Real Time Embedded Systems – ECEN 5623

Summer 2019

Professor Sam Siewert

Report By: Poorn Mehta

Platform: Jetson Nano (Camera: Logitech C920s)

Homework 5

Q1: Research, identify and briefly describe the 3 worst software and hardware computer engineering design flaws and/or software defects of all time [product lack of success can be considered a failure such as Blackberry Storm, Windows Genuine Advantage, Windows 8, Windows ME, Apple Lisa, Pentium FPU bug, as well as mission failures such NORAD false alarms, Mars Express Beagle 2, Challenger and Columbia Shuttle Loss]. State why the 3 you have selected are the worst in terms of negative impact, negligence, and bad decisions made, but why the failure was not really a real-time or interactive systems design flaw. Rank them from worst to least bad.

Solution:

(1) **Columbia Space Shuttle Loss** [1]

- In my opinion this is the worst failure for multiple reasons – including massive negative impact, humongous loss property and personnel, poor project management, and immature decisions.
- **Short Description** of the event:
  - The Columbia Space Shuttle was destroyed while reentering the orbit of Earth, after end of a mission in space.
  - During the Launch, a piece of foam insulation broke off, and hit the left wing of the orbiter.
  - While this was downplayed by authorities at NASA (loosely based on previous similar events), it actually dealt serious damage to the thermal insulation coating of that wing.
  - Some engineers tried to get a detailed estimation of this – soon after the launch, however, they're restrained by NASA managers with the reasoning – that even if it can be proven that the wing indeed is faulty, there's no possible action which can be taken to counter it.
  - This argument however, was proven to be completely baseless by CAIB (Columbia Accident Investigation Board), which was formed for thorough investigation, following the mishap.
  - CAIB indicated two possible solutions – if the fault was found on time (had the NASA managers allowed engineers to dive deeply to get an accurate estimate over the probable damage, and then instructed Astronauts to conduct a spacewalk and manually inspect the left wing): First, was to rescue the Astronauts through Atlantis, which was possible since the Columbia was

carrying unusually large amount of consumables due to an extended duration orbiter package, and Atlantis was scheduled to launch just a few days after planned landing of Columbia. Second, was to use a few specific means to repair the wing in the space, which might have succeeded.
    - The material used in the wings – RCC (reinforced carbon-carbon) was believed to be 'impenetrable' by NASA, especially from objects such as foam. This was again a false assumption, which wasn't proven in any possible way.
    - NASA was accustomed to 'foam shedding' events, even though they're not expected, nor normal – simply, since they didn't pose threats to earlier missions. This in particular, was too lax approach to critical projects such as manned space missions.
- **Negative Impact**: This accident cost the Nation in many ways, major ones being lives lost of Astronauts, as well as financial penalties in hundreds of millions of dollars. NASA's space program lost some reputation, and it also received probation of more than a year – on many other space missions.
- **Negligence**: This was at its peak, since NASA ignored all other similar events, discarding their significance because they didn't prove to be fatal in the end. Clearly, it is a very poor practice which must be avoided – at least for such high-stake missions.
- **Bad Decision** Making: As described in the event overview, there were good chances of bringing Astronauts back safely, and minimizing the damage to important equipment too – however, owing to ignorance, and flawed logic used to make key decisions, ensured the catastrophic accident.
- **Not a Real Time Problem**: The issue was largely – flawed mechanical design which was shedding the slabs of foam insulation during the Launch. No real time constraints were violated (or at least, played any role in this incident), and therefore, it can't be considered a real time failure.

(2) **Challenger Space Shuttle Loss** [2] [3]

- In my opinion this is the second worst failure – only after Columbia Space Shuttle Loss for reasons similar to the latter. The key factor reducing intensity of this failure, is that there was no possible chance/solution/action that could have been executed, after the launching.
- **Short Description** of the event:
    - The Challenger Space Shuttle was destroyed shortly after the Liftoff.
    - The root cause was the failure of O-rings mounted on SRB (Solid Rocket Boosters), which occurred due to very cold temperatures
    - The exact problem which lead to failure of Challenger, was foreshadowed years ago by an Aircraft Manufacture: clearly stating that if a SRB design failure somehow leads to burn-through of the casing by hot gases, and should that take place adjacent to fuel tank or orbiter, it can easily prove to be a fatal incident.
    - The contractor responsible for the construction and maintenance of SRBs that were used in Challenger – Morton-Thiokol, used a fundamentally flawed design including rubber O-rings for specific joints of SRBs.

