An Evaluation of the CAN bus for Use on the Europa Lander Motor Controller

Hieu Tran, Gary Bolotin, Ben Cheng, Allen Sirota, Malcolm Lias
Jet Propulsion Laboratory, California Institute of Technology.
4800 Oak Grove Dr.
Pasadena, CA 91109
Hieu.C.Tran@jpl.nana.gov, Gary.Bolotin@jpl.nasa.gov,
Ben.Cheng@jpl.nasa.gov, Allen.R.Sirota@jpl.nasa.gov,
Malcolm.Lias@jpl.nasa.gov

Summary — **This paper will present the application of the Controller Area Network (CAN) for communication between the Motor Control Processor and the Europa Lander Motor Control Cards. The Europa Lander preproject team is concerned about lowering the estimated mass, volume, and power of the mission concept so that it could maximize its science return. The motor controller uses multiple Motor Control Cards to control the 24 motors that will be operating in the extremes of the Europa environment. The CAN bus was considered because of its multi-drop nature, low power interface, and simple communication protocol. Beyond communication with the motor controller cards we also use the CAN bus interface to gather telemetry from the Power Conversion Card. Our system requires the collection of telemetry at a rate of 64Hz from the Power Conversion Card and all active motor controllers along with the sending of control packets and the receipt of control parameters at 512 Hz. In the paper we will address the ability of the CAN bus to meet these requirements along with our Arduino based test system used to learn about the CAN bus standard and verify our analysis.**

Table of Contents

1. Introduction

The Europa Lander is a proposed NASA astrobiology mission to Europa, an icy moon of Jupiter. If funded and developed as a Flagship mission, it would be launched as soon as 2025 to complement the studies by the Europa Clipper orbiter mission and perform analyses on site. The objectives of the mission concept are to search for biosignatures at the subsurface ≈10 cm, to characterize the composition of non-ice near-subsurface material and determine the proximity of liquid water and recently erupted material near the lander's location. [1]

The savings in the predicted mass and volume of the Europa Lander was the leading driver that led to a miniaturized motor controller currently planned for use on the mission. As illustrated in figure 1, the design consists of a computer card along with enough Motor Control Cards (MCC) necessary to control 12 motors. Each motor card can control up to three motors. Only one motor can run at a time per card. Those Motor Control Cards along with the Power Conversion Card (PCC) communicate to Motor Control Processor (MCP) by means of the CAN bus interface. [2]

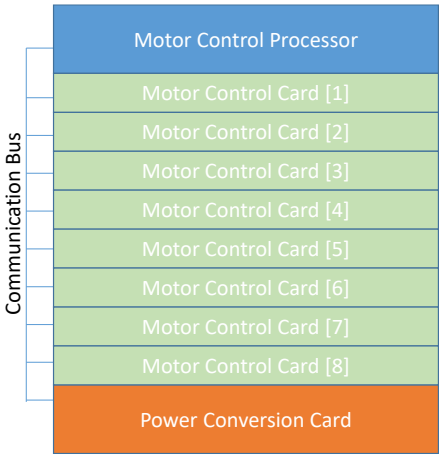

Figure 1: Europa Lander Motor Controller

2. Background of CAN bus

A Controller Area Network (CAN) bus is a robust bus standard designed to allow microcontrollers and devices to communicate with each other's applications without a host computer. The CAN bus was originally designed for the automotive industry. The CAN bus has been used in various fields of industrial automation, marine, medical equipment, industrial equipment, and the aerospace industry. [3] The CAN bus communication protocol is widely recognized for its high performance and reliability and has been used in space flight applications. The benefits of CAN bus are speed, flexibility, and reliability. The CAN bus offers signal transfer rates up to 1 Mbps with a cable length of up to 40m. The CAN bus is a flexible standard because all nodes are equal; there is no master slave relationship. Also, as shown in figure 2, the CAN bus is reliable because CAN-Hi and CAN-Lo independently carry the data, the bus can still function if one signal line is broken, albeit with lower noise rejection. [4]

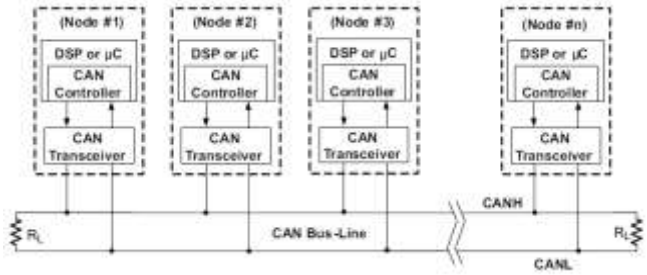

Figure 2: Typical CAN bus Implementation [5]

When data are transmitted over a CAN network no individual nodes are addressed. Instead the message is assigned an identifier which defines the message contents and the message priority. As shown in figure 3, 4, and 5, a CAN message consists of an identifier field and the data field. The identifier field can have 11 bits (Standard CAN) or 29 bits (Extended CAN). The data field can have a maximum of 8 bytes. Also, as shown in table 1, full length of the standard CAN message is 108 bits and extended CAN message is 128 bits [6].

