IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 12 February 2023, accepted 4 March 2023, date of publication 27 March 2023, date of current version 11 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3262410

RESEARCH ARTICLE

Fault Analysis and Mitigation Techniques of the I2C Bus for Nanosatellite Missions

AMINA ALBALOOSHI[1], ABDUL-HALIM M. JALLAD[2,3], (Member, IEEE), AND PRASHANTH R. MARPU[4], (Senior Member, IEEE)

[1]National Space Science Agency, Manama 51115, Bahrain
[2]Department of Electrical Engineering, United Arab Emirates University, Al Ain, United Arab Emirates
[3]National Space Science and Technology Center, United Arab Emirates University, Al Ain, United Arab Emirates
[4]Group 42, Abu Dhabi, United Arab Emirates

Corresponding author: Abdul-Halim M. Jallad (a.jallad@uaeu.ac.ae)

This work was supported in part by the Startup Grant Funded from UAE University under Grant 12N096.

ABSTRACT Despite the reliability concerns that are associated with the I2C bus, it is still one of the most popular on-board data busses to be used in nanosatellite missions. This paper provides a detailed fault analysis for the I2C bus in the context of nanosatellite missions, and consequently investigates and proposes potential mitigation techniques. The failure of the I2C bus is a risk that most CubeSat missions has to deal with, as the related bus failures can cause some catastrophic failures. Therefore, this study analyzes the I2C bus characteristics, the hardware and software requirements, and the key factors leading to I2C bus failure. By conducting experimental testing using the appropriate hardware and software to construct a comprehensive list of I2C bus requirements, characteristics, and failures. Based on the experimental testing possible mitigation approaches are proposed and finally a qualitative risk analysis is delivered to measure the impact of the methods on the overall mission success. The study shows high influence of the I2C bus on the CubeSat health and mission success, thus emphasizing on the importance of design considerations to reduce missions' risk level, as well as counting for runtime failures that can occur during mission operation.

INDEX TERMS Nanosatellites, CubeSats, I2C bus, reliability, fault mitigation.

I. INTRODUCTION

In the late 1990s Polytechnic State University and Stanford University introduced a new concept of satellites, that are built according to the CubeSat Standard Specification. The specifications included the definition of various mechanical and electrical standards that have helped in the rapid spread of CubeSats around the world. The concept of CubeSat is to have a class-based design with standard unit, one unit is referred to as 1U and has dimensions of 10 × 10× 10 cm and not to weigh more than 1.33 kg. However, the design of CubeSats is not constrained by one unit, as CubeSat designers have the choice of building bigger CubeSats that consist of multiple units 2U, 3U, 6U, etc. This concept made it easier and less costly to build satellites, and hence enabled universities and educational institutions to join the field alongside governments, military, space agencies and huge commercial firms [1], [14].

Satellite reliability is an extremely critical aspect in the overall design of any space mission; since it has direct impact on the mission success, partial mission success or even catastrophic failures that lead to CubeSat loss. Several missions have been reported to have had such failures such as: CP4, Delfi-c3, Delfi-n3xt. Thus, every aspect of the CubeSat mission design must be carefully considered for reliability. This paper deals with the on-board data busses commonly deployed on modern CubeSat missions.

The I2C bus is one of the most commonly employed data busses on-board CubeSat missions [2]. I2C Bus failures are difficult to detect and resolve making it essential for CubeSat developers to account for them during the design phase. Several previous missions with catastrophic failures hypothesized that the mission failure was due to the I2C bus, MYSat-1 CubeSat launched by Khalifa University [20], lost communication with the ground station for more than

The associate editor coordinating the review of this manuscript and approving it for publication was Claudio Zunino.

a year, with a most likely cause of failure to be due to a I2C bus-related failure. Another satellite, MeznSat, also faced in-orbit issues that have been analyzed to be related to the I2C bus [21].

Researchers proposed various approaches to increase I2C bus reliability and robustness against different failures, applying either hardware or software changes to the original I2C bus design. Previous work on I2C reliability involving hardware enhancements included the addition of hardware to control devices' availability on the bus [11], or to avoid address conflicts [8]. Also, previous work has been reported on adding a hardware circuit to isolate the I2C nodes from the bus in the event of device failure [16], [22]. This approach works on the prevention of having the faulty nodes holding the bus lines permanently and hence causing the whole bus to malfunction. On the other hand, software modifications were implemented to avoid I2C bus lockups, and to alert I2C leader of the occurred fault [12].

In this paper, we present an analysis of the suspected faults and failures of the I2C data bus in CubeSat projects. In the analysis, we list the expected I2C data bus threats over CubeSats' mission success, where each of the threats is further analyzed by experimental testing to detect the root cause of fault or failure. The goal of this study is then to propose mitigation methods to terminate those threats or reduce their impact on the missions' success.