- This was pointed out multiple times by engineers at Marshall Space Flight Center, however, they were completely ignored.
- A Discovery flight experienced a fair amount of hot gas blow-by roughly 2 years before Challenger disaster, however, it still was discarded simply as 'an acceptable risk' by engineers at Morton-Thiokol.
- Even after 7 out of 9 flights experienced similar issues, a few to much greater extent – the future flights were still not grounded, and the management simply started to improve design. Effectively, this resulted in the Challenger accident, not much later.
- What worsened the situation – was a couple of added facts: such as low enough temperatures on launching day which eventually prevented O-rings from working completely, ignorant NASA management towards concerns expressed regarding O-rings safety on multiple occasions.
- Evidently, engineers at Morton-Thiokol, told explicitly to NASA management in a conference call that there is a high chance of failing for both O-rings, in the given weather conditions, just a day before the launch. However, the NASA management simply disregarded it completely, while violating multiple flight rules (the O-rings were considered Critical 1 components, meaning that if they fail, it will result in complete loss of property and personnel – and no Critical 1 component warning should be ever taken lightly)
- As correctly estimated – by the engineers, the temperatures were low enough to switch the properties of O-rings material from elastic and flexible, to that of rigid and brittle. This made the O-rings unable to seal the SRB joints in time, to prevent a burn-through of the casing by hot gases.
- Adding to this, was stronger than ever experienced wind shear forces by Challenger, just a few seconds after launch. This caused the initial seal of Oxides (which were created as a result of the vaporized brittle O-rings) to be removed as well, and finally leaving the right SRB structure with a rapidly expanding hole on side.
- Shortly enough, the liquid hydrogen tank failed, owing to multiple events that occurred due to uncontrolled burn through, and the entire shuttle was enveloped in a fireball.
- Consequently, the vehicle broke up, ending with crew cabin hitting the ocean, with deceleration on impact being somewhere around 200g.

- Negative Impact: This was quite similar to that of Columbia disaster.
- Negligence: It was evident from this accident, that politics shouldn't be involved in critical technical matters. Multiple managements – at Marshall Space Center, Morton-Thiokol, and NASA – neglected high level warnings given by engineers. In many ways, the way NASA handled this and Columbia incidents, are largely disappointing.
- Bad Decision Making: Due to political reasons, and unknown agenda, a chain of decisions were made over the years regarding concept of O-rings, effect of cold temperatures on it, issues regarding SRB safety etc. These put number of manned space flights at huge risk, before the Challenger incident.

- Not a Real Time Problem: The issue was mostly related to structural design flaws, combining mechanical engineering and material science. No real time constraints were violated (or at least, played any role in this incident), and therefore, it can't be considered a real time failure.

(3) **NORAD False Alarms** [4] [5]

- In my opinion, this is the third worst failure of all time, not due to the actual damage caused – but because of the severe potential it had to do so. Possibly, it could have triggered a Third World War, with high chances of wiping off entire mankind, due to the number of nuclear warheads present at that time.
- Short Description of the event:
  - The North American Aerospace Defense, has installed multiple computer systems to detect an incoming nuclear attack.
  - This system, failed on multiple occasions due to various reasons – including both, Human Error, as well as Technical Glitches.
  - The first time, a technician put on a test tape, however without taking it offline first. The tape contained a highly realistic simulation of massive Soviet Nuclear attack, which effectively raised alarms across the US military bases indicating the same.
  - On other occasions, a communication failure of some kind, resulted in multiple computers falsely identifying nuclear threats, which led to serious alarms as well.
  - At those times, the US was in Cold War with Soviet Union, which was naturally combined with arms race. Due to this, both nations had hundreds of nuclear weapons. Had the warnings not been cross checked first, it definitely would have resulted in a Third World War.
- Negative Impact: Fortunately, US didn't act hastily and averted potential crisis on multiple occasions, however, it did pose a significant threat.
- Negligence: It is not very clear how negligence played role, except for the assumption that the technician who loaded test tape, could have potentially ignored the proper procedure – and thus triggering the alarms.
- Bad Decision Making: It can be argued that the bad decisions would have been taken from the system developers, since they failed to figure out and fix communication problems that lead to false alarms more than once.
- Not a Real Time Problem: The system certainly had some real time concepts embedded in it; however, the presented faults don't fall in that category – since the root causes lied in the system design, probably software or hardware, which went undetected.

