ECEN 5623 - Real-Time Embedded Systems
Final Project Proposal
7/24/2020
Brian Ibeling and Josh Malburg

Project Overview

The goal of this project is to use a camera to observe a periodic physical process (a clock), synchronize with that process, controlling jitter and drift, and experience no glitches (frame blurring, skipping, or missing) over a 30 minute period. This needs to be observed at a minimum periodic rate of 1 Hz, and produce a predictable response over a long period of time. A stretch goal will be to observe the changes of a clock at a 10 Hz rate, showing the subsecond changes on a stopwatch over a 3 minute period. In either scenario (1 Hz monitoring for 30 minutes vs 10 Hz monitoring for 90 seconds), a total of 1800 frames will be captured, for which there should be no two frames which are the same or have any frames with blurring or distortion.

The goal of this project is characterize, analyze, and verify a real-time system, as well as to simulate a mission critical real-time system. A glitch or failure could result in loss of the asset and/or potentially loss of human life, so we need to be able to verify our software will work on a consistent, deterministic, and predictable manner.

Project Requirements

The following list provides requirements for completing the standard extended lab as defined here.

1. The system shall monitor the physical process of an Analog Clock or Stopwatch.
2. The system shall receive frame images from a camera at a rate of approximately 30 Hz.
3. The system shall acquire individual frames at a minimum resolution of 640 x 480.
4. The system shall acquire frames at a minimum rate of 20 Hz.
5. The system shall add a timestamp to each acquired frame in the frame header or onto the image directly.
6. The system shall add the target platform name to each acquired frame in the frame header or onto the image directly.
7. The system shall save frames in a PPM frame format.
8. The system shall save frames to persistent storage memory at a minimum rate of 1 Hz.
9. The system shall save frames to persistent storage memory at a maximum rate of 10 Hz.
10. The system shall operate for a maximum duration of 30 minutes when saving frames to persistent memory at a 1 Hz rate.
11. The system shall operate for a maximum duration of 1.5 minutes when saving frames to persistent memory at a 10 Hz rate.
12. The system shall only save unique frames (no duplicate frames).

13. The system shall only save frames without blurring or distortion.
14. The system shall save a frame for every unique position of the Analog Clock or Stopwatch.
15. The system shall have less than 1 second of error over a 30 minute duration.
16. The system shall capture logging data to determine system jitter and drift.
17. The system shall support an additional image processing feature for edge detection.
18. The system shall add hough circles/line transforms to find/draw the hands and border of the clock.
19. The system shall support the ability to enable or disable image processing features via a runtime flag passed to the application when it is started.

Project Design and System Analysis

To meet the project requirements as listed above, the following hardware and software services will be utilized.

System Diagrams

Figure 1 below shows the hardware that will be utilized for the Standard Final Project, as well as the interfaces connecting the various components. Capture of frames from a standard webcam (C270 or better) will be captured, stored, and analyzed on the Raspberry Pi 3B+ (RPi), with system logging data also captured for analysis of system performance as well as timing jitter and drift. Frames and logging data will then be passed to a remote server (desktop computer) for further analysis.

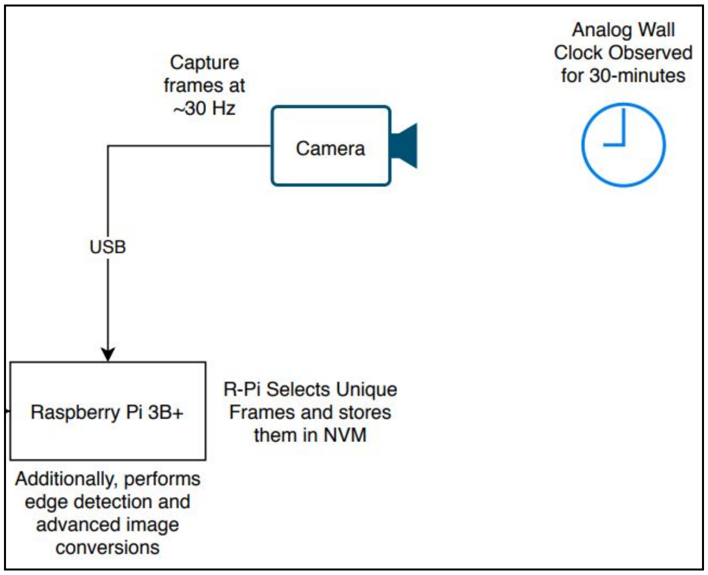

Analog Wall Clock Observed for 30-minutes
Capture frames at ~30 Hz
Camera
USB
Raspberry Pi 3B+
R-Pi Selects Unique Frames and stores them in NVM
Additionally, performs edge detection and advanced image conversions