The literature lacks a comprehensive review and practical analysis of the failure risks associated with the I2C bus, particularly in relation to its use in nanosatellite space missions. This has been noticed by researchers as reported in some recent publications [22]. The contribution of this paper is that it precisely presents and analyzes possible failure risks of the I2C bus and mitigation methods of these failures. We emulate in the lab the identified risks and their consequences on the bus and hence on the mission. We identify the level of each of the identified risks and propose and evaluate the mitigation techniques for each of these risks. This will enable researchers and designers of nanosatellite missions to improve the reliability of their missions, given that the I2C bus reliability is central to the success of some of those missions. To the best of our knowledge, such a practical risk analysis study has not been reported previously in the literature. Thus, this paper provides a guide to CubeSat developers, on how to increase their CubeSats reliability and robustness against failures, when utilizing the I2C bus. By following the guideline provided and considering the risk analysis conducted; the I2C bus failures impact are either reduced or eliminated.

The remainder of this paper is organized as follows: Section II provides a brief explanation about the I2C bus, Section III provides the related work conducted by previous researchers, Section IV presents the methodology used to conduct this study, Section V discusses the results attained from this research, Section VI presents the practical implication for implementing the I2C bus in CubeSats, and the final section provides the conclusions.

II. THE INTER-INTEGRATED CIRCUIT (I2C) BUS

A. OVERVIEW OF THE BUS

Inter-Integrated Circuit (I2C) is a data bus used to handle data transmissions between peripherals, originally designed for short distance communications within a single device. The I2C bus has gained its popularity due to its simplicity and low cost [3]. The I2C bus allows communication between low-speed peripheral ICs to processors and microcontrollers in short distances [2], enabling a variety of different topologies such as single leader to single/multi followers or multi leaders to multi followers. Manufacturers prefer I2C over other data buses for its low manufacturing cost, prioritizing the cost of the data bus over the speed [4].

The I2C protocol adopts a simple bidirectional communication concept as shown in Figure 1, in which only two wires are required for connecting all devices on the bus [5]. The two wires are Serial Data Line (SDA) and Serial Clock Line (SCL), each carrying different type of data; the SDA line is a bidirectional line which carries the data between the primary and secondary, SCL line is a one directional line which carries the clock signal from the primary device [5]. The two wires are connected to a high voltage through pullup resistors, which determine the bus speed, and the typical I2C bus pullup resistor values are 2 k Ω for the fast mode with speed of 400 kbps and 10 k Ω normal mode with speed of 100 kbps speeds, respectively [6]. The data transmission on the I2C bus is controlled by start and stop conditions sent by the master, both start and stop signals can be identified by the change of the SDA line while the SCL line is held high, start condition is a falling edge of a pulse on the SDA line and stop condition is the rising edge of a pulse on the SDA line.

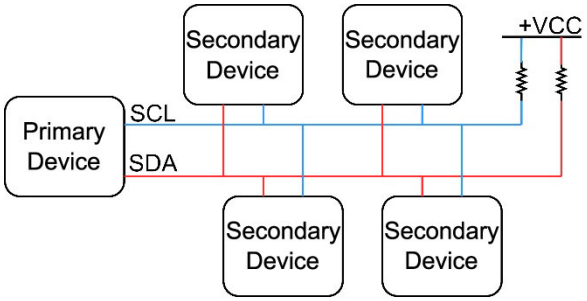

FIGURE 1. The Top-level architecture of the I2C Bus.

The I2C protocol defines a unique address for each slave with a size of 7-bits. The primary device is the only peripheral that does not require an address for it being the only initiator of all transactions. The protocol divides the data transmitted on the SDA line to bytes; the first is reserved for the slave address, second for the register address and the last of the data. All transactions on the bus start with a start signal and for each of the sent bytes the master awaits an acknowledgement from the slave it is communicating with. After successful establishment of communication, the primary or secondary starts sending the data bytes. Finally, the master sends a stop condition to terminate the session.

The simplicity of the I2C bus makes it susceptible to faults and failures especially when implemented without shielding in the harsh space environment as it is vulnerable to failures due to exposure to radiation. Radiation plays a role is causing bit-flips on the I2C bus and I2C devices, which can lead to corruption in the operating system, program, or microcontroller pointer, eventually leading to bus lockups [3], [7]. Bus lockup is the state where the I2C master loses its connectivity with all slaves and the data transmission on the bus is blocked. Bus lockups are very common I2C failure and occur mainly due to the simplicity of the bus design especially for not implementing any data integrity check nor failure recovery [3], making it a single point of failure to CubeSat missions [7]. Additionally, transient faults that occur in the I2C cores such as a bitflip in registers, and missing acknowledgements can lead to bus lockups as well. Missing acknowledgements on the I2C bus occur for various reasons, those include timing delay between SCL and SDA lines, missing or unexpected SCL pulse, incomplete 8-bit block, or missing bytes [8].

The key differences that make the I2C bus more prone to failure in comparison to the other deployed data buses in CubeSat missions, namely SPI and RS-232. Firstly, all communication between peripherals occurs on the SDA line not allowing protocol handshaking on a dedicated line. Secondly, Connecting high number of nodes on the same data bus unlike other protocols. The prior two characteristics increase the probability of having errors that can lead to faults or failures such as errors resulting from microcontroller state-machine which are then passed to the bus [2]. Also, the lack of component isolation on the I2C bus which allows faulty devices to fail the whole bus [9].

