsqlmsnpuprtqrtqrurstqrsjhsjhxorzpszzzssstq
swukovmpzps□x{{vxniksnpxst{qtxorqhktjm□u}wmusoy{w□urstqrrrrsssywxywxtqrxvwytvtoqzps~uxywxywxwuvywxxplxpl□ws}
```

3. Jitter and WCET

Service 2 - Timing analysis:

Average - 33.349033333 msec WCET – 42.281 msec

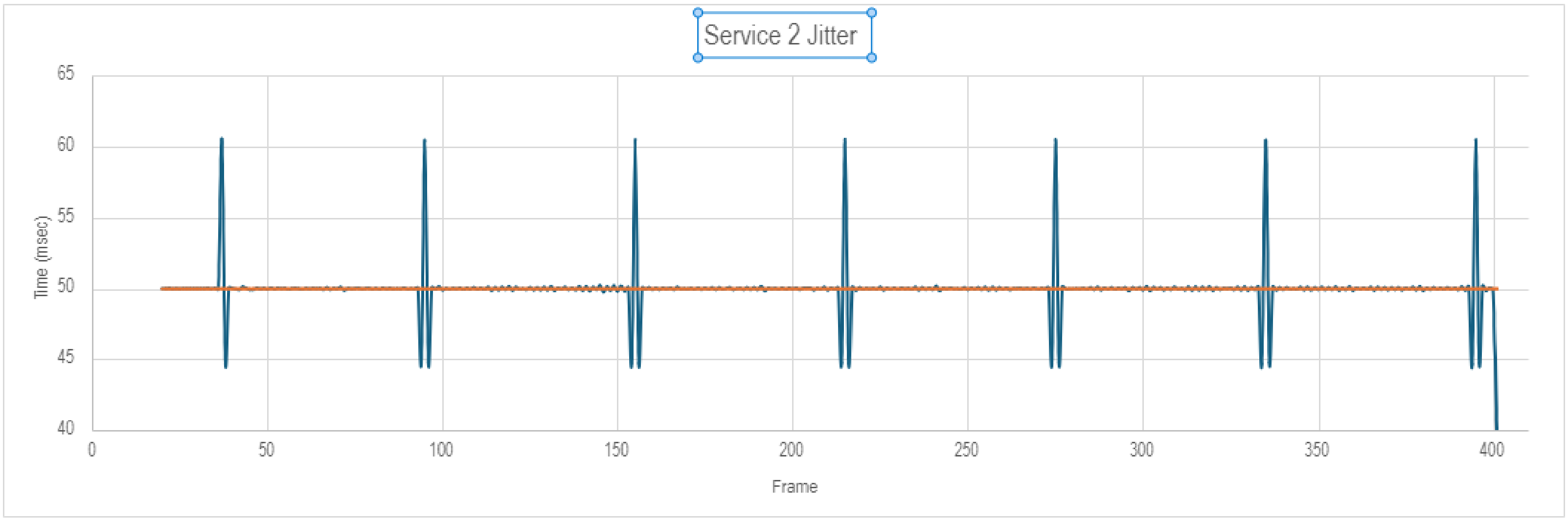
Jitter analysis: Expected – 30.33 msec (red line below for trend)



Service 3 – Timing analysis:

Average – 49.988 msec WCET – 60.527 msec

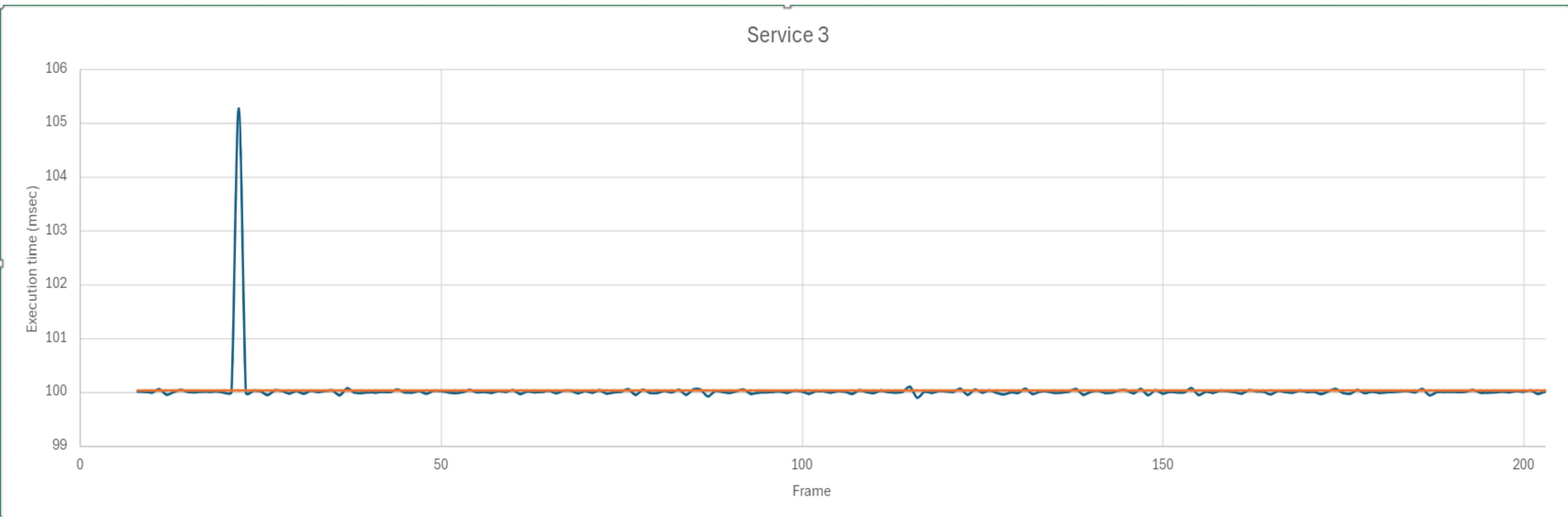
Jitter analysis: Expected: 50msec (orange line below is the trend)



Service 4 – Timing analysis

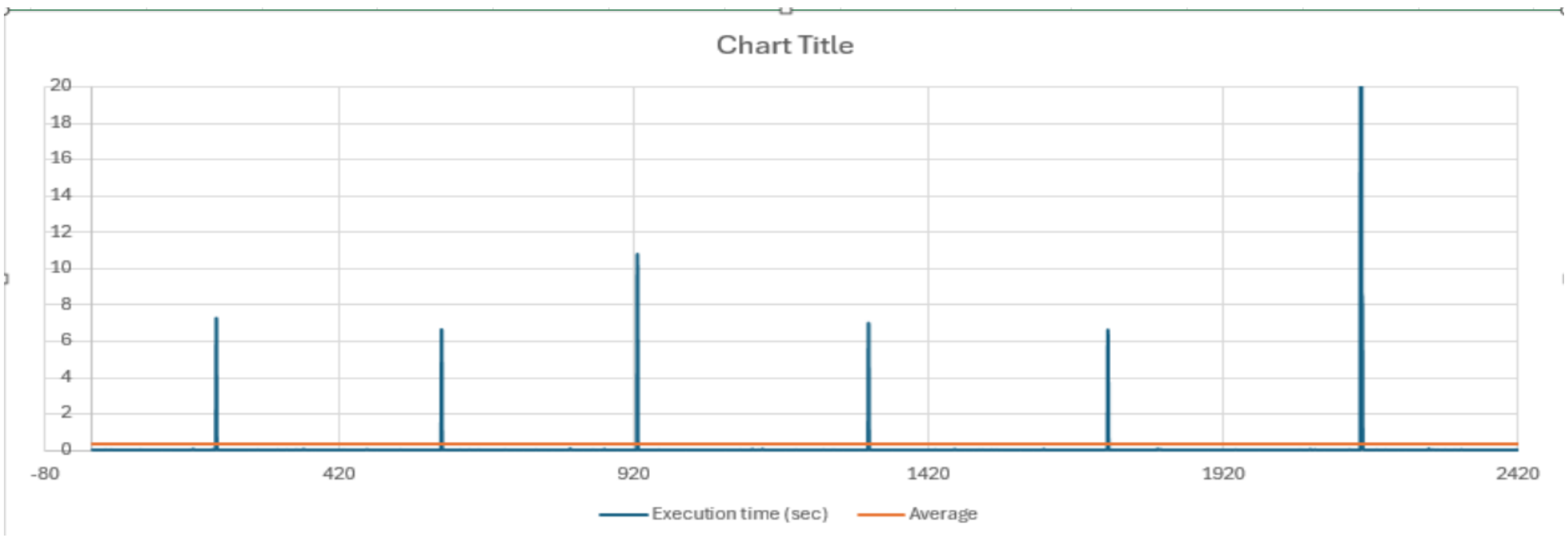
Average - 100.0279 msec WCET –105.272 msec

Jitter analysis: Expected: 100 msec (orange trend line)



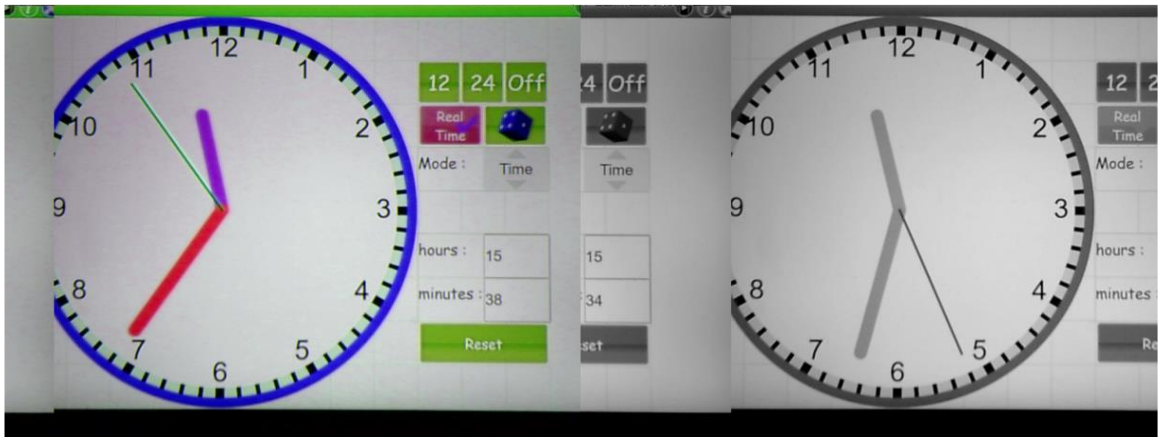
Service 5 – Timing analysis: (best effort service)

Average – 0.3543 msec WCET – 763.73 msec



4. Frames at 1Hz and 10Hz:
   Refer to this repository - frames@1Hz, frames@10Hz

5. Transformation to Gray



Conclusion

This project involved developing a multi-service system for real-time frame processing, leveraging a multicore architecture to efficiently manage tasks and ensure timely processing. The services were carefully designed and scheduled based on Rate Monotonic Scheduling (RMS) principles to meet specific real-time requirements.