Q2: Research, identify and briefly describe the 3 worst real-time mission critical designs errors (and/or implementation errors) of all time [some candidates are Three Mile Island, Mars Observer, Ariane 5-501, Cluster spacecraft, Mars Climate Orbiter, ATT 4ESS Upgrade, Therac-25, Toyota ABS Software]. Note that Apollo 11 and Mars Pathfinder had anomalies while on mission, but quick thinking and good design helped save those missions]. State why the systems failed in terms of real-time requirements (deterministic or predictable response requirements) and if a real-time design error can be considered the root cause.

Solution:

(1) **Three Mile Island** [6] [7]

- I consider it as one of the worst failures, due its potential to deal damage on a vast scale. The accident exposed about 2 million individuals to harmful radiation. Fortunately, it was small enough to not cause any impact on the health of this population.
- **Short Description** of the event:
  - The cooling system malfunction caused a partial meltdown of the reactor core 2 (TMI-2), at Three Mile Island Nuclear Power Plant.
  - It was result of cascading failures, which without the combination – would have been considered not so significant.
  - In an attempt to clean some specific filters, some water entered instrument airline, shutting down multiple pumps.
  - This led to desired switching on of auxiliary pumps, however, all three valves of these were manually closed as a part of maintenance. This violated a very important safety guideline of nuclear power plants, which prevented reactor from operation – if valves to auxiliary pumps are closed.
  - This factor, was later marked as the root cause of the failure.
  - Other than that, poor UI and System Design lead to a major confusion to the crew, falsely leading them to believe that the coolant in the core was ‘more’ than required, while in fact, it was potentially the other way around.
  - Due to all these contributing factors, the meltdown took place, and the building was heavily contaminated with radioactive materials.
  - However, the external environment remained largely unaffected, averting a disaster.
- **Negative Impact**: Largely, it posed a significant danger to a very large group of people, and also it raised concerns about safety of Nuclear Power Plants. It also penalized the US financially, more than 1B$.
- **Negligence**: For the most part, the crew neglected a few important rules of Nuclear Safety, and also failed to understand the seriousness of the situation.
- **Bad Decisions**: Owing to the flawed indicator design, operators reduced coolant flow further – doing exactly opposite of what should have been done. This exacerbated the situation quite a lot.

- **Real Time Defect**: The system used a bad UI design, as well as completely flawed Valve Indicator – that lead to great confusion for the operators, and resultantly, worsened the incident. The fault can be considered real time, because the design was not deterministic (of the valve indicator), and provided misleading information in critical real time situation. However, it wasn't the root cause in this case.

(2) **Therac-25** [8]

- Therac-25 was a computer-controlled radiation therapy machine, faults in which – severely overdosed multiple patients. Due to the nature of it, and probable wide scale of health threats, including deaths – makes it quite a serious failure.
- **Short Description** of the event:
  - Owing to a glaring mistake in the software for the Therac-25, which was ported from older machines, it gave heavy dosages of radiation – as much as 100 times more, when combined with specific fault mode of operators inputting settings rapidly.
  - The X-Ray mode of the instrument, ideally should be working with a target in place (flattening filter and collimator) – which reduces the radiation to desired amount.
  - However, when operators incorrectly selected X-Ray mode before quickly changing to Normal Mode, the instrument – due to asynchronous design, stayed in X-Ray mode, but didn't bring out the target.
  - This clearly resulted in massive direct overdose of the radiation.
  - It also had several other issues, such as bad software design practices, due to which it was almost impossible to fully test the instrument, buggy old code – which worked in older instruments because there were hardware locks present; while in the newer design they were removed completely, etc. lead to a major issue.
  - The design was also open loop – with hardware lacking any abilities to provide a feedback to the software, which allowed the problems to occur without being detected even.
  - At times when system did notice an erroneous state, it simply prompted a generic error message with a number, which was not included in the user manual. Therefore, the operators simply performed an override, without being aware of the severity of that action.
- **Negative Impact**: It resulted in death of multiple patients, while health of several others was affected quite seriously. For the manufacturer, it caused drastic repercussions as well.
- **Negligence**: The operators, as well as the company personnel didn't believe the multiple complaints received from patients. This was quite irresponsible, for such a sensitive field.

