REAL TIME VISUAL SYNCHRONOME USING POSIX THREADING AND RASPBERRY PI EMBEDDED LINUX

Authored By: Sricharan Kidambi S

Introduction

As stated in the project proposal, the aim is to build a visual synchronome with hard real time constraints over 1Hz and 10Hz frequency.

A C-2700 Web camera captures a set of incoming frames at a fixed rate. The perfect image (without any glitches, motion blur, and skips) is selected by applying a frame differentiation algorithm through a set of ranges ranging from 0-255. The final 180 images will be displayed at a rate of 1Hz thereby creating a time-lapse effect when played continuously.

Minimum requirements

The minimum requirement for this project requires a raspberry PI A53 microprocessor with USB connection to a web camera. The Logitech C200 series of camera is used for this purpose. It can also be done using beagle bone and the same functional and timing correctness can be observed.

Basic Requirements to be noted

1. The VGA output of camera must have a resolution of 640 x 480.
2. The individual frames are captured in the camera
3. The resulting Image file can be in PPM or PGM (RGB or Grey Scale).
4. No Observable glitches (Motion blurs or skips should be observed).
5. Host or target platform “uname -a” should specify clearly which was platform for test run.
6. Use Timestamps that should be embedded on every frame header

Source: <https://drive.google.com/drive/search?q=minimum%20requirements>

Verification of Minimum requirements

1. You must verify that your image acquisition is accurate enough so that you can observe an EXTERNAL WALL CLOCK for a period of 30 minutes with high accuracy, or EXACTLY 1800 frames and that you capture the minutes display in a range of 0 to 30 or shifted anywhere within a start between 0 and 59 and stop anywhere at start+30 as displayed on the external clock and captured in your frames. So, for example, I should see 6:06 PM in the first frame and 6:36 PM in the last. Since your observability is 1 second, there is a small probability that you would see 6:06 PM and 6:35 PM if you acquisition started at 6:06 PM and 0.5 seconds and ended at 6:35 PM and 59.5 seconds. To ensure that you are accurate to +/- 1 frame or 1 second over the 30-minute period, you should count the initial frames that show the starting time and minute (e.g., 6:06 PM) and the number of frames that show the final stop time and minute (e.g., 6:36 PM or 6:35 PM). You acquisition should be accurate so that you have no more than 1 second of error in the observation of the external clock over the 1800 second time-lapse acquisition – the same goes for your timestamps in the headers - the difference should be 1800 seconds +/- 1 second or less. Please see this diagram for more detailed information. You must compute your average frame jitter and determine if you have accumulated latency that builds up, or if you are running fast and by how much.
2. You must ALSO observe some physical process like the melting of a cube of ICE alongside your EXTERNAL CLOCK so that it proves this is a true time-lapse sequence when you encode it.
3. You must encode your final video which includes the EXTERNAL CLOCK and the observed physical process into an MPEG2 or MPEG4 program stream using ffmpeg and include this on your final Zip files or DVD along with your PPM frames.
4. Include one more time-lapse video of any physical process indoors or outdoors that is of interest to you, but do not include the raw PPM frames for this second video, just the MPEG on your Zip files or DVD.

Target goals

Please choose to add one or more of the following features to your project and re-run your verification:

1. Compression of frames on your target so you can store more than 2000 frames and so that you have less to transfer over Ethernet.

2. Continuous download of frames over Ethernet so that you can run indefinitely and never run out of space on your flash file system which should maintain only the last 2000 frames.

3. Any other method you can dream up to capture and save at least 3x the minimum number of frames through compression and download or other means.

4. After adding either or both above features for compression and download, re-verify your frame jitter and accumulated latency as described above and describe and account for any differences you see.