B. I2C BUS ON CubeSats FROM A RELIABILITY PERSPECTIVE

The most employed data bus in CubeSats is I2C bus, the percentage of CubeSats that employ I2C bus are 71% and 81% for launched and to be launch CubeSats accordingly. The second most employed data bus in CubeSats is RS-232 with 53% for deployed CubeSats and 45% for to be deployed CubeSats. Followed by SPI with 50% of deployed CubeSats. Other least employed data buses are namely: CAN, USB, and UART. In addition, some CubeSats use wireless data channels for intra data exchange between on-board peripherals, such as Wi-Fi and Bluetooth [2].

Besides bus popularity, bus reliability is an important aspect to investigate. The research published by Bouwmeester et al. [2] focused on I2C, SPI, and RS-232, and the number of CubeSats that employed each data bus were 37, 23, and 24 respectively. The failure tolerance techniques investigated were error signaling line, bus lockup protection, supplementary watchdog circuitry, etc. Among all missions, 42% of mission used RS-232 bus with no fault tolerance, 35% of missions using SPI, and 16% of missions using I2C; thus, it was clear that I2C bus was the highest to have failure tolerance techniques. Furthermore, the most common failure tolerance techniques implemented amongst the surveyed missions were having supplementary watchdog timer 54% of mission, implementing bus lockup protection 49% of missions, and deploying separate buses connection to different subsystems 30%.

The overall mission data bus success reported 95% of missions using RS-232 did not report any issue with the bus, followed by SPI with 94%, and finally 40% for missions using I2C. It was clear from the survey that CubeSat developers and operators had reported many issues with the I2C bus, the most reported failure was bus lockups with 43% of cases that lasted for more than 1 minute and 21% for less than one minute. Other issues with I2C bus were loss of packet transmission 21%, catastrophic hypothesis 7%, 3% for proven catastrophic failure, 3% high bit error rate and 3% performance degradation. Example of CubeSats with such I2C failures are: CP4 launched by California Polytechnic State University CubeSat that failed due to the I2C bus failure [18], Delfi-c3 of TU Delft experienced high error-rates and bus lockups, and Delfi-n2Xt failure is related to the experimental transponder activation and assumed to be the malfunctioning of the I2C buffer [19]. Several other CubeSats have also been reported the I2C bus to be the known source of mission failures [15], [16], [17].

III. RELATED WORK

The I2C bus failure detection and mitigation have attracted many researchers to investigate the topic due to its popularity and high vulnerability to failures. Researchers considered different approaches to perform integration test to test CubeSats immunity to failure by developing fault injection tool, the main interest was to insert faults in the communication channel between the CubeSat subsystems. The faults injected were presented as false values and change of electrical signals using two types on fault injection: Driver Fault and Interceptor Fault, the former simulate errors by injecting faults on the communication channel and the letter emulate a faulty communication channel by injecting faults. Authors emphasized on the importance of utilizing Failure Emulator Mechanism (FEM) to increase the CubeSats' robustness; by allowing enhancements during the development phase [10].

Other researchers investigated the vulnerabilities of I2C bus failure, and its resulting errors being generated by continuously monitoring the I2C bus using external device that sits invisible. The monitor is designed to analyze the data being transmitted over the I2C bus, understanding the packet structure and communication messages. Then it generates signals which summarizes the failures occurred on the bus and recovery actions. The monitor's implementation allows checking the success of the data transactions by calculating the checksum of the data transmitted or detecting NACK signal on the bus. Additionally, allowing fault recovery by forcing the I2C master to reset [3].

Several approaches have been applied to increase the I2C bus reliability and robustness, with the price of additional cost, rules, hardware, and processing. Ryan et al. [11] presented a designed developed by California Institute of Technology, the design has two I2C cores that regularly send messages to slaves to check if they are alive. Patrick et al. [12] proposed an approach to monitor the I2C bus for lockups and alert the master in case a lockup was detected. The work presented by [13] Intel Corporation proposed SMBus which is a protocol that runs on the I2C bus, the SMBus uses additional set of rules to control the data transmission over the I2C bus targeting various categories such: electricals, timing, protocols, and operating modes. The SMBus sets many rules that the I2C bus does not support such as minimum clock frequency, cumulative clock low extend time for slave, clock low time-out, etc. Such features give SMBus more reliability over the I2C bus but lowers the compatibility with I2C devices. Ferrando [8] presented two methods to mitigated conflicted addresses on the I2C bus, both methods require hardware to be connected in addition to the conventional the I2C bus setup. The first method utilizes additional multiplexer between the I2C bus and followers with conflicted address, the master device then enables the required slave electrically bus connecting it to the bus or isolating it. This method does not require additional pins or control logic for I2C programmable devices. The second method is adding I2C buffer this method demands additional lines and control logic depending on the devices used.

IV. METHODOLOGY

The methodology used for this study included five stages as highlighted in Figure 2.