Figure 3: Standard CAN: 11-Bit Identifier (2.0A) [5]

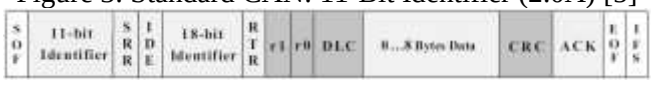

Figure 4: Extended CAN: 29-Bit Identifier (2.0B) [5]

Figure 5: Standard CAN FD 11-Bit Identifier [7]

Table 1: The Bit Fields of Standard CAN (2.0A) [5]

Field Name	Sub-Field	Length(bits)	Purpose
Start of Frame	SOF	1	Indicates the start of frame on CAN Bus(must be dominant (0))
Arbitration Field	Identifier	11	Decides the Message Priority(Arbitration) on CAN BUS
	RTR	1	Differentiate between Remote Frame or Request frame
Control Field	IDE	1	Tells about frame format : standard or extended For Standard : Always Dominant(0) For Extended : Always Recessive(1)
	R0	1	Reserved(must be dominant 0)
	DLC	4	Data length on the CAN bus
Data Field	D0-D8	64	Data
CRC Field	CRC	15	CRC
	CRC Delimiter	1	Must be recessive(1)
Ack field	Ack	1	Acknowledgment by receiving node.
	Ack Delimiter	1	Must be recessive(1)
End Of Frame	EOF	7	Indicates end of current frame(Must be recessive(1))

On the CAN bus, all nodes are equal. There is no such thing as a bus master. This is made possible by the arbitration process during the idle time between messages. The priority of CAN bus message is based on the data and not where it came from.

When there are two or more messages sent simultaneously, the arbitration process causes controllers with lower priority messages to enter the receiving mode. The unsung controllers wait until the end of the transmission intermission field before they attempt to communicate again. As soon as the bus is detected as idle, the CAN node sends an SOF (Start of Frame) bit by putting a dominant (low) level onto the bus. Every other node in the network, that did not request bus access, will immediately switch to a receiving mode. Then, the CAN controller sends the message ID. Then, all the CAN controller of all nodes will compare their output signal with the actual bus level at the end of each bit cycle. In the meantime, any node will lose the arbitration, in case it did send a recessive level (high) and detects a dominant (low) bus level. Consequently, all those nodes will switch into receiving mode. If the node has finished sending all arbitration bits (message ID plus RTR) without losing the bus arbitration, it will transmit the rest of the message. At this time all other CAN nodes in the network will have switched to receiving mode.

There are three variants of the CAN bus standard that were evaluated; Standard CAN(2.0A), Extended CAN(2.0B) and the Standard CAN with Flexible Data (Standard CAN FD). [7]

Standard CAN(2.0A) has the following features:

- 11-bit Identifier
- Maximal CAN message payload can only reach 8 bytes, so full standard CAN message have total 108 bits.
- Max speed 1Mb/s

Extended CAN(2.0B) has the following features:

- 29-bit Identifier
- Maximal CAN message payload can only reach 8 bytes, so full extended CAN message have total 128 bits.
- Max speed 1Mb/s

Standard CAN FD has the following features:

- 11-bit Identifier
- The allowed message payloads of CAN FD are 1/2/3/4/5/6/7/8/12/16/20/24/32/48/64 bytes
- CAN FD can improve the transmission bit-rate of CAN to as much as 12 Mbps

This paper presents and evaluation of Standard CAN(2.0A). The other buses present additional features and capabilities that will need to be evaluated in the future.

3. Approach

The SPI bus, SpaceWire, and CAN bus were all considered as a means of communication between our cards. The CAN bus was considered for evaluation because of its multi-drop nature, low power interface, and simple communication protocol. This paper outlines this evaluation.

The implementation of CAN bus interface into the Europa Lander Motor controller required us to map our telemetry and command requirements to the CAN bus protocol. This required a specific protocol for the Motor Control Cards and another one for the Power Conversion Card. We then did an analysis of the bus bandwidth required relative to the capability of the bus standard.

To learn about the bus and to test out the mapping of our protocol to the CAN bus standard we used an Arduino based system for testing. This enabled us to get experience about the bus and to verify our assumptions and models.