Through a comprehensive timing analysis, I established that each service meets its respective deadlines within the constraints of its execution time. Specifically, the worst-case execution times (WCET) for the services. The Sequencer Service (S1) provides real-time execution triggers for the other services at a 100Hz rate, maintaining a completion time under 1ms. It ensured the timely triggering of other services while sharing Core 1 with the Frame Capture Service (S2). S2, operating at a 33Hz rate, communicated with the camera driver to capture frames, convert YUV to RGB, and push data to the circular buffer efficiently, meeting its real-time constraints with a computation time of 25ms within a 30.3ms period.

On Core 2, the Differencing and Marking Service (S3) analyzed frames for differences at a 20Hz rate and marked useful frames within its 50ms period, ensuring frames with significant differences were flagged for further processing. Sharing Core 2 with S3, the Frame Selection Service (S4) operated at a rate of 1Hz or 10Hz, selecting marked frames and pushing them to the FIFO queue, performing its task within a 100ms period.

The Write Back Service, being a best-effort service, wrote frames from the FIFO queue to memory without stringent real-time requirements, ensuring the system's overall efficiency. The circular buffer successfully decoupled frame capture from other services, while the FIFO queue effectively managed frames ready for write-back.

Furthermore, the Cheddar analysis provided further verification of the system's real-time capabilities. Tweaking the time periods of the differencing and frame selection service, I was able to fulfill the RMS policy requirements and obtain deterministic timing diagrams

Overall, the system design based on RMS principles ensured that all real-time services met their deadlines, with S1 providing timely triggers, S2 capturing and converting frames efficiently, S3 processing frames as required, and S4 pushing useful frames for write-back. The best-effort write-back service (S5) complemented the real-time operations, ensuring seamless memory write-back. This project demonstrates a robust real-time frame processing system optimized for performance and reliability on a multicore platform.

References:

[1] Github of Prof Sam Siewert.

[2] draw.io for the Flow charts.

[3] Cheddar for timing analysis.