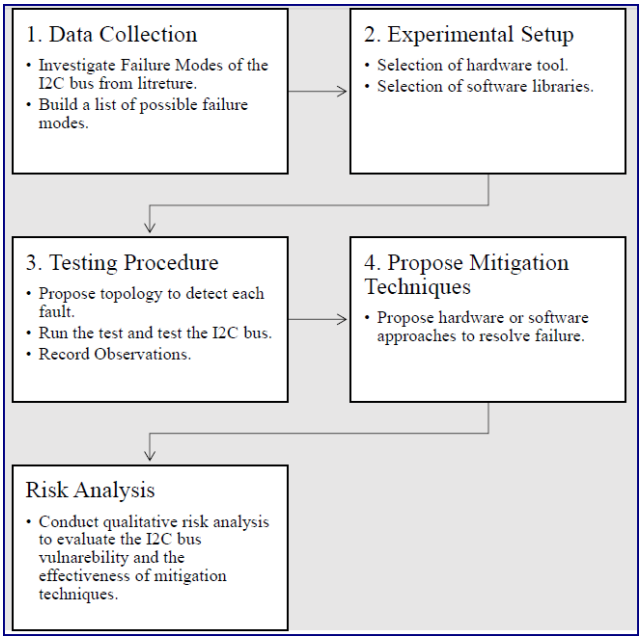

FIGURE 2. Methodology used for the Study.

A. DATA COLLECTION

The data collection in this study was mainly based on literature review and collecting observation from experimental testing. The data collection to construct the initial list of hypothesized failure modes was constructed by investigating previous CubeSat missions' failure analysis reports and published surveys. On the other hand, the data collected from the experimental testing were based on collecting observations from setups simulating the hypothesized failure modes.

B. EXPERIMENTAL SETUP

The experimental setup was constructed using selected hardware, software, and monitoring tools to conduct the testing scenarios. For each of the hypothesized failure mode different hardware and software components were utilized. The hardware boards used were Arduino Uno, Arduino Mega, and TI Tiva C Launch pad, in order to provide a variety of I2C bus implementations. For monitoring the hardware and software components of each of the failure modes deferent tools were utilized such as:

- Serial output monitor, to capture the data transmission on both primary and secondary sides.
- I2C packet sniffer, which was implemented using Arduino MEGA board, to capture I2C packets being transmitted between leader and followers and to monitor the data transmitted on the bus. Figure 3 shows the output of the packet sniffer, showing the device address, read/ write operation and finally the stream of bytes exchanged.

```
03:53:39.937 -> 31Analyzing data, number of transitions = 2873
03:53:39.973 -> Next start condition: 143
03:53:40.009 -> Dev=0x03 R Data=0x60 0x32*0x10 0x20 0x62 0x20*
```

FIGURE 3. Example of I2C packet sniffer output.

C. TESTING PROCEDURE

The testing procedure in this study followed three main steps, firstly by selecting the topology setup of the I2C bus by determining the number of slaves and the population of the devices on the I2C bus. The experimental setup is used to run a set of pre-identified tests to stress the I2C bus using different Commercial-Off-The-Shelf (COTS) development hardware. The different hardware and software selection were based on the need to fulfil the aim of each test scenario to generate a failure mode and to detect the cause of that failure mode.

D. PROPOSE MITIGATION TECHNIQUES

Based on the observations made during the experimental testing, different approaches were proposed to overcome the failure modes. Each failure mode was examined individually to propose methods utilizing hardware and/or software components to mitigate the root cause leading to that failure by either eliminating it or reducing its likelihood and consequence.

E. RISK ANALYSIS

The best way to analyze and evaluate the data collected is by conducting a qualitative risk analysis, since the data obtained from testing were mainly based on observation and no quantitative output were recorded.

Qualitative risk analysis is a helpful tool to measure the risk associated to the I2C bus failure and to help minimize the risk of failure thus increasing the I2C bus reliability.

The qualitative risk analysis conducted in this study is based on six main steps:

1. Determination of risk phase, whether it is an implementation risk that occurs in the development phase or a mission risk that occurs during operation phase.
2. Definition of risk parameter, each risk is mapped to a probability and consequence level of seriousness as shown in Table 1 and Table 2.

TABLE 1. Risk consequence.

#	Consequence	Description
1	Very Low	Minimal or no impact on mission objectives
2	Low	Minimal reduction in mission return
3	Moderate	Cannot meet full mission success
4	High	Cannot meet minimum mission success
5	Very High	Mission catastrophic

TABLE 2. Risk likelihood.

#	Likelihood	Description
1	Very unlikely	May occur in exceptional circumstances
2	Unlikely	Unlikely to occur but could occur under some circumstances
3	Possible	Moderately likely to occur
4	Likely	Will probably occurs at some time
5	Very Likely	Is expected to occur

3. Map the risk to the risk-matrix as shown in Figure 4
   a. [L] Green corresponds to low impact
   b. [M] Yellow corresponds to medium impact
   c. [H] Red corresponds to high impact

		Consequence				
		1	2	3	4	5
Likelihood	1	Low	Low	Low	Low	Med
	2	Low	Low	Low	Med	Med
	3	Low	Low	Med	Med	High
	4	Low	Med	Med	High	High
	5	Low	Med	High	High	High

FIGURE 4. Risk priority matrix.

4. Propose mitigation techniques, that can help in solving or reducing the risk. Four approaches are possible:
   - Treat: by having an action plan to take if the risk occurs which results in reducing either the likelihood, consequence, or both.
   - Terminate: by applying enough margins, perform tests and simulations to eliminate the risk.
   - Transfer: by having insurance or a third party to handle the risk.
   - Tolerate: is when no action can be taken, and the risk must be accepted.
5. Re-evaluate the risk after mitigation technique implementation, based on likelihood and consequence.
6. Defining the net risk, which is the risk remaining after the mitigation techniques is applied.