4. MCC & PCC Data Requirements

Before beginning our analysis, we started by first compiling the data requirements between the Motor Control Processor (MCP) and the Motor Control Cards (MCC). The data requirements can then be mapped onto the different standard CAN bus protocols.

As illustrated in table 2, the MCCs are required to send back resolver position and velocity, motor phase currents to the MCP at a rate of 64hz. This data is used for telemetry collection. The MCCs need to send primary resolver position information along with position commands at a rate of 512hz. This data is used for control of the motors. The requirements list the data required to control 1 motor.

Table 2: MCC Data Requirements (1 Card)

	Register Data	Bits	Hz	Total	Direction
Motor 1 or Motor 2 or Motor 3	Resolver Position	16	64	1024	To MCP
	Resolver Odometry	16	64	1024	To MCP
	Resolver Velocity	16	64	1024	To MCP
	Current Phase A	16	64	1024	To MCP
	Current Phase B	16	64	1024	To MCP
	Current Phase C	16	64	1024	To MCP
	PWM Duty Cycle	16	64	1024	To MCP
	Kirchoff Current	16	64	1024	To MCP
	Bridge Current	16	64	1024	To MCP
	Compesation Current	16	64	1024	To MCP
	Current Limit	16	64	1024	To MCC/FGPA
	Current Command	16	512	8192	To MCC/FGPA
	LVDT Postion	16	512	8192	To MCP
	LVDT Odometry	16	512	8192	To MCP
	LVDT Velocity	16	512	8192	To MCP

As illustrated in table 3, the Power Conversion Card (PCC) is required to send back the status of the voltage and current for each of its output power rails. In addition, it sends back status information on its assessment of its own health. This data is required to be transferred back to the processor every 64 Hz.

Table 3: Power Conversion Card Data Requirements

	Register Data	Bits	Hz	Total	To
Power Card	Temperature PRT 1	16	64	1024	to MCP
	Temperature PRT 2	16	64	1024	to MCP
	28V Power Monitor	16	64	1024	to MCP
	15V Power Monitor	16	64	1024	to MCP
	12V Power Monitor	16	64	1024	to MCP
	5V Power Monitor 1	16	64	1024	to MCP
	5V Power Monitor 2	16	64	1024	to MCP
	5V Power Monitor 3	16	64	1024	to MCP
	3.3V Power Monitor	16	64	1024	to MCP
	2.5V Power Monitor 1	16	64	1024	to MCP
	2.5V Power Monitor 2	16	64	1024	to MCP
	Status/Fault Word 1	16	64	1024	to MCP
	Status/Fault Word 2	16	64	1024	to MCP
	TBD Word 1	16	64	1024	to MCP
	Command Word 1	16	64	1024	to MCC/FPGA
	Command Word 2	16	64	1024	to MCC/FPGA

5. CAN Bus 2.0A Implementation

I. Adaptation to Standard:

Based on the requirement for the Europa Lander, the motor controller uses multiple Motor Control Cards to

control the 24 motors, and each Motor Control Card can control 3 motors. The CAN bus for our system operates at 1 MHz. As illustrated in figure 6, each of the cards connects to a common CAN bus network.

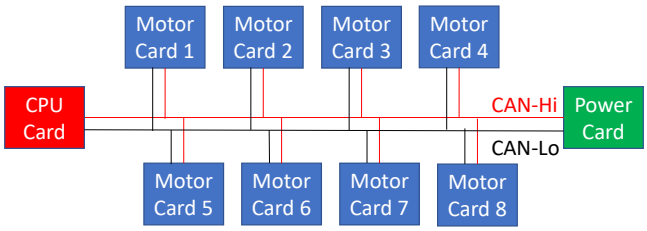

Figure 6: Europa Lander Motor Controller CAN Bus Implementation

In the CAN bus protocol, there is no concept of a master and a slave. Nor is there a concept of a node address. Instead, messages are given unique identifiers based upon message type. The standard assigns priority to these messages based upon their type.

For Europa Lander Motor Controller, as in other real time systems, we have a requirement to have consistent latency for our control, and telemetry communication. We typically achieve this by have remote nodes on a network send data only when requested by the processor. This enables the processor to control the bus communication and predictable latencies throughout the system.