Figure 1: Key Hardware Elements for Standard Final Project

Figure 2 below shows the Software-Hardware Stackup for the project, highlighting the various software services that will be developed, the utilized software libraries (OpenCV) and Linux drivers, and hardware used. The use of OpenCV, a widely used machine vision tool, will greatly reduce complexity of developing image processing functionality with the limited schedule remaining (see Project Schedule section).

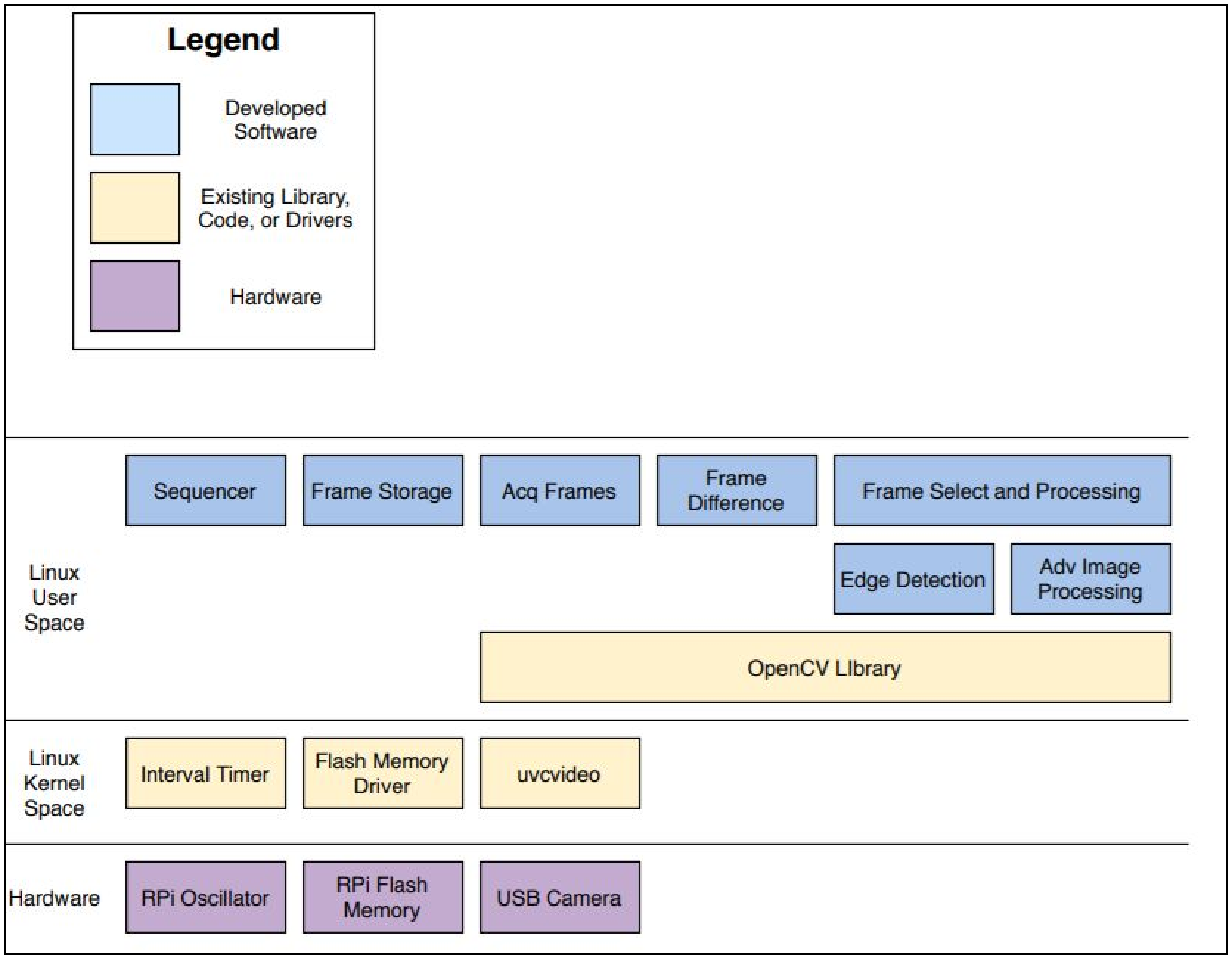


Figure 2: Software-Hardware Stackup for Standard Final Project

Figure three (3) presents a block diagram view of our software process. Our current concept includes four (4) real-time tasks and one best-effort task. A timer will be used to run the sequencer and the sequencer will synchronize all other RT tasks using semaphores. The Frame Acquisition (FA) thread will read 20 frames a second which should be sufficient to capture each tick of a clock's second hand. The FA thread will add frames to a circular buffer that will be periodically processed by the Clock Sync (CS) thread. The CS thead's primary task is to identify frames in which the second hand moved; the thread will threshold the difference between subsequent frames to detect motion of the clock hands. Frames for each unique second hand position will be inserted into the Select message queue (selectQueue). Periodically, as dictated by the sequencer, the Frame Processing thread will read the Select MQ, use Hough transform to color the clock arms and boundary and add the necessary text overlays before adding the enhanced image to the Write message queue (writeQueue). The best-effort Frame Writer thread will block on reads to the write queue, waiting for new frames to write. Since this thread is best effort it will be preemptible by the RT tasks and we need to ensure there's adequate slack time for writes to occur and the queue doesn't fill.