V. RESULTS

In this section, the results of the experimental testing for each of the hypothesized failures is presented, the I2C bus failures were observed either as bus lockups where the data transmission on the bus was lost, or as corrupted data where the data received by the peripheral's did not match the message sent. The most common I2C failure was bus lockups, based on the different hypnotized scenarios bus lockups occurred either due to hardware or software issue; therefore, hardware components and software tools were utilized to emulate the failure scenarios and detect the causes leading to it. The study detected four causes that lead to bus lockups: the value of the pullup resistor, infinite loops, device state, and missing acknowledgements. In addition, data corruption occurred due to conflicted addresses or loss of synchronization between devices.

The rest of the section will discuss each failure cause and explain the test conducted to detect it.

A. PULLUP RESISTOR VALUE

In this test, the pullup resistor value effect on the I2C bus behavior was tested by connecting a single master device to two slaves. The theoretical value of the pullup resistor for the experimental setup was calculated and found to fall between minimum 1.5kΩ and maximum 2.95kΩ. In this experiment the pullup resistor values were varied, and the I2C bus behavior was observed. The value that caused a bus lockup was resistor value of 100 Ω and below; the value of the resistor is subjected to the node population on the bus; thus, it differs based on the mission design. One factor to notice is that with several COTS nodes (subsystems and components) added to the spacecraft I2C bus, the value of the pull-up resistors build-up, which will impact the quality of the SCL signal as observed in the waveforms in figure 5. This leads to the malfunctioning of the bus operations.

B. INFINITE LOOPS

The I2C bus protocol software design is based on interrupts and register states, meaning that all data transmissions and

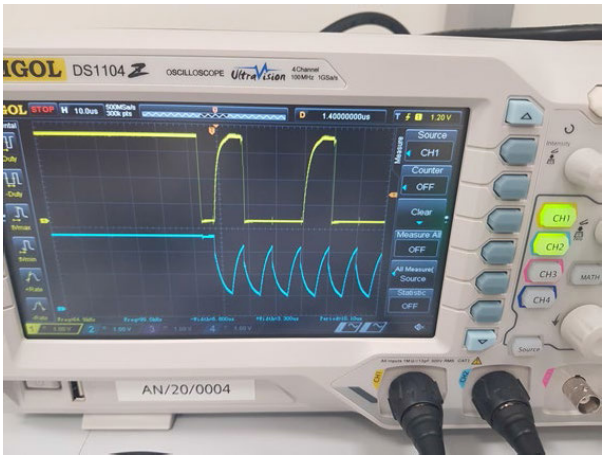

FIGURE 5. SDA (yellow) and SCL (Blue) waveforms with pullup resistor value of 100 KOhm.

control signals are being handled by the Two Wire Interface (TWI) registers. The simple design of the protocol lacks failure and error handling and sets numerous infinite loops to wait for register states to change, such as: wait for start condition as shown in eq. 1, if the device was unable to set the TWCR register then the I2C bus would lock up.

$$TWCR = (1 \ll TWINT)\,|(1 \ll TWEN)|(0 \ll TWSTO));$$
$$//StartCondition$$
$$While\,(!\,(TWCR\&\,(1 \ll TWINT)))\,;$$
$$//checkforstartcondition \quad (1)$$

Infinite loops are also implemented with acknowledgements and stop conditions, which can be extremely critical in terms of software reliability. For simple I2C implementation and utilizing non programable devices, failing to execute a start, stop, or acknowledgement statement was found to lead to bus lockups.

C. DEVICE STATE

The device state being ON or OFF while connected to the I2C bus was tested, by examining the voltage on the bus using two different I2C slaves (AVR processor on the Arduino UNO and the TM4C123 processor on the TIVA C board).

In normal cases I2C slaves can only pull the bus low to enable the bidirectional communication when transmitting data to the master. In this experiment the normal bus voltage was held high at +5V, and then disconnecting the power of the Arduino slave (OFF) while it is connected to the I2C bus, hence leading to voltage drop to less than 1V on both SDA and SCL lines as shown in Table 3. This sequence was found to cause a bus lockup. On the other hand, the same test was repeated using a TIVA C slave, the voltage on both lines remained at almost 5V after disconnection. Different hardware act differently on the I2C bus, in this case turning the Arduino board OFF while connected to the I2C bus acted as circuit ground and dropped the voltage on the bus which leaded to bus lockup. One the other hand, TIVA C is design to isolate the I2C bus from ground when it is switched OFF. Thus, slave devices should not have the ability to pull the bus low unless transmitting data over the SDA line.

D. MISSING ACKNOWLEDGMENT

Acknowledgements are an important part of the I2C protocol definition. For this test, the effect of a missing acknowledgement was investigated by disabling one acknowledgement statement from the slave controlling the bit assignments TWINT, TWEA, TWEN, and TWIE of the Two Wire Control Register (TWCR) register as shown in eq. 2.

$$TWCR = (1 \ll TWINT)\,|1 \ll TWEA|\,(1 \ll TWEN)|(1 \ll TWIE) \quad (2)$$

The failure of the slave to set the TWCR register or the loss of the acknowledgement on the SDA line led to I2C bus lockup, where the master did not have the ability to communicate with any of the other slaves on the bus.