Table 4: Motor Control Data Frame Definition

CAN2.0A Frame	ID (11 bits)		Data Frame (0-8 bytes)							
	Command	Node ID								
Number of Bits	5	6	8	8	8	8	8	8	8	8
Write Command	0x00	Slave-ID	REG - ID	0x0F	16 bits data		Null	Null	Null	Null
Read Command	0x01	Slave-ID	REG - ID	Null	Null	Null	Null	Null	Null	Null
Response command of Power	0x02	Host-ID	REG - ID	Slave-ID	16 bits data		Null	Null	Null	Null
Response command of Motor Card	0x03	Host-ID	REG - ID	Slave-ID	16 bits data		Null	Null	Null	Null
Write 512Hz	0x04	Slave-ID	0x0F	0x0F	16 bits data		Null		Null	
Poke 512Hz	0x05	Slave-ID	Null		Null		Null		Null	
Response poke 512Hz	0x06	Host-ID	Group 10	Slave-ID	16 bits data		16 bits data		16 bits data	
Write 64Hz - to Motor Card	0x07	Slave-ID	0x0F	0x0F	Current limit		Null		Null	
Write 64Hz - to Power Card	0x08	Slave-ID	0x0F	0x0F	16 bits data		16 bits data		Null	
Poke 64Hz	0x09	Slave-ID	Null		Null		Null		Null	
Response poke 64Hz data - POWER CARD	0x0A	Host-ID	Group A	Slave-ID	16 bits data		16 bits data		Null	
	0x0A	Host-ID	Group B	Slave-ID	16 bits data		16 bits data		16 bits data	
	0x0A	Host-ID	Group C	Slave-ID	16 bits data		16 bits data		16 bits data	
	0x0A	Host-ID	Group D	Slave-ID	16 bits data		16 bits data		16 bits data	
	0x0A	Host-ID	Group E	Slave-ID	16 bits data		16 bits data		16 bits data	
Response poke 64Hz data - Motor Card	0x0B	Host-ID	Group 1	Slave-ID	16 bits data		16 bits data		16 bits data	
	0x0B	Host-ID	Group 2	Slave-ID	16 bits data		16 bits data		16 bits data	
	0x0B	Host-ID	Group 3	Slave-ID	16 bits data		16 bits data		16 bits data	
	0x0B	Host-ID	Group 0	Slave-ID	16 bits data		16 bits data		16 bits data	

For our initial mapping to CAN bus we control latencies by requiring the slave nodes to only send data when requested by the host through a POKE command. As illustrated in Table 4, the POKE command is a simple packet with no data in the data frame. A different POKE command is created for each data type requested. The assignment of priorities is done based upon the 11-bit ID field. The first 5 bits define the type of data requested with the next 6 bits defining the node the data is requested from.

The requested data is returned through standard CAN messages with bits allocated in the data frame for the group number, slave id and returned data.

The Standard CAN data frame is limited to 8 bytes of data. The telemetry from our motor controllers is many bytes more than this limitation. We get around this by breaking the telemetry into 8-byte groups. When requested by the host through the POKE command the requested data is identified by the NODE ID of the slave along with the GROUP the data is requested from. Table 5 shows the way how we separate all the telemetry into 10 groups data for motor card, and 5 groups data for power card.

Table 5: Group Assignments

Group (Hex)	Freq Signal (Hz)	Decription	From
0x00	64	6 bytes data of all motors	Motor Card 1
0x01	64	1st - 6 bytes data of Motor 1	
0x02	64	2nd - 6 bytes data of Motor 1	
0x03	64	3rd - 6 bytes data of Motor 1	
0x04	64	1st - 6 bytes data of Motor 2	
0x05	64	2nd - 6 bytes data of Motor 2	
0x06	64	3rd - 6 bytes data of Motor 2	
0x07	64	1st - 6 bytes data of Motor 3	
0x08	64	2nd - 6 bytes data of Motor 3	
0x09	64	3rd - 6 bytes data of Motor 3	
0x0A	64	1st - 6 bytes data of Power Card	Power Card
0x0B	64	2nd - 6 bytes data of Power Card	
0x0C	64	3rd - 6 bytes data of Power Card	
0x0D	64	4th - 6 bytes data of Power Card	
0x0E	64	5th - 6 bytes data of Power Card	
0x10	512	512 Hz Signal of Motor 1	Motor Card 1
0x11	512	513 Hz Signal of Motor 2	
0x12	512	514 Hz Signal of Motor 3	

II. Analysis transmission time

Once we have completed the mapping of our required transmissions to the CAN bus protocol, the next step is to perform an analysis to show that our data requirements can be met. Based on the Motor Control Data Frame Definition in table 4, we can see that the full CAN message have 108 bits. Theoretically, the speed of CAN bus is 1Mb/s. Therefore, for sending a CAN message with 108 bits, the bus takes 108 µs without any error, any delay and any latency. Hence, we have the detailed time for each CAN Message in table 6.