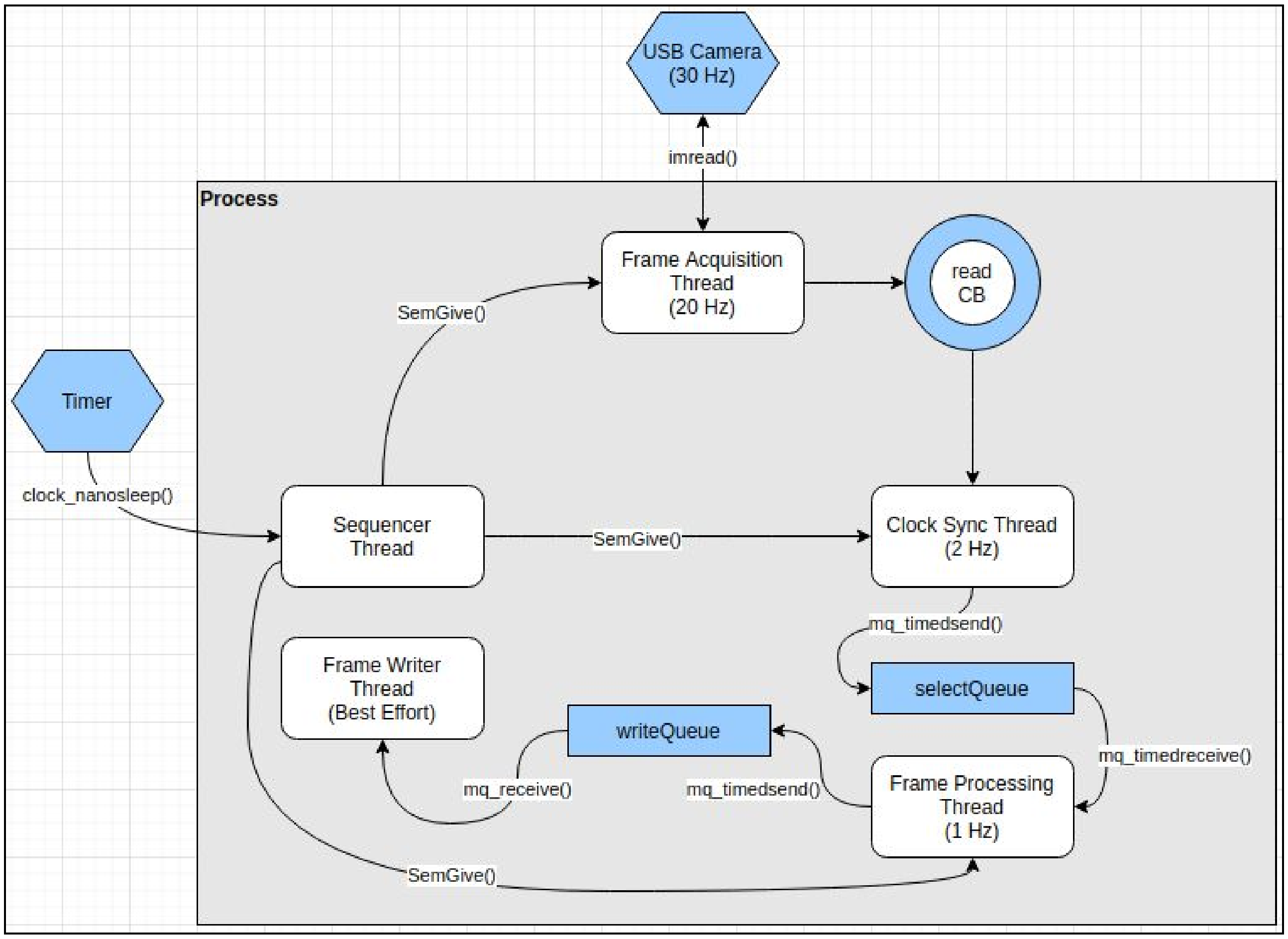


Figure 3: Software Block Diagram

Software Services

The list below provides definitions of all software services that will be implemented to meet the requirements as detailed above. All services will write logging data to syslog to allow for tracing and validation of the system.

1. Sequencer service thread (120 Hz RT task)
    a. High frequency executive which drives everything at a sub-rate of this highest frequency
    b. Oscillator drives an Interval Timer (IT), which based on a comparator sends an interrupt to the sequencer.
    c. Can duplicate the oscillator to avoid a single point of failure of the interrupt timing source (nice to have)
2. Frame Acquisition service thread (20 Hz RT task)
    a. Writes data to Frame Acquire global ring buffer stored in RAM
3. Clock Sync Threshold service thread (2 Hz RT task)
    a. Threshold frame differences to analyzes frames in the ring buffer and find new clock time frames
    b. Add new time frames to selectQueue
4. Frame Processing service thread (1 Hz RT task)
    a. Use an edge enhancement filter and Hough Lines/Circle Transforms to highlight clock hands and border.
    b. Add necessary text overlays
    c. Passes that selected frame to the Frame Storage service thread
5. Frame Storage service thread (1 Hz, 10 Hz)
    a. Should run on its own core, and be implemented as a Best Effort
    b. Likely have some IO involved with writing frame to NVM storage

Thread State Diagrams

We are planning to assign thread priorities according to the Rate Monotonic policy which means the threads from highest to lowest priority will be: Sequencer, Frame Acquisition, Clock Sync, Frame Processing and Frame Storage. All of these threads will implement the state diagram shown below. When executing lower-priority threads are preempted by higher-priority threads they will move into the Ready state. Executing threads that need an unavailable resource (e.g. semaphore or queue) will transition to the pending state and when the resource becomes available the thread moves into the ready queue waiting to be dispatched by the scheduler. The schedule of course dispatches threads in the ready queue by priority. We are planning to either use clock_nanosleep (w/ absolute time to avoid drift) or a signal with a hardware time to trigger the sequencer. When waiting on a signal a thread goes to the pending state, but when using a sleep function it goes to the delayed state.

