Question 3 - Controller Area Network

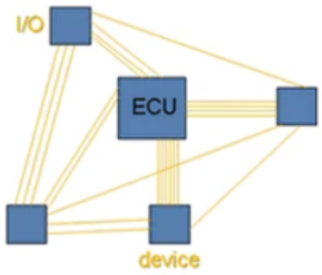
Nitin G EE22B041

March 2023

1 Answers

1.1 Part A

You now have some idea about how CAN works. Now explain the disadvantages and advantages of CAN instead of the communication network shown below.



The most obvious disadvantage with the network show bellow is the amount of wires required to create a point to point network. Can networks use a centralised bus and therefore eliminates this problem. Some other advantages of can are:

- Single point of connection - we can connect a computer from a single point and read all the data.
- Easier to add new nodes
- Robust nature - works in all kinds of environments and is robust enough to correct any errors due to environment.
- It is possible to have many inbuilt error checking mechanisms, like crc, ack error, form error, bit stuffing, etc.

Despite these, there are a few disadvantages of CAN network over the shown network as well. These are:

- Limited distance: CAN is designed for use over short distances, typically no more than a few hundred meters.
- Limited bandwidth: CAN has a limited bandwidth of up to 1 Mbps, which may not be sufficient for some high-speed applications.
- Limited number of nodes: CAN networks are typically limited to a few dozen nodes, which may not be sufficient for large-scale applications.
- It is a complex protocol and is difficult to implement in software and hardware.
- High cost of maintenance.

1.2 Part B

CAN protocol uses 2 data lines to send information across namely CAN-H and CAN-L. What is the significance of using 2 lines? Explain the advantages and disadvantages of this method.

The use of two data lines (CAN-H and CAN-L) in the CAN protocol is known as differential signaling.

Differential signaling refers to a technique where data is transmitted across two wires with opposite polarity, where one wire is positive (CAN-H) and the other is negative (CAN-L). The voltage difference between the two wires determines the value of the transmitted data. If CAN-L ¡ CAN-H then the signal is a dominant (0), if not then it is a recessive (1).

The significance of using two lines in the CAN protocol is that it provides several benefits, including:

- Noise immunity: Differential signaling helps to minimize noise interference, as any external noise signal will affect both wires equally. The receiver only detects the difference between the two signals and ignores any common-mode noise.
- Increased range: The use of differential signaling allows for longer cable runs without signal degradation. This makes CAN protocol suitable for use in harsh environments, such as automotive or industrial settings.
- High-speed data transmission: By transmitting data differentially, CAN protocol can achieve high-speed data transfer rates of up to 1 Mbps.

Using this method increases the robustness and reliability of transmission. Though there are some disadvantages of this type of transmission, which is:

- Increased complexity: the differential signalling must be done within a CAN network and this increases the amount of complexity and the implementation becomes tougher.
- Limited number of nodes as explained above.
- Synchronisation issues: both the signals must be synchronised or else we will read non sense values, therefore extra effort must be put in order to keep the signals in sync.

1.3 Part C

Let us first decode the data in all four cases:
The bits transmitted are:
i) 001100101001110
ii) 001100101001111
iii) 001010101011011
iv) 001101011001011

The first bit is the start of frame bit, which must be dominant. And it is dominant (0) in all cases.

The next 11 bits are the first part of the Identifier.

The next bit is the Substitute remote request (SRR). This must be a 1 in case of extended CAN frames. But in case (iii) and (iv) it is 0 indicating that it is a standard frame and not an extended one. The standard frame takes arbitration over the extended frame.

The next bit is the Identifier extension bit (IDE). It must be 1 for all the messages with a 29 bit ID. In all cases here it is 1.

The next bit is the first bit of the part B of the identifier.

Now, on taking the relevant bits and writing the id's for each, we get:
i) 011001010010
ii) 011001010011
iii) 010101010111
iv) 011010110011

whenever there is a difference in the ID bit, the node(s) with a 0 will take arbitration, i.e will gain preference and the recessive node will stop transmitting and will try again later.