Table 6: Time Analysis – Closing loop with an MCC and a PCC at the rate of 64 Hz and 512 Hz

CAN 2.0A	Group	Data (bits)	Overhead (bits)	Full Msg (bits)	Hz	Total (bits/s)	Sum (bits/s)
	Poke	**0**	**44**	44	**64**	2816	
Response Power Card	A	48	44	92	**64**	5888	36,352
	B	64	44	108	**64**	6912	
	C	64	44	108	**64**	6912	
	D	64	44	108	**64**	6912	
	E	64	44	108	**64**	6912	
	Poke	**0**	**44**	44	**64**	2816	
3 motors	0	64	44	108	**64**	6912	
Response Motor 1	1	64	44	108	**64**	6912	
	2	64	44	108	**64**	6912	
	3	64	44	108	**64**	6912	
Response Motor 2	4	64	44	108	**64**	6912	71,936
	5	64	44	108	**64**	6912	
	6	64	44	108	**64**	6912	
Response Motor 3	7	64	44	108	**64**	6912	
	8	64	44	108	**64**	6912	
	9	64	44	108	**64**	6912	
	Poke	**0**	**44**	44	**512**	22528	
512Hz Signals Motor 1	10	64	44	108	**512**	55296	188,416
512Hz Signals Motor 2	11	64	44	108	**512**	55296	
512Hz Signals Motor 3	12	64	44	108	**512**	55296	
Closing loop of an MCC and a PCC (bits/s)							**296,704**
Speed of CAN BUS (bits/s)							**1,000,000**
Percent Total Bandwidth Available							**70.33%**

As illustrated in table 6, for the system with an MCP, a PCC and an MCC, total bus traffic is nearly 300kbps. Therefore, we could easily see that with the speed 1Mbps, the Standard CAN was too slow for our application with 8 MCCs running at the same time. However, in the future, we will be closing the loop within the FPGA for reducing the bandwidth requirements for the system, so our system only requires the collection of all telemetry at a rate of 64Hz. Therefore, the total bus traffic for system with 1 MCP + 1 PCC + 8 MCC is:

*Total = Total a PCC response + Total an MCC response * 8 + LVDT information 64Hz * 8*

*= 36352 + 71,936*8 + 3*108*64*8*

= 777,728 bits/s

Consequently, total free bandwidth is 22.23% (about 23 kbps). Therefore, the standard CAN will work in this way. For more detail, we can analysis with the time line of the bus traffic. We are using the major time frame is 1 cycle of 64 Hz signal (15625 µs) and minor time frame is 1 cycle of 512 Hz signal (1953µs). In other words, we know that 1 cycle of 64 Hz signal equal 8 cycles of 512 Hz signal. In each minor time frame, it has all full telemetry for 1 motor card (3 motors). Therefore, in major time frame, it has total 8 minor time frames for 8 motor cards.

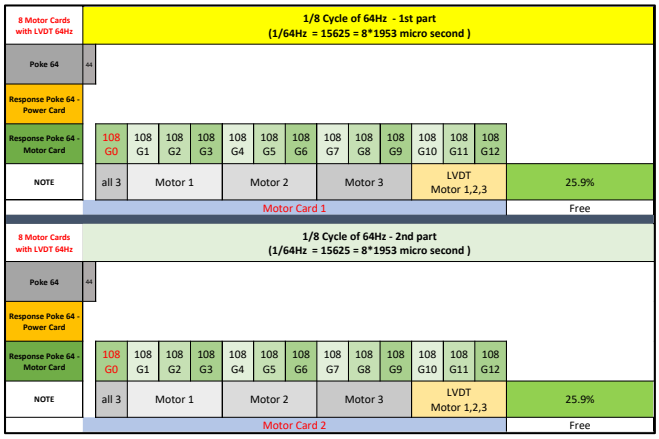

Figure 7: First and Second of Minor Time Frame with 8 MCCs at the rate of 64 Hz

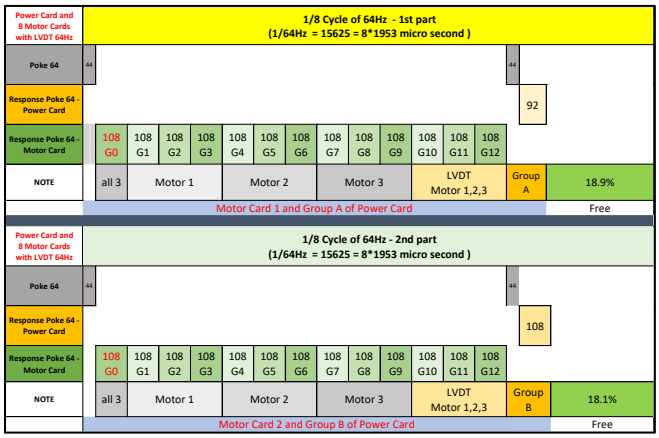

Figure 8: First and Second of Minor Time Frame with 8 MCCs and PCC at the rate of 64 Hz