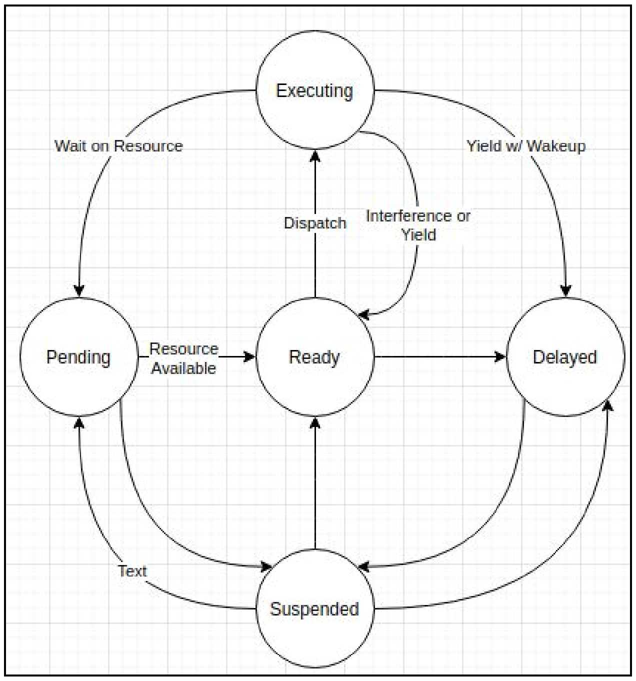


Figure 4: SW Service State Machine Diagram

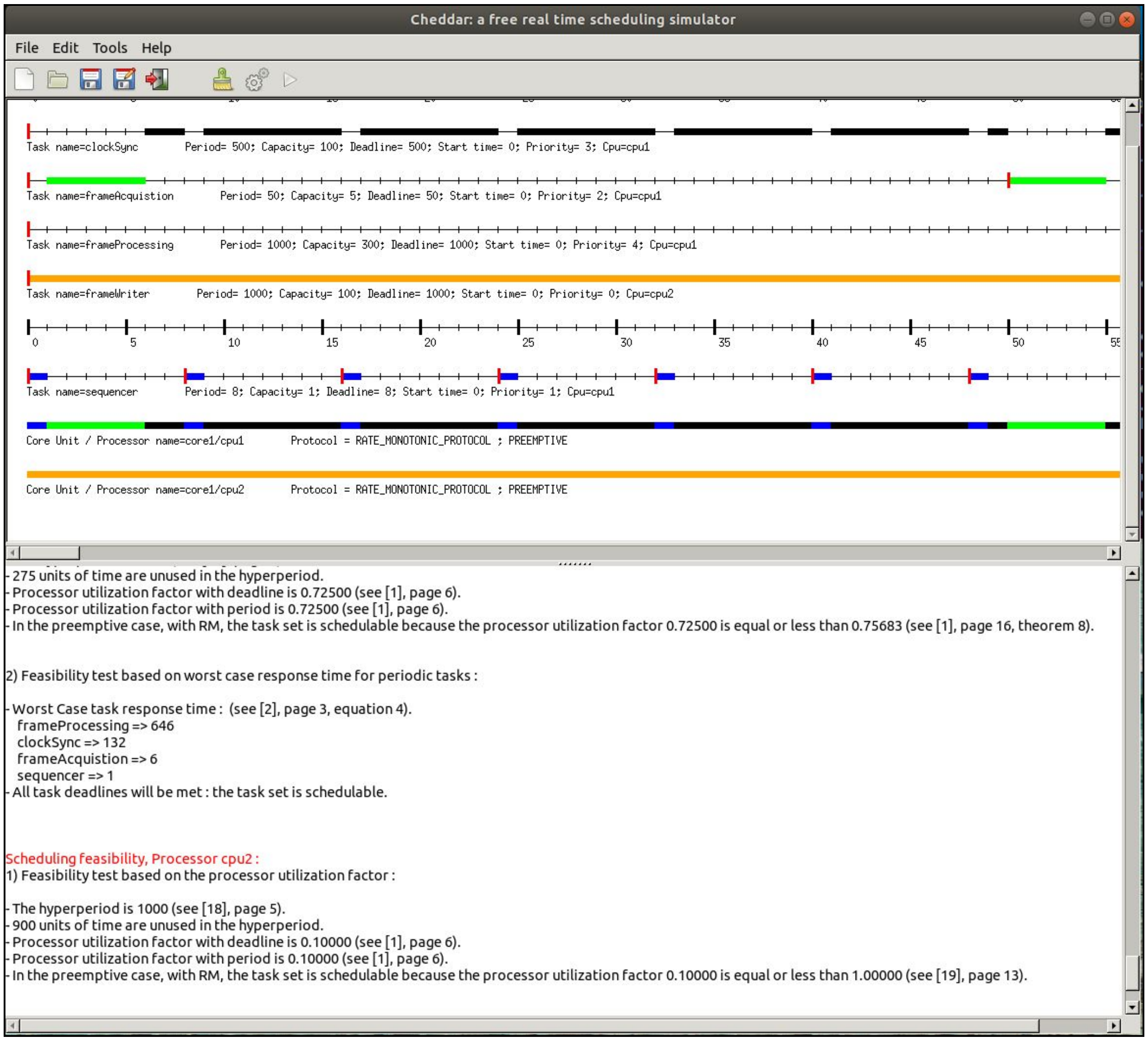
Software Analysis and Verification Methods