Stretch goals

Please choose to add ONE of the following features to your project:

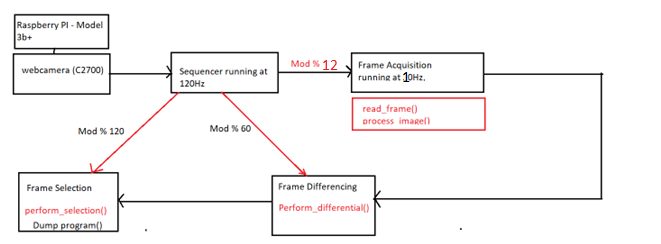
1. Run at a higher frame rate of 10 Hz with continuous download, this time for 9 minutes, which will produce up to 6000 frames (about 6GB uncompressed for 640x480) and repeat the jitter and accumulated latency verification at this higher rate.
2. Run at full frame rate and scan your RAW YUV or RGB image for foreground changes and acquire only images where you detect change that exceeds more than 1% of the total pixel value summed. Timestamp each image when you trigger and observe a physical process with change. Stop when hit the same 1800 frames used in the verification. Attempt to repeat the EXTERNAL CLOCK verification and see if you can get the trigger to detect the change of display on the CLOCK you are observing with any sort of reliability.
3. Any other test you can dream up that runs at a higher rate, adds an element of real-time image processing, or otherwise significantly increases CPU loading significantly higher than 70% and allows you to verify if this causes more jitter and accumulated latency.

Real - Time requirements

Minimum Goals Met

1. Asymmetric Multiprocessing method on one AMP core.
2. The requirements should be free of glitters.
3. 180 frames acquired at 1Hz

BLOCK DIAGRAM



High Level Explanation

1. The hardware involved a Raspberry 3b+, for running the OS in ARM Cortex A53 processor.

The processor runs Linux – soft real time capability. But this project intends to provide hard real time capability. By providing visual synchronome.

1. Totally 4 threads are called in addition to the main thread.
2. The explicitly defined threads are provided with scheduling attributes.
3. This is done to perform load balancing so that the cores are not exhausted making rate monotonic scheduling infeasible and satisfying the minimum requirements. (At least 2 processes should run on the same core).
4. I have performed the task with both transforming from greyscale to negative and vice-versa.
5. The resulting frames are dumped to the flash drive and hence we obtain the required output without glitches.

Detailed Explanation of Every Subsystem

Main Subsystem - Sequencer

1. The sequencer runs on 120Hz.
2. This function is responsible for sequencing all the events in order and in their assigned frequencies.
3. Semaphores are assigned to ensure that deadlock does not happen due to extensive resource sharing.

Subordinate Subsystem – 1 The acquisition routine

1. The acquisition consists of 2 main subroutines
2. readframe() – This is the main routine that senses the incoming image and identifies whether it can be read by the input camera or not.
3. Processimage() – This routine does the image processing and writes the image to the buffer hence one buffer instance of size 640x480 will consist of all the pixels needed to hold one frame.

Subordinate Subsystem – 2 The differentiation routine

The differentiation is where the real process of finding the stable frames happen.

The differentiation routine will calculate pixel by pixel difference of one frame to next incoming frame. From this point there are multiple ways to determine whether to find the most stable image.

The method I used involved taking ranges between 0-255 and counting in how many ranges value is stored. The more variety of ranges having values, that is the point of tick detection, meaning that is where the secondhand slips one step forward.

Subordinate Subsystem – 3 The Selection and Dump routine

The selection and dump routine will filter out the stable images obtained from differentiation and dump it in the flash drive. We can either do that on the same method or create another instance and perform this operation if we need to do some transforms as well.

Processor Core Task Accommodation Details

1. Sequencer - Core 1
2. Acquisition - Core 2
3. Differentiation - Core 3
4. Selection - Core 3

RMA details

Since only differentiation and selection runs on the same core, it is enough if we run RMA tests for those two.

Frame Differentiation

C1 = 22.60

T1 = 500 - differentiator service runs on 2Hz

Frame Selection

C1 = 4.22

T1 = 1000 - selector service runs on 1Hz

The documents and analysis screen captures supporting this are attached in this zip file

Conclusion

2 services running on single core, 1Hz requirement is satisfied successfully.

References:

1. Prof Sam Siewart for being patient with us and conducting several debug sessions.
2. The TA’s for holding repeated office hours
3. Friends Abijith Ananda Krishnan for giving some important debugging techniques for conformation, and many others who gave their insightful information on Slack channel

Appendices

1. Firmware -> Main program – code.