E. CONFLICT ADDRESS

The I2C bus can be implemented by either allocating 7-bits or 10-bits for the address which allows having 128 or 1024 addresses. The limited range increases the probability of having two slaves with the same address on the bus, which leads to corrupted data transmission on the I2C bus as both slaves reply at the same time. Moreover, slaves with hardcoded addresses suffer this problem unlike I2C slaves which allow the modification of their address either by programming or having configuration pins to flip a bit or two. Conflicting addresses was found to lead to corrupted data, i.e., the data received did not match the data sent.

F. DEVICE SYNCHRONIZATION

The I2C protocol depends on clock signals from the master device to control all transactions, and synchronization faults between I2C devices happen when the master encounters a sudden restart while communicating with I2C slaves. The I2C protocol does not define device state messages, thus, the slave devices do not get updated by the sudden restart. The observation made by this experiment is that slaves continue sending the data from last stopped byte, which results in having data at the master end that does not match the originally sent, hence leading to data corruption.

TABLE 3. SDA and SCL lines voltage.

Device State	SDA (v)	SCL (v)	Lines Connected
ON	4.7	4.7	All
OFF	0.96	0.96	GND, SDA, SCL
OFF	1.3	0.02	SDA, SCL
OFF	4.9	4.9	None

TABLE 4. I2C qualitative risk analysis.

Risk	Type	Consequence	Likelihood	Level	Mitigation	Consequence	Likelihood	Level	Net risk
Faulty device (OFF) on the I2C bus.	M	5	3	H	Terminate: Employ bus isolators for devices cause the risk.	1	1	L	Reduced consequence and likelihood of occurrence, under normal circumstances.
Data request with high number of bytes.	M	3	4	M	Terminate: Test slaves' capabilities and apply control over the maximum data size.	1	1	L	Reduced consequence and likelihood of occurrence, when transmitting within calculated data size.
Missing acknowledgment from a slave.	M	5	5	H	Treat: Implement enforced bus release with loop break.	2	2	L	Fault may occur but the probability of the bus getting locked up is much lower.
Unsynchronized devices on the I2C bus.	M	3	4	M	Treat: Schedule resets and reset slave after OBC reset.	2	2	L	Reduced consequence and likelihood of occurrence.
Devices with conflicted addresses on the I2C bus.	I	4	1	L	Terminate: Test slaves' address uniqueness prior to launch and isolate conflicted address.	1	1	L	Fault terminated unless hardware damage happens after deployment.
High number of I2C nodes.	I	3	3	M	Terminate: Design bus nodes number based on capacitance with margin.	2	1	L	Fault terminated.
Lengthy I2C bus line.	I	3	3	M	Terminate: Test wire length with margin.	1	1	L	Fault terminated.
Software infinite waiting loops.	M	4	5	H	Treat: Implement timers to break loops, add watchdog timers.	2	2	L	Reduced consequence and likelihood of occurrence.
High pull up resistor value.	I	4	3	M	Terminate: Employ pullup resistors with enough margin.	1	1	L	Fault terminated, when implemented with enough margin.
Low resistor value.	I	4	3	M	Terminate: Employ pullup resistors with enough margin.	1	1	L	Fault terminated, when implemented with enough margin.

VI. DISCUSSION AND PRACTICAL IMPLICATION

The usage of the I2C bus on-board CubeSats can lead to catastrophic failures for missions, due to the I2C bus related risks and the unpredictability of their occurrence. Therefore, constructing a robust risk mitigation plan for CubeSat missions is essential, considering the different development phases of the project and the various aspects that must be counted for. Thus, significantly reducing the impact of the risks or the frequency of occurrence. A summary of the risks associated with the I2C bus and their respective mitigation techniques is presented in Table 4.

During the design phases, CubeSat developers should examine both: the individual I2C slave device/nodes and the design of the I2C bus.

Firstly, the I2C slave addresses must be ensured to be unique; in case of encountering conflicted address, developers are expected to resolve the issue by either implementing hardware or software solution. The simplest approach with programmable devices is to reprogram the address, but in many cases this solution is not feasible for I2C slaves as the simplicity of their design does not support programmability. Thus, the issue of having slaves with conflicted address can be solved using hardware solutions such as: exchanged the conflicted slave with other slaves, connect slaves with conflicted addresses to different I2C busses, connect conflicting slaves via multiplexers [8], or connect conflicted slaves through I2C buffers [8]. Hardware solutions demand additional hardware in addition to software modification, which adds complexity to the solution.

Second aspect to be investigated during the design phase is the device electronic design characteristics, mainly the effect of the slave on the I2C bus in case of having faulty power supply to that slave. Some slaves pull the bus low in case of losing the power supply, this failure can be avoided by isolating the slave from the I2C bus by utilizing additional hardware, such as connecting the slave device an I2C buffer.

Thirdly, CubeSat developers should always ensure appropriate voltage levels on SDA and SCL lines. The SDA and SCL lines should be held high voltage when idle, and this is achieved by integrating pullup resistors. Some I2C slaves come with integrated internal pullup resistors but connecting an external pullup resistor ensures a more robust design.