We will use syslog messages to verify our real-time performance. Each thread will add a comma-delimited log message to the log at the start and end of execution time and when key events occur. This will enable us to evaluate that the periodicity of our threads agrees with our schedule and enables us to find each thread's worst-case execution time over many iterations. Our log file will contain messages when each read frame was written to the circular buffer, messages when the Clock Sync thread started, found a new time frame and stopped execution; messages when the Frame Processing thread started, read a message from the Select queue, inserted a message in the Write queue and ended execution; and messages from the best-effort write back thread showing when the thread was able to perform its work. All these log messages will be imported as a CSV file into a spreadsheet tool and we'll plot the timestamps to verify all thread start times are monotonically increasing and calculate the start time jitter.

During implementation we'll occasionally plug our thread statistics into Cheddar to verify our schedule is feasible. Since our only thread accessing I/O is best effort, we will set our timeline equal to our deadline. We will also evaluate the impact to CPU Utilization with the image processing features On/Off.

We also plan to use htop to verify thread priorities and average CPU loading for each thread. And time-permitting try to include ftrace, sysprof and/or gpro.

The following screenshot shows our preliminary Cheddar evaluation. We haven't implemented any of the image processing algorithms so the run-times are pure guesses. We will be assigning the best-effort frame writer thread to its own coreto prevent impact to our RM schedule. For this analysis we assume the read function just copies the most recent frame into a local Mat structure and all the heavy lifting is done by the UCV device driver on another core. For CPU utilization we assumed the acquisition thread would be 10% (probably overkill if only doing two copies), 20% for the clock sync thread because it does some image processing, 30% for the Frame Processing thread because it will end up doing the most pixel math and 12.5% for the sequencer. This analysis shows the current CPU utilization as 72.5 % which is feasible and less than the RM LUB. If the any of the threads consume more CPU than was modelled, we'll have to consider using more than 2 cores.



```
Task name=clockSync        Period= 500; Capacity= 100; Deadline= 500; Start time= 0; Priority= 3; Cpu=cpu1
Task name=frameAcquistion       Period= 50; Capacity= 5; Deadline= 50; Start time= 0; Priority= 2; Cpu=cpu1
Task name=frameProcessing       Period= 1000; Capacity= 300; Deadline= 1000; Start time= 0; Priority= 4; Cpu=cpu1
Task name=frameWriter       Period= 1000; Capacity= 100; Deadline= 1000; Start time= 0; Priority= 0; Cpu=cpu2
Task name=sequencer        Period= 8; Capacity= 1; Deadline= 8; Start time= 0; Priority= 1; Cpu=cpu1
Core Unit / Processor name=core1/cpu1       Protocol = RATE_MONOTONIC_PROTOCOL ; PREEMPTIVE
Core Unit / Processor name=core1/cpu2       Protocol = RATE_MONOTONIC_PROTOCOL ; PREEMPTIVE
```

```
- 275 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.72500 (see [1], page 6).
- Processor utilization factor with period is 0.72500 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.72500 is equal or less than 0.75683 (see [1], page 16, theorem 8).

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

- Worst Case task response time :  (see [2], page 3, equation 4).
  frameProcessing => 646
  clockSync => 132
  frameAcquistion => 6
  sequencer => 1
- All task deadlines will be met : the task set is schedulable.


Scheduling feasibility, Processor cpu2 :
1) Feasibility test based on the processor utilization factor :

- The hyperperiod is 1000 (see [18], page 5).
- 900 units of time are unused in the hyperperiod.
- Processor utilization factor with deadline is 0.10000 (see [1], page 6).
- Processor utilization factor with period is 0.10000 (see [1], page 6).
- In the preemptive case, with RM, the task set is schedulable because the processor utilization factor 0.10000 is equal or less than 1.00000 (see [19], page 13).
```

