ECEN 5623-401, University of Colorado Boulder

RTES Final Project: Synchronome

Dayton Flores

August 8, 2022

Table of Contents

[INTRODUCTION 2](#_Toc110452040)

[FUNCTIONAL REQUIREMENTS 2](#_Toc110452041)

[Problem Statement and Objectives 2](#_Toc110452042)

[REAL-TIME REQUIREMENTS 2](#_Toc110452043)

[Approach and Methodology for Evaluation 2](#_Toc110452044)

[FUNCTIONAL DESIGN 2](#_Toc110452045)

[Overview 2](#_Toc110452046)

[Diagrams 2](#_Toc110452047)

[REAL-TIME DESIGN 2](#_Toc110452048)

[Overview 2](#_Toc110452049)

[Diagrams 2](#_Toc110452050)

[PROOF-OF-CONCEPT 3](#_Toc110452051)

[Tests Completed 3](#_Toc110452052)

[Example Output 3](#_Toc110452053)

[CONCLUSION 3](#_Toc110452054)

[APPENDICES 3](#_Toc110452055)

[Appendix A 3](#_Toc110452056)

# INTRODUCTION

|  |  |
| --- | --- |
|  | This project uses a Logitech C270 camera connected to a Raspberry Pi 4b+ via USB to snap pictures of an external analog clock with a non-sweeping seconds hand. Each picture snapped will show the seconds hand placed on a unique tick without motion blur. The images will be stored into the Pi’s FLASH storage. |

# FUNCTIONAL REQUIREMENTS

|  |  |
| --- | --- |
|  | The main functional hurdle is that there are 2 independent clocks in this project that we want to have synchronized with each other, the external analog clock and the internal RPi4b+ clock.  In order to accomplish this synchronicity, an oversampling must be done. This is easily accomplished due to the physical limitation of the Logitech C270 camera of <30 Hz while the RPi4b + has a clock in the GHz range.  After oversampling, the next functional hurdle is to analyze and determine the first bad frame. This is the done by taking the difference between current frame and previous frame. Once a current frame has passed the threshold, it will be marked as bad and passed down the pipeline.  The next service will read each frame passed from previous service and look for bad frames. Once a bad frame has been detected, it will store the frame indexed ½ the period away from the bad frame as a good frame and pass it along to be processed.  Processing will take place next, as each selected frame passed to this service will be converted from color into grayscale. From there, the processed stable, unique image will be stored into FLASH storage of the RPi4b+. |

# REAL-TIME REQUIREMENTS

|  |  |
| --- | --- |
|  | The main real-time hurdles are the blocking limitations generated by 2 services.  1 of these limiting services is frame acquisition, and the limitation comes due to blocking while waiting for a frame to load into the I/O buffer (i.e. the **select()** function). If no frame is detected then the function will block for a certain amount of time and throw the synchronicity of the project off.  The other limiting service is frame writeback, and the limitation comes from the nature of writing to FLASH memory. Specifically, the large amount of time it takes to writeback to FLASH.  Both of these limitations require the system to be implemented with separate real-time services analyzed with rate monotonic analysis. This will allow the system to be more robust and prevent unexpected blocking. |

# FUNCTIONAL DESIGN

## Overview

|  |  |
| --- | --- |
|  | S1 Frame Acquisition will acquire X frames per invocation, operating at 10 Hz. This will ideally operate at the lower bound of expected FPS from the Logitech C270 (20-30) to bring in enough oversampling but also prevent opportunity for excessive blocking with **select()**. Thus at 10 Hz, X would be 1 frame for this service.  S2 Frame Difference Threshold will parse X amount of frames per invocation, operating at 2 Hz. In this case, X would be the amount of frames inserted by S1 since the last invocation of S2. This will ideally operate at 10 frames per second, and thus at 2 Hz means X would be 5 frames for this service.  S3 Frame Select will parse X amount of frames per invocation, operating at 1 Hz. In this case, X would be the amount of frames marked by S2 since the last invocation of S3. This will ideally operate at 1 frame per second, and thus at 1 Hz means X would be 1 frame for this service.  S4 Frame Process will parse X amount of frames per invocation, operating at 0.5 Hz. In this case, X would be the amount of frames selected by S3 since the last invocation of S4. This will ideally operate at 1 frame per second due S3’s output, and thus at 0.5 Hz means X would be 2 frames for this service.  S5 Frame Writeback will parse X amount of frames per invocation, operating at 0.25 Hz. In this case, X would be the amount of frames processed by S4 since the last invocation of S5. This will ideally operate at 1 frame per second due to S4’s output, and thus at 0.25 Hz means X would be 4 frames for this service. |