Also, considering the I2C slave capabilities is critical for CubeSat software design, because the consequence of exceeding the slaves' capabilities would lead to getting unexpected errors. For example, the data buffer size, the master device should not transfer data at a rate that would overload the slave buffer size. Thus, the software design should adhere to the slaves' specifications.

Additionally, the I2Cbus protocol suffers several software vulnerabilities which should be considered during the design phase. The I2C protocol design assigns several infinite loops, the control signals on the bus depend on register assignments and the devices wait indefinitely for the register assignments to succeed, and the failure of assigning the register leads to bus lockup. Thus, developers should implement timers to exit loops and enforce the slave devices to release the bus.

Also, the I2C protocol defines waiting time for acknowledgement on both master and slave sides, if the acknowledgment is lost or corrupted during the transmission then the I2C bus gets locked because the receiver device will keep on waiting for the lost acknowledgement. Thus, software development shall consider releasing the bus after a predefined wait-time.

Furthermore, some I2C slaves suffer from synchronization with the master device, this happens after an unscheduled master restart. The unsynchronized slave devices then transmit faulty values, in which it can be overcome by having a scheduled reset to the slave devices after the master's reset.

In addition, CubeSat developers must ensure having watchdog timer on-board, in which it can reset the devices in case of exceeding an idle time of no communication happening in the I2C bus. And it is important to note the assigned wait time of the devices should not exceed the idle time of the watchdog timer.

Besides the key factors that ensure the reliability of the I2C design it is important to be consider future run-time risks which should be addressed at the implementation phase.

VII. CONCLUSION

Although the I2C bus is commonly used on-board CubeSat missions, and is deemed responsible for several mission failures, there is a lack of a comprehensive study on the possible causes of I2C related failure, and their possible mitigation techniques. Previous research lacked the investigation of the root causes of I2C failure, where only few failure triggers were identified. In this paper, we investigated some of the suspected I2C failures that were encountered by past CubeSat missions, and hence to confirm the occurrence of the hypothesized failures by conducting experimental testing. The paper shows that the I2C bus design suffers hardware and software issues when implemented in CubeSats, due to various reasons including, bus simplicity (and hence lack of fault detection mechanism), the harsh space environment, and the lack of shielding. Previous research identified catastrophic failures of CubeSat missions with the hypothesis that the failure occurred due to I2C bus failure, yet no further risk analysis was conducted to measure the impact of I2C bus failure on the overall CubeSat mission success. In this paper, out of ten identified risks related to the I2C bus, three of them were identified as high-risk vulnerabilities, six as medium-risk and one as low-risk. Faulty on-board devices are of particular concern, as they are highly likely and can lead to serious consequences on the mission. The paper proposes different mitigation methods, some that consent with other research proposals, and other new proposed approaches, which CubeSat mission developers should consider when developing CubeSat; to eliminate failures and/or reduce their impact on CubeSat missions. Additionally, the paper also presents a risk mitigation procedure for CubeSat developers to consider when utilizing I2C bus in their missions.