- **Bad Decisions**: The engineer's approach to use the old software without considering changes in the hardware, and the management's decision to launch the product without testing at all – were indeed poorly made decisions. These are standard practices, and must be followed at all costs for healthcare industry.
- **Real Time Defect**: The instrument used open-loop asynchronous configuration, and with much slower response time which allowed a window of opportunity for errors to pass through, in case of quick human inputs. If the system was made hard-real time, then all of those issues would have been averted.

(3) **Ariane 5-501 & Cluster** [9] [10]

- This accident resulted in loss of property, valued at about 370M$, while also posing a small threat to the environment.
- **Short Description** of the event:
  - The Ariane 5 launcher ended in a failure, shortly after the liftoff – effectively breaking up, and exploding. It took down Cluster spacecraft with it.
  - Ariane 5 used 2 inertial reference platforms, which is a key component in guidance of the launcher. Both of them, however, used an old software from Ariane 4, which was quite less powerful than Arian 5, as well as incorporating a different design.
  - Moreover, the software wasn't protected against integer overflow, while converting a floating-point number to 16bit signed number.
  - From the perspective of the Ariane 5 system, the part of code that caused trouble was unnecessary, since it was performing alignment of the inertial reference system, only serving purpose in Ariane 4.
  - On top of this, Ariane 5's much greater horizontal acceleration, overflowed signed 16bit integer variable, and thus crashed both – primary and backup inertial reference systems.
  - This resulted in the launcher vehicle losing direction, sharply veering off from the designated path, experiencing massive aerodynamic forces, and finally destroying itself after the vehicle breakup.
- **Negative Impact**: Apart from significant financial loses, it didn't have any other significant negative impact.
- **Negligence**: This was quite a lot, as the software developers didn't even run a simulation before the real flight. Had they done that with parameters of Ariane 5 and the software they've used, the issue would have been found out. Moreover, they used the code from Ariane 4 as is, without even thinking about implications it might have due to a different system design, and power.
- **Bad Decisions**: The management which overviewed the software development, took a poor decision to not cross check or verify the actual work done by the engineers, which ended up being a giant loss.

- **Real Time Defect**: The root cause was a real time problem, since the service was running when it should have stopped long ago, as well as the bug in the code which ended up crashing guidance system in real time, which obviously is critical to the launcher. Due to unavailability of the Hard-Real Time service, the accident took place.

Q3: The USS Yorktown is reported to have had a real-time interactive system failure after an upgrade to use a distributed Windows NT mission operations support system. Papers on the incident that left the USS Yorktown disabled at sea have been uploaded to Canvas for your review, but please also do your own research on the topic. a) Provide a summary of key findings by Gregory Slabodkin as reported in GCN. b) Can you find any papers written by other authors that disagree with the key findings of Gregory Slabodkin as reported in GCN? If so, what are the alternate key findings? c) Based on your understanding, describe what you believe to be the root cause of the fatal and near fatal accidents involving this machine and whether the root cause at all involves the operator interface. d) Do you believe that upgrade of the Aegis systems to RedHawk Linux or another variety of real-time Linux would help reduce operational anomalies and defects compared to adaptation of Windows or use of a traditional RTOS or Cyclic Executive? Please give at least 2 reasons why or why you would or would not recommend Linux.

Solution:

Key Findings [11]

- The USS Yorktown implemented an automated system, which while definitely reduced workload for sailors aboard the missile cruiser – some software glitches effectively made the whole cruiser ‘dead in water’, unable to move at all for hours.
- It happened when a bad data was fed into one of the computers, by an operator.
- The root cause was that the software wasn’t designed to handle infamous ‘divide by zero’ condition, or rather not designed to reject such data.
- Due to this flaw, when one of the data field contained a zero – which ideally would never have that, the computer buffers resulted in an overflow, affected all other computers in network, and crashed the whole system.
- Reportedly, there weren’t any software developers included in this project, and it was pushed forward hastily, to be used as a ‘testbed’ – so that the effectiveness of such an automated system can be evaluated for cost and performance, compared to conventional cruisers.
- Now, there are two conflicting side regarding this problem – one supports the OS being utilized on those computers, while the other opposes.