As shown on the figure 7 and 8, we are using time frames for sending message, so for the synchronization between nodes and our initial mapping to CAN bus we control latencies by requiring the slave nodes to only send data when requested by the host through a POKE command. After the Power Card or Motor Card receive the Poke command from CPU card, they send back all the telemetry to the CPU card. Based on the 11-bits ID and 1-bytes group ID and 1-byte for nodes ID in standard CAN message, the CPU can know exactly where the CAN message come from and what telemetry data are contained in the message. Consequently, as shown in figure 7, all 8 parts of the 64 Hz cycle have 25.9% free bus traffic, so the total free bandwidth is **25.9%** for 8 MCCs. Also, as shown in the figure 8, the first part of 64 Hz cycle has 18.9% free bus traffic; the second, third, fourth, or fifth part has 18.1% free bus traffic; the sixth, seventh, or eighth part has 25.9% free bus traffic; so, the total free bandwidth is about **21.13%** for 8 MCCs and a PCC.

6. Arduino Simulation

For simulating and demonstrating the real system response, the Arduino UNO microcontroller was chosen as the desired hardware platform. The Arduino IDE using the Arduino complier served as our software development environment. Both Arduino UNO and Arduino IDE are open source and freely with no proprietary intellectual property costs. As shown in figure 9, the Arduino UNO was interfaced with the Arduino Controller Area Network Bus (CAN bus) shield which is also open hardware, and was interfaced with TFT Touch Screen shield for controlling and monitoring all the testing data. [8]

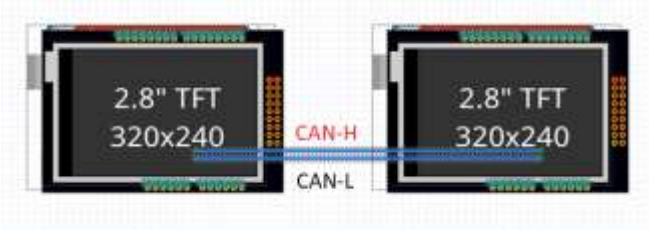

Figure 9: Wiring Diagram of Arduino is connected with CAN bus Shield and TFT Touch Screen Shield.

As shown in the figure 10, one Arduino simulated the computer card. The other one simulated the power card's microcontroller. The computer card has two modes of operation: writing data to power card node and reading data from power card node.

- Writing command: The computer card generates data, displays it on the screen and sends it to the power card by CAN 2.0A message. Then, the power card displays the data on the screen
- Reading Command: The computer card sends the read request to the power card by CAN 2.0A message. The power card generates the data and displays on screen. Then, it sends back the data to the computer card by CAN 2.0A messages. The computer card receives the data and displays on the screen.
- All commands are controlled by touch screen. Also, all data are easily monitored on the screen by both the rolling bar and number.
- The data are voltage, current and temperature.

The Arduino UNO performed CAN bus communication using the CAN bus shield. The CAN bus shield uses the MCP2515 CAN bus controller coupled with a Serial Peripheral Interface (SPI) and a MCP2551 CAN bus transceiver. A total of 2 Arduino UNO was coupled with 2 CAN bus shield and 2 TFT Touch Screen to form the CAN bus network topology comprising of Master/Host and Peripheral/Slave CAN bus nodes. The master node is used to simulate the microprocessor GR712, and the slave node is used to simulate the Power Conversion Card and the Motor controller Cards. Based on the design of the CAN 2.0 data frame, the system was working well. The Slave node is used to simulate to generate the signals for the voltage, current and temperature as the power card. The master node with the Touch Screen is used to simulate to monitor and control the slave node as the microprocessor.