REFERENCES

[1] M. Swartwout, "The first one hundred CubeSats: A statistical look," *J. Small Satell.*, vol. 2, no. 2, pp. 213–233, 2013.
[2] J. Bouwmeester, M. Langer, and E. Gill, "Survey on the implementation and reliability of CubeSat electrical bus interfaces," *CEAS Space J.*, vol. 9, no. 2, pp. 163–173, Jun. 2017.
[3] V. Carvalho and F. L. Kastensmidt, "Enhancing I2C robustness to soft errors," in *Proc. IEEE 8th Latin Amer. Symp. Circuits Syst. (LASCAS)*, Feb. 2017, pp. 1–4.
[4] S. Van Der Linden, J. Bouwmeester, and A. Povolac, "Design and validation of an innovative data bus architecture for CubeSats," in *Proc. Reinventing Space Conf.*, 2016, pp. 1–13.
[5] J. Valdez and J. Becker, "Understanding the I2C bus," Texas Instruments, Dallas, TX, USA, Tech. Rep. SLVA704, 2015. [Online]. Available: https://www.ti.com/lit/an/slva704/slva704.pdf
[6] R. Arora, "I2C bus pullup resistor calculation," Texas Instruments, Dallas, TX, USA, Tech. Rep. SLVA689, 2015.
[7] H. Askari, E. W. H. Eugene, A. N. Nikicio, G. C. Hiang, L. Sha, and L. H. Choo, "Software development for Galassia CubeSat—Design, implementation and in-orbit validation," in *Proc. Joint Conf. 31st Int. Symp. Space Technol. Sci. (ISTS)*, 2017, pp. 1–8.
[8] M. Ferrando, "Troubleshooting I2C bus protocol," Texas Instruments, Dallas, TX, USA, Tech. Rep. SCAA106, 2009.
[9] L. Kepko, L. S. Soto, C. Clagett, B. Azimi, D. Chai, A. Cudmore, J. Marshall, and J. Lucas, "Dellingr: Reliability lessons learned from on-orbit," in *Proc. Conf. Small Satell.*, 2018, pp. 1–14.
[10] C. L. G. Batista, E. Martins, and M. D. F. Mattiello-Francisco, "On the use of a failure emulator mechanism at nanosatellite subsystems integration tests," in *Proc. IEEE 19th Latin-Amer. Test Symp. (LATS)*, Mar. 2018, pp. 1–6.
[11] F. Ryan, D. Leonard, H. R. L. Robert, and C. Savio, "I2C bus protocol controller with fault tolerance," U.S. Patent 6 728 908, Apr. 27, 2004.
[12] B. Patrick, H. Daniel, L. Vinh, W. Kirby, and W. Lee, "Systems and methods for correcting errors in I2C bus communications," U.S. Patent 02 400 19A1, Oct. 11, 2007.
[13] *System Management Bus (SMBus) Specification Version 2.0*, SBS Implementers Forum, Aug. 2000.
[14] J. Bouwmeester and J. Guo, "Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology," *Acta Astron.*, vol. 67, nos. 7–8, pp. 854–862, 2010.
[15] M. Noca, F. Jordan, N. Steiner, T. Choueiri, F. George, G. Roethlisberger, N. Scheidegger, H. Peter-Contesse, M. Borgeaud, R. Krpoun, and H. Shea, "Lessons learned from the first Swiss pico-satellite: SwissCube," in *Proc. 23rd Annu. AIAA/USU Conf. Small Satell.*, 2009, pp. 1–20.
[16] N. Cornejo, J. Bouwmeester, and G. Gaydadjiev, "Implementation of a reliable date bus for the Delfi programme," in *Proc. 7th Int. Symp. Int. Acad. Astronaut. (IAA)*, Berlin, Germany, May 2009, pp. 4–8.
[17] E. Oland, "The HiNCube student satellite—Lessons learned," in *Proc. 7th Int. Conf. Recent Adv. Space Technol. (RAST)*, Jun. 2015, pp. 429–432.
[18] G. Manyak and J. M. Bellardo, "PolySat's next generation avionics design," in *Proc. IEEE 4th Int. Conf. Space Mission Challenges Inf. Technol.*, Aug. 2011, pp. 69–76.
[19] J. Bouwmeester, L. Rotthier, C. Schuurbiers, W. Wieling, G. Van Der Horn, F. Stelwagen, E. Timmer, and M. Tijssen, "Preliminary results of the Delfi-n3Xt mission," in *Proc. 4S Symp.*, Porto Portugal, 2014, pp. 1–15.
[20] B. O. Alnaqbi, "The first UAE multi-disciplinary space program," in *Proc. 23rd IAA Symp. Small Satell. Missions*, 2016, pp. 1–7.
[21] A.-H. Jallad, P. Marpu, Z. A. Aziz, A. Al Marar, and M. Awad, "MeznSat—A 3U CubeSat for monitoring greenhouse gases using short wave infra-red spectrometry: Mission concept and analysis," *Aerospace*, vol. 6, no. 11, p. 118, Oct. 2019, doi: 10.3390/aerospace6110118.
[22] M. Holliday, Z. Manchester, and D. G. Senesky, "On-orbit implementation of discrete isolation schemes for improved reliability of serial communication buses," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 58, no. 4, pp. 2973–2982, Aug. 2022, doi: 10.1109/TAES.2022.3142713.

AMINA ALBALOOSHI was born in Bahrain. She received the B.Sc. degree in computer engineering from the University of Bahrain, in 2016, and the M.Sc. degree in engineering systems and management with the Space Systems and Technology Center, Khalifa University, in 2021. From 2018 to 2019, she was a Network Engineer with Batelco. Since 2019, she has been a Senior Space Engineer with the National Space Science Agency, Bahrain. Her research interests include CubeSat development, CubeSat risk management, and failure detection isolation and recovery.

ABDUL-HALIM M. JALLAD (Member, IEEE) received the B.Eng. degree from the University of Kent, U.K., in 2003, and the Ph.D. degree from the University of Surrey, U.K., in 2009. At the University of Surrey, he was a member of the Surrey Space Centre, where he was involved in several research and development projects in collaboration with Surrey Satellite Technology Ltd., (SSTL), a World Leader in the development of small satellites. Currently, he is with the Department of Electrical Engineering and the National Space Science and Technology Centre, United Arab Emirates University. Prior to that, he was the Director of the Center of Information, Communication and Networking Education and Innovation (ICONET), American University of Ras Al Khaimah (AURAK). His research interests include embedded systems, the Internet of Things, system-on-chip designs, spacecraft on-board data handling, middleware designs, VLSI designs, and reconfigurable architectures. He received several academic achievement prizes.

PRASHANTH R. MARPU (Senior Member, IEEE) received the M.Sc. degree in wireless engineering from the Technical University of Denmark, in 2006, and the Ph.D. degree from TU Freiberg, Germany, in 2009. He was an Associate Professor with the Department of Electrical Engineering and Computer Science, Khalifa University of Science and Technology (KUST), Abu Dhabi, United Arab Emirates. He was involved in designing and building small satellites as a project manager of four satellite projects. He is currently the Technical Lead of Space Program with Group 42, Abu Dhabi. His research interests include space systems, remote sensing, and machine learning.