Conflicts [12] [13]

- The side that supports Windows NT, uses the logic that filtering out bad data, is work of the application itself, and not of the OS.
- It does have a valid point, that lack of good code is the first thing to blame, however, it doesn't mean that error as obvious as this should be able to crash not just one computer – but the entire network.
- And therefore, a pool of technical experts strongly opines that the use of Windows NT was directly responsible for this incident.
- Correctly, they point out that Windows NT is known to have multiple failure modes, and is much less stable, secure, tested than Unix operating systems.
- An application error – which is executed in the user space, must not be able to disturb the kernel space greatly – resulting in such giant faults.
- Many accusations were made that Windows NT was chosen by individuals who had almost no technical knowledge, and mainly due to political reasons.
- The defenders of Windows NT also made a point saying, that USS Yorktown was essentially a 'testbed' for the new system, developed rapidly – with clear expectations of such problems, and thus the OS wasn't at fault.
- There are also conflicting reports on what happened to the cruiser, after the problem – with opposers of the OS saying that the ship has to be towed, and took quite a lot of time to fix the problem. On the other hand, the defenders of OS clearly denies the any of these events taking place.
- The link [12] provides overview of anti NT opinions, while [13] provides overview of the pro NT opinions.

Analysis of the accident in My Opinion

- While I acknowledge some valid points made from the NT favoring arguments, overall, I firmly believe that Windows NT OS was indeed at fault.
- It is a wide known fact that the Windows NT is not as stable as it should be, for use in such crucial applications.
- It can be easily observed, by the tons of issues and crashes reported of this OS by significant number of users – especially in the industrial settings.
- Due to this, and lack of clear decoupling of multiple computer nodes on a LAN network, crash in one system started cascading failures in sequence, making the entire system unusable.
- It is actually surprising to learn that a simple mistake, such as entering 0 in a data field, can cause the entire network of computers to be crashed.
- Such errors – made in user space, should not even crash that one computer even, and causing the crash of all other computers is something must not be done under any condition.

Why UNIX should be Chosen instead of Windows NT [14] [15]

- In UNIX, the condition which took place on USS Yorktown would be handled quite efficiently – even without filtering out data first. The process trying to divide by 0 would have been killed, or put in the infinite loop – which can be easily detected and killed. In either case, the kernel operations would have remained unaltered, allowing system to operate smoothly even with the bad data.
- This process then, can be restarted – giving another chance to the operator to reenter data.
- Thus, the failure would never crash a single computer. Even if it somehow ends up in kernel panic for the UNIX, it would never really affect other computers on network – in a way that they crash as well!
- Almost all professionals who have quite a lot of experience with both operating systems, prefer UNIX over NT – almost always. Following are the few points which strongly works in favor to the UNIX.
    - UNIX is more widely tested, mature, and technically superior group of operating systems, which has proven itself many times in performance, stability, security, reliability etc. especially in server environment.
    - The continuous development of UNIX over decades, have included individual testers and developers from around the globe, who are genuinely passionate for this field, and works very hard to improve the OS.
    - The Linux is preferred in many scenarios since, the open source project – with source code availability, and tons of enthusiasts striving to improve the OS – results in much better support mechanism than any traditional vendor support.
    - Typically, due to memory access violations and I/O errors, Windows NT crashes much too frequently.
    - The infamous ‘BSOD (Blue Screen of Death) is one of the most critical issues for this OS.
    - In this situation, the only solution is to restart the whole computer. There is just no other way around it.
    - Moreover, the debugging of what caused this issue, it extremely difficult – due to two main reasons: (1) Due to the very generic error codes displayed at the time of crash and (2) Due to a very long list of ‘things’ that can easily lead to this situation.
    - The Kernel Panic is similar to BSOD in UNIX systems; however, they are extremely rare. The UNIX server crashes are almost always due to hardware failures, and not due to bad software.
    - Windows NT is particularly prone to virus attacks, since the OS on Intel Hardware must be booted from a Hard Drive, and a virus that can attack MBR (Master Boot Record) of the hard drive, can be the death of OS itself.
    - However, Linux (and other UNIX operating systems that run on same hardware), can load a compressed kernel from a boot floppy disk, and thus circumventing this problem.
    - This also means that the Windows NT can be crashed from a virus that written decades ago for MS DOS even.

    - On the other hand, most of the viruses that can potentially destroy Windows NT, can't do anything at all on a UNIX OS, since a lot of them rely on the MS Windows environment to be able to deal damage.
- Due to these, and many other reasons – I strongly believe that the use of RedHat Linux or other equivalent OS, would have reduced operational anomalies on the Aegis system.