Therefore the preference order will be (iii) then (i) then (ii) and then (iv).

1.4 Part D

Part D

We first process each frame to binary

In [ ]:
```
frame1 = bin(0x127D555503876B92FF)
frame2 = bin(0x127EAAAA046798B912FF)
frame3 = bin(0x27EAAAA059867F922FF)

frames = [frame1, frame2, frame3]
n_s = [(len(frame)//4 + 1)*4 for frame in frames ]
processed_frames = [frames[i][2:].rjust(n_s[i], '0') for i in range(len(f
```

We define the decode() function to iterate through the frame and pick out the id and data and ensure all other forms are as expected

In [ ]:
```
def decode(frame):
    pointer = 0
    id = ''
    if frame[pointer] != '0':
        print('Start of frame bit error')
    pointer +=1

    for i in range(11):
        id += frame[pointer]
        pointer +=1

    if frame[pointer] != '1':
        print('SRR bit error')
    pointer +=1

    if frame[pointer] != '1':
        print('IDE bit error')
    pointer +=1

    for i in range(18):
        id += frame[pointer]
        pointer +=1


    if frame[pointer] != '0':
        print('It is not a data frame')
    pointer +=1

    pointer += 2 # reserved bits we dont care about

    num_bytes = int(frame[pointer: pointer+4], 2)
    pointer +=4

    data = frame[pointer: pointer+(num_bytes*8)]
    pointer += (num_bytes*8)

    crc = frame[pointer: pointer+15]
    pointer +=15

    if frame[pointer] != '1':
        print('crc delimiter error')
```

```
    pointer +=1

    pointer +=1 #ack slot, we dont care about this for the scope of the q

    if frame[pointer] != '1':
        print('ack delimiter error')
    pointer +=1

    if frame[pointer: pointer +7] != '1111111':
        print('end of frame error')
    pointer +=7

    return id, data
```

We print the id and data for each frame.

In [ ]:
```
for i in range(3):
    id, data = decode(processed_frames[i])
    print(f'Frame {i+1}:')
    print(f'ID: {id}')
    print(f'Data: {data}')
```

```
Frame 1:
ID: 00100100111010101010101010101
Data: 11000011
Frame 2:
ID: 00100100111010101010101010101010
Data: 0011001111001100
Frame 3:
ID: 000001001111010101010101010101010
Data: 1100110000110011
```

Looking at this we can say the priority order of frame 3 > frame 1 > frame 2.

1.5 Part E

We will first list out all the modules in order of priority and their messsage id's. First being highest priority

1. Emergency Stop - '00000000000000000000000000000000'
2. Brakes - '00000000000000000000000000000001'
3. Steering - '00000000000000000000000000000010'
4. Throttle - '00000000000000000000000000000011'
5. Brake Lights - '00000000000000000000000000010000'
6. Headlights - '00000000000000000000000000010001'
7. Horn - '00000000000000000000000000010100'
8. door lock/unlock - '00000000000000000000000001000000'
9. longitudinal velocity feedback - '00000000000000000000000010000000'
10. steering angle feedback - '00000000000000000000000010000001'
11. battery monitoring details - '00000000000000000000000010000010'

Now I shall justify this order. Emergency stop must always have highest priority. Then comes brakes then steering as these are the most important form of control of the vehicle. These are the basic functional features thus they have high priority, then comes the medium priority. These are safety features like brake lights, headlights and horn. Then we have door lock/unlock. Then we have the feedbacks. These complete the control loop of brakes and steering without which we would not be able to control both, therefore they are necessary too, but we have it lower in our priority because they transmit data continuously and if they have a higher priority they would keep stalling the other systems. Finally we have battery monitoring as it is a process that happens at all times, and is transmitting data at all times, we have it at least priority.