## Diagrams

|  |  |
| --- | --- |
|  | Pending diagram showing algorithm implemented for S2 to ensure synchronicity between the 2 clocks as well as finding stable, non motion-blurred frames. |

# REAL-TIME DESIGN

## Overview

|  |  |
| --- | --- |
|  | S0 Sequencer has its frequency generated by the RPi4b’s internal oscillator signaling an armed interval timer at 120 Hz. From here, S0 executes which will post semaphores respective to each RT service depending on the modulus result (i.e. if S0 is executing at 120 Hz and S1 Frame Acquisition is expected to execute at 10 Hz, then S1 will have its semaphore given at a rate of MOD 12 since 120/12 = 10). The number 120 Hz was taken as advised by Professor Siewert recommended in his Coursera video about the synchronome (having S0 execute at over 100 Hz but under 1000 Hz is ideal), as it fulfills requirements and also has many whole factors to produce derivative frequencies out of (i.e. 1, 2, 3, 4, 5, 6, 10, 12, 20, 24, 30, 40, 60, 120).  S1 Frame Acquisition has its frequency set to 10 Hz. This is due to the physical limitation of the Logitech C270 (operating between 20 and 30 Hz). If we set S1’s frequency of operation any higher, there is higher chance of blocking due to **select()**. The blocking delay in **select()** was also modified to be no more than the period at which S1 is executing (i.e. 1/10 Hz = 0.10 s) so if a frame is not read by that time no significant blocking will occur. However, any significant blocking at this frequency will be highly unlikely due to the camera’s ability to consistently operate between 20-30 Hz in good lighting.  S3 Frame Select has its frequency set to 1 Hz. This is due to the nature of this service, when working with external 1 Hz analog clock the goal is to grab 1 frame per second. Setting S3 any higher or lower will require additional work so this is done for simplicity’s sake.  Having the above already defined, the rest of the frequencies should follow rate monotonic analysis for having higher priority tasks operate at higher frequencies.  Keeping the above in the mind, the services surrounding S3 (S2 Frame Difference Threshold and S4 Frame Process) have their frequencies multiplied/divided by 2, respectively, since that is the smallest factor of S0’s 120 Hz. These were guessed and checked, and WCET estimates showed that these frequencies fulfilled rate monotonic analysis (although the code has been made modular enough with macros that frequency changes for any RT service would not be such a heavy lift). |

## Diagrams

|  |  |
| --- | --- |
|  |  |

# PROOF-OF-CONCEPT

## Analysis

|  |  |
| --- | --- |
|  | WCETs were calculated by isolating each service into separate source codes and removing the semaphores. Then, clock\_gettime’s were placed before and after each invocation of the respective service. Difference was calculated using **delta\_t()** (referenced from starter code **update-location**) and syslogged.  Average of WCETs for each service was taken over ~180 samples and used in the RM LUB feasibility calculation below:  **Feasibility**    **Cheddar Scheduling Stimulation**    **Jitter**  Jitter analysis was done for services S0-S5 by syslogging the **clock\_gettime()** for each invocation during full operation. See below for screenshots of these plots: |

## Example Output

|  |  |
| --- | --- |
|  | See below for pictures of successful test over 1 minute (frame 0000 and frame 0060): |

# CONCLUSION

|  |  |
| --- | --- |
|  | The conclusion here is that for any limiting any potentially blocking services, separating services into RT services using rate monotonic analysis to assign frequencies and priorities allows for synchronizing 2 independent clocks. A note to make is that continuous re-synchronization between the 2 clocks would be ideal as to avoid drift and jitter (at periods much shorter than the periods it would take to drift). If drift and jitter are small enough, synchronization can also be crudely achieved with 1 Hz external analog clock using the “human shotgun” method. However, for higher frequency clocks this is not possible as human reaction time is not quick enough. This shotgun method would not be nearly as reliable a method as re-synchronization between the clocks every so often. |

# REFERENCES

|  |  |
| --- | --- |
|  | Referenced **simple-capture-1800** + **sequencer-generic** projects written by Professor Siewert. Also referenced Professor Siewert’s **L-N9.X** video series regarding the synchronome project.  Met with Professor Siewert in multiple debug sessions for assistance with conceptualizing and polishing the theoretical approach, with emphasis on the frame difference threshold. |

# APPENDICES

## Code

|  |  |
| --- | --- |
|  | See attached **..\FinalProject\synchronome** for source code. |