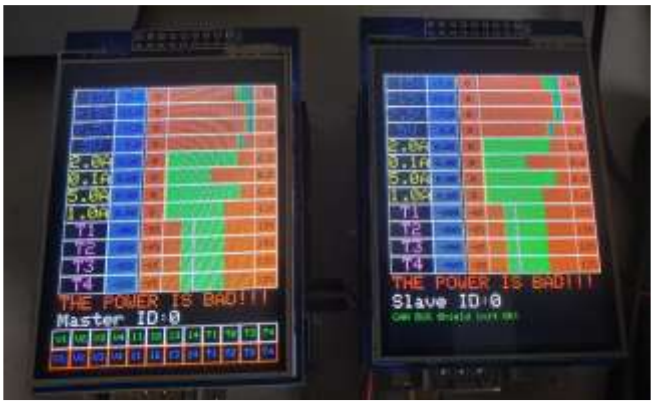

Figure 10: The interface of 2 TFT Touch Screen of Computer card (master) and Power Card (slave)

7. Latency

As shown on table 8, it is important to consider the latency of our system. Latency is dominated by the microprocessor, CAN controller, CAN transceiver, and the bit stuffing. These factors are beyond the bandwidth of the physical layer and contribute to effective throughput of the system.

- The latency of the microprocessor (propagation latency) and refers to the time to process the last message and start another one. This time depends on the speed of the microprocessor and the implementation code. This latency can be reduced by using a high-speed processor, a Field Programmable Gate Arrays (FPGA), and a Direct Memory Access (DMA) engine.
- The latency of the CAN controller and CAN transceiver stems from hardware delays. Therefore, we can't reduce it on Arduino. However, the GR712, which we are considering using on flight, has a better CAN controller and CAN transceiver, so its latency is significantly lesser. The details are shown in table 7.

Table 7: Latency of the CAN controller and transceiver

Platform	Arduino		GR712	
Part	CAN Controller	CAN Transceiver	CAN Controller	CAN Transceiver
Chip name	MCP2515	MCP2551	SJA1000	UT64CAN333x
Total delay for input and output	200-2100ns	1240ns + 5us	40ns	745ns + 1.5us

Table 7: Latency of the CAN controller and transceiver

- The latency of the bit stuffing: CAN protocol requires a bit of the opposite polarity to be inserted into the message by the CAN controller whenever 5 continuous bits of the same polarity are transmitted to ensure enough transitions to maintain synchronization [9]. The bit stuffing increases the maximum transmission time of CAN messages by nearly 20% in the worst case, as shown in table 8. The time taken for bit stuffing is dependent on the data.

Table 8: Measure latency of the Arduino

Unit in Microsecond (µs)	Measured with Arduino		Theory - Ideal - no process		Latency	
Speed of CAN BUS (kbps)	1000kbps	500kbps	1000kbps	500kbps	1000kbps	500kbps
Sending Poke msg - No data	156	208	44	88	112	120
Sending CAN msg with "AA" 8 bytes data - prevent stuff bit	228	336	108	216	120	120
Sending CAN msg with " FF" 8 bytes data - expected stuff bit	256	364	120	240	136	124
Closing loop 1 - 1 poke and 1 response "AA"	476	620	152	304	324	316
Closing loop 1 - 1 poke and 1 response "FF"	492	664	164	328	328	336
Closing loop 3 - 1 poke 3 response	964	1324	402	804	562	520

8. Future work

In the future we intend to design and develop the CAN bus using CAN FD because of its advances from standard CAN. These advances include a bigger data frame, and higher speed.

- Big data frame: With the increased data frame size of 64 bytes per data frame, we can reduce the overhead by transferring all of our telemetry in one message. With 64 data frame, 1 CAN FD message can include all the data for 1 motor card (telemetry from 3 motors).

- Higher Speed: The speed of CAN FD can go up to 12Mb/s. This is 10 times faster than Standard CAN. With the higher speed, CAN FD can reduce transmission time. Therefore, the bus will have more free time and allow more active motor controllers to communicate over a single bus.

9. Conclusion

Our CAN bus simulation using Arduinos and CAN Bus Shields is similar to the actual CAN bus network control process on the proposed Europa Lander. This method provides us a means of simulating the CAN bus enough to assess its applicability to Europa Lander. This system allowed us to easily change simulation parameters, and measure system throughput and latency. We found the Arduino simulation to have high latency when compared to our flight GR712 processor. This is due to slower microprocessor and non-flight like CAN bus shield..

In this paper we looked at Standard CAN and CAN FD. We showed that Standard CAN bus (CAN2.0A) would not work as initially as required. The bus was not able to keep up with 512hz messages required to close the motor control loops. However, in the future, we will be closing these loops within the FPGA which reducing the bandwidth requirements for the system. In that case, Standard CAN will work for us.