Q4: Form a team of 1, 2, or at most 3 students and propose a final Real-Time prototype, experiment, or design exercise to do as a team. For the summer RT-Systems class, all groups are asked to complete at time-lapse monitoring design as outlined in requirements with more details provided in this RFP. Based upon requirements and your understanding of the UVC driver and camera systems with embedded Linux (or alternatives such as FreeRTOS or Zephyr), evaluate the services you expect to run in real-time with Cheddar, with your own analysis, and then test with tracing and profiling to determine how well your design should work for predictable response. You will complete this effort in Lab Exercise #6, making this an extended lab exercise, and ultimately you will be graded overall according to this rubric and you must submit a final report with Lab Exercise #6 as outlined here. a) Your Group should submit a proposal that outlines your response to the requirements and RFP provided. This should include some research with citations (at least 3) or papers read, key methods to be used (from book references), and what you read and consulted. b) Each individual should turn in a paragraph on their role in the project and an outline of what they intend to contribute (design, documentation, testing, analysis, coding, drivers, debugging, etc.)

Solution:

Project Name: Time-Lapse Image Acquisition [16]

Developer: Poorn Mehta

Overview: This project aims to use Jetson Nano development board, running Ubuntu 18.04, along with a UVC supported Camera Logitech C920s, to create a real time system that captures images with a fixed interval (either 1fps or 10fps), without having jitter accumulation over a large period of time. The Rate Monotonic theory will be used to manage at least 2 services in real time sharing a CPU core, while some other functionalities will be handled on different CPU cores. The project will be attempted to support the remote uploading and downloading of the captured frames, to enable the capturing hardware for long continuous operations without worrying about the memory usage.

Minimum Requirements:

- Resolution is set to at least 640x480 for all captured frames.
- Individual frames are acquired (and not a video stream) from the camera.
- The system is able to capture unique, and clear images at 1Hz, indicating position of second indicator in an analog clock. This test must be passed for all 1800 frames captured during 30minutes of runtime.
- The captured image must be at least in Greyscale (P5 – PGM) format.

- Every stored image file must have a timestamp included in its header, which indicates the time at which the image was generated from the captured frame.
- The platform name on which the code was executed, must also be embedded in each image file headers.
- All captured PPM images to be converted in a video stream (for example, AVI) so that it can be played for easier verification.

Additional Goals/Targets:

- A physical process – melting of an ice cube, is observed along with the analog clock, with 1fps framerate, for verification.
- Sending files to a remote computer (which is assumed to have large memory) using sockets, to be able to store many more images without worrying about onboard memory of the system.
- Ensuring that this file sharing doesn't disturb the original time lapse requirements.
- Run the project at 10fps, and verify that the frames captured have no jitter.

Details on my approach:

- I will be using V4L2 (UVC driver) directly using C, instead of utilizing OpenCV.
- At least 2 services will be running on same CPU core, and be managed under RM in real time: (1) Frame Capture, and (2) Image Processing (of some kind, maybe YUV to RGB, contrast adjust, etc.)
- The storing of images on system memory (SD card in this case) will be decoupled from image capturing processes, using circular buffer.
- I am not including Cheddar analysis right now because the C of the services can only be known after intensive testing. I have this data from previous assignment, however in that the I/O was not decoupled, and therefore that data can't be used.
- Other services that will be running (might be on other CPU) – would be storing of images on local memory, and ideally a socket service to transfer data over the ethernet.
- Syslog will be used extensively with event names and timestamps for troubleshooting.

Research/Citations:

- Here are few of the webpages which I used to arrive on the details of this project: [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28]

Tentative Plan:

- Complete first revision of architecture planning (block diagram, software flow chart, identification of services, allocating resources, initial tests to verify assumptions etc.) by 28th July
- Get basic framework for minimum requirements up and running. Jitter is allowed at this point, but functionally incorrect output is not. To be done by 31st July
- Remove jitter, implement frame select function (either at startup, or in runtime). This will satisfy all minimum requirements. Complete this by 3rd August.

    - Test the system rigorously, with external clock on 1Hz, to ensure that everything is working as it should. No exceptions allowed beyond this point. To be achieved by 8th August.
    - Prepare a detailed report on the project by 9th August.
- Work on additional goals as much as possible, till 15th August.

Note: All of the above work will be completely done by me, and no one else is going to be working alongside with me on this project.