Figure 5: RM Analysis of Proposed Final Project Solution

Project Research

1. 4-part video series Professor Siewert provided in the Lightboard Studio Rough Cuts

    a. http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Lightboard-Studio-rough-cuts/L-N9.1-Final-Project-Services-Decomp-Does-Not-Work.mp4
    b. http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Lightboard-Studio-rough-cuts/L-N9.2-Final-Project-Services-Decomp-Not-Fault-Tolerant.mp4
    c. http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Lightboard-Studio-rough-cuts/L-N9.3-Final-Project-Services-Decomp-Somewhat-Fault-Tolerant.mp4
    d. http://ecee.colorado.edu/~ecen5623/ecen/video/lecture/Lightboard-Studio-rough-cuts/L-N9.4-Final-Project-Services-Decomp-Best.mp4
2. Leverage JPEG Turbo library to accelerate image encoding. Not sure if this will be required but was interesting and could help reduce WCET.
    a. https://libjpeg-turbo.org/
    b. http://imrid.net/?p=3917
3. Synchronizing Linux timers with network time (NTP). Wanted to do some research in case this was required / would provide improvements to our performance.
    a. https://www.akadia.com/services/ntp_synchronize.html
    b. https://askubuntu.com/questions/254826/how-to-force-a-clock-update-using-ntp
    c. https://www.linux.com/topic/networking/keep-accurate-time-linux-ntp/
    d. https://en.wikipedia.org/wiki/Network_Time_Protocol
    e. http://doc.ntp.org/4.1.0/ntpd.htm

Project Schedule and Individual Contributions

Project Schedule

There are two major milestones for this project: the Project Report due August 3rd and the Final Project Demonstration and Report with due August 7th. Additionally, there is the Course Exam 2 on 7/29, which will occupy available development time for the team and need to be accounted for in the schedule. These items are in bold in the project schedule outlined below.

Proposal Development Schedule

- 7/24: Complete Project Proposal with outline of software services, system verification, and Cheddar analysis.
- 7/25: Begin development of Sequencer, Frame Acquisition, and Frame Difference Threshold threads.
- 7/26: Study for Exam 2
- 7/27: Complete Sequencer, Frame Acquisition, and Frame Difference Threshold threads.
- 7/28: Study for Exam 2
- **7/29: Exam 2**
- 7/30 - 8/1: Complete development of Frame Select and Frame Storage threads. System integration to show minimum requirements being met by project
- 8/2: Complete majority of Project Report
- **8/3: Project Report Due**

- 8/4-8/5: Implement Stretch Goal and Extra Group Requirement services (Edge Finding and Advanced Image Processing)
- 8/6: Capture project demonstration video
- **8/7: Final Project Demonstration and Updated Report Due**
- 8/8: Celebrate the end of the semester!

Brian Ibeling

I'll be assisting with overall system design, developing software for the Sequencer and Store Frames service threads, supporting system testing, and developing tools for the analysis of timing drift and jitter. I helped to outline the project proposal document, project requirements, and project schedule, and will work to outline the final project report similarly. I'll also focus on ensuring that the system is meeting all requirements and operating in a deterministic and real-time fashion through the analysis tools described in the Software Analysis and Verification Methods section above.

Josh Malburg

I will primarily be focused on the intermediate threads for acquiring images, detecting clock hand motion, selecting frames for image processing and shoveling images to the frame writer thread. I'll be setting up the circular buffer and message queues for passing data. My current plan is to threshold difference frames to detect motion but will apply additional image enhancement techniques if required. I'll alo develop the necessary image processing pipeline to highlight the clock hands and borders. I'll be responsible for capturing the WCET of the threads and will tweak the algorithms to ensure our RT schedule can be met. I'll use the WCET with Cheddar as evidence.

Both of us will work on the final report and assist with preparing the final demonstration video.