10. Acknowledgement

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

11. References

1. Europa Lander Mission Concept Overview. (PDF) Grace Tan-Wang, Steve Sell. Jet Propulsion Laboratory, NASA. AbSciCon2019, Bellevue, WA - June 26, 2019.
2. G. Bolotin, D. Hunter, D. Sheldon, Y. He and D. Foor, "Compact low power avionics for the Europa Lander concept and other missions to ocean worlds," *2018 IEEE Aerospace Conference*, Big Sky, MT, 2018, pp. 1-11, doi: 10.1109/AERO.2018.8396418.
3. Y. Lv, W. Tian and S. Yin, "Design and Confirmation of a CAN bus Controller Model with Simple User Interface," *2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*, Qinhuangdao, 2015, pp. 640-644, doi: 10.1109/IMCCC.2015.140.
4. Anthony, P.L, H.B Crawley, P.-A Fischer, R.L Mckay, and W.T Meyer. "CANbus and Microcontroller Use in the BaBar Detector at SLAC." 1999 IEEE Conference on Real-Time Computer Applications in Nuclear Particle and Plasma Physics. 11th IEEE NPSS Real Time Conference. Conference Record (Cat. No.99EX295) 47 (1999): 260-63. Web.
5. Corrigan, S., 2016. Introduction to The Controller Area Network (CAN). [pdf] Texas Instrument. Available at: <https://www.ti.com/lit/an/sloa101b/

sloa101b.pdf?ts=1597936955031> [Accessed 20 August 2020].

6. J. M. Giron-Sierra, C. Insaurralde, M. Seminario, J. F. Jiménez and P. Klose, "CANbus-based distributed fuel system with smart components," in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, no. 3, pp. 897-912, July 2008, doi: 10.1109/TAES.2008.4655351.
7. Y. Xie, P. Huang, W. Liang and Y. He, "Comparison between CAN and CAN FD: A Quantified Approach," *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, Guangzhou, 2017, pp. 1399-1403, doi: 10.1109/ISPA/IUCC.2017.00212.
8. N. Yee, P. Chand and S. Foehst, "Student Designed CANBus Simulator Used as Teaching Aid in Autotronics Course," *2017 4th Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, Nadi, 2017, pp. 82-87, doi: 10.1109/APWConCSE.2017.00023.
9. J. M. Giron-Sierra, C. Insaurralde, M. Seminario and J. F. Jimenez, "Distributed control system for fuel management using CANbus," *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*, Salt Lake City, UT, USA, 2004, pp. 8.D.2-8.1, doi: 10.1109/DASC.2004.1390770.
10. Mary, G. I., Alex, Z. C., & Jenkins, L. (2013). Response Time Analysis of Messages in Controller Area Network: A Review. Journal of Computer Networks and Communications, 2013, 1-11. doi:10.1155/2013/148015

Biography

Hieu Tran is currently a full-time intern in Summer 2020 and part-time in Fall 2020 at NASA Jet Propulsion Laboratory. He will be graduating from California Polytechnic University, Pomona in December 2020 with a B.S in Electrical Engineering. Upon graduation with GPA 3.96, his dream is to be a member of the JPL motor control team. Being named on Dean's Honor List and President's Honor list in 2018, 2019, 2020 and being selected for Chevron Scholarship, Edison Scholarship, Robert R Sprague Scholarship, and so on while studying at Cal Poly Pomona.

Gary Bolotin received a B.S. in Engineering from Illinois Institute of Technology in 1984 and a M.S. in Engineering from University of Illinois at Urbana Champaign in 1985. He has been with JPL for more than 34 years. He is currently the lead for the Europa Lander Motor Controller. He has also managed engineering teams as both teams leads and line manager at the section and group level.

Ben Cheng holds a B.S. in Electrical Engineering from California Polytechnic University, Pomona. He is currently working at NASA Jet Propulsion Laboratory as a member of the Motor Control Card Development Team since 2018. Ben is supporting the test and development of the Europa Lander motor control modules and board designs. Prior to joining JPL, he was the Vice President and a Board of Director for MAG Laboratory (a makerspace) and the head of multiple Robotics-centered student organizations and projects.

Allen Sirota is the technical group supervisor of the Robotic Actuation and Sensing Group at NASA Jet Propulsion Laboratory, California Institute of Technology, where he has been since 1983. He received his B.S. degree in Electronics Engineering from the University of California, Los Angeles, in 1976. Allen Sirota received the 1997 NASA Exceptional Engineering Achievement Medal for his contributions to the Mars Pathfinder Sojourner Rover mission.

Malcolm Lias holds a B. S. in Electrical Engineering from Rochester Institute of Technology. He is currently developing Motor Control Card for Europa Lander at Jet Propulsion Laboratory (JPL). Testing and documenting the Distributed Motor Control Multi-Chip Module for use on the Europa Lander. Prior to joining JPL, Malcom worked for Word Inc where he was responsible for design, product support, and testing of control electronics for missiles, smart bombs, and aircraft.