1.6 Part F

Explain the algorithm used by each node to determine whether a message has to be ignored or not (Hint: Masking Filtering)

The algorithm for ignoring a message is a basic mask and filter one. There are 2 layers to this algorithm. First, there is a mask, which is the same size as that of the frame. Now whichever bit of the mask is 1, those bits alone will be considered by the filter. Secondly the filter takes these bits that are to be considered and checks if that bit is same as the one in its memory. That is it will check if those bits are the correct ones as per specified. If it is, then the frame is taken else it is ignored.

In short, the mask selects which bits to compare and the filter tells if the bits are same as the ones we require. If not then we ignore the frame.

1.7 Part G

Find the number of IDs the node accepts for the given mask and filter
a) Mask = 0x15D75D7D, Filter = 0x13579BDF
b) Mask = 0x12D5AFAA, Filter = 0xFDB97531

We note here that the exact contents of the filter doesn't matter in this context. This is because all the bits that have 1 as thee bit in the mask, will have to match to the filter, whatever the value is, doesn't matter. So to find the number of ID's that the node can accept, we do simple permutation combination logic to see that it will be two raised to the number of zeroes in the mask binary number. This is because whenever a mask has 1, then the id bit in that place can only take one value, if the mask is zero then it can take 2 values, either 0 or 1.

$$answer = 2^{numberofzeroes} \tag{1}$$

We write the masks in binary.
a) Mask = 10101110101110101110101111101
b) Mask = 10010110101011010111110101010

There are 9 zeroes in case a. Therefore the number of id's that the node accepts is $2^9 = 512$.

There are 9 zeroes in case b. Therefore the number of id's that the node accepts is $2^{12} = 4096$.

1.8 Part H

Explain Acknowledgement error and Cyclic Redundancy Error briefly.

An Acknowledgment (ACK) error occurs when a sender does not receive an acknowledgment message from the receiver after sending data. In data communication, acknowledgement messages are used to confirm that data has been received successfully by the recipient. The ack bit is set to a recessive 1, when a reciever succesfully receives the message, it transmits a dominant 0 in the ack bit. If the transmitter does not detect a 0 in this bit, then an ack error occurs. The sender might try to retransmit the data, leading to delays or even communication breakdown.

Cyclic Redundancy Check (CRC) is a type of error-detection code used in digital networks and storage devices. The CRC algorithm calculates a unique code based on the data being transmitted or stored, which is added to the data as a checksum. A generating polynomial that both the sender and receiver is aware of is used to encode the data, k extra bits (zeros) are added to the end of the data. here k = degree of the generating polynomial. This new data is binary divided by the generating polynomial's binary representation. This will leave a reminder, this is called the checksum. Now this is added to the original data. Now the receiver can binary divide the entire package of data and if they get reminder zero then the file has not been corrupted. This can be used to detect

errors in single bit, double bit and even burst errors with number of errors less than the degree of the polynomial. It is a strong error detection method though not a very strong error correction method.

1.9 Part I

When a Transmitter sends a CAN frame 0x493FFC33021E9FEBFF what happens to the frame when an acknowledgement error happens and what happens to the frame when a receiver acknowledges the message?

When the CAN Frame is sent by the transmitter, the ack slot is defaulted to 1.

In this case, the frame is

10010010011111111111100001100110000001000011110100111111110101111111111.

We can see that the ack slot is transmitted at one by the transmitting node.

Whenever a node successfully receives the frame, the node transmits a dominant 0 in the ack slot. The transmitter listens for this 0. If it doesn't detect a Zero in the ACK slot, then an acknowledgement error is raised. According to the system it can raise an error frame and let all the nodes know that an error has occured. It could also re transmit the message again after some time.

1.10 Part J

A transmitter sends a CAN frame of 0x527EAAAA0618B3200003FF, and the receiver reads a CAN frame of 0x527EAAAA063166400002FF. Does this result in Cyclic Redundancy Error or not? justify your answer.The probability of a bit flip is 1% in the data field, what is the probability that this particular event happens?

Part J

We first process each frame to binary and decode it through the decoder from part D with slight modifications

We define the decode() function to iterate through the frame and pick out the id, data and crc checksums, and ensure all other forms are as expected

In [ ]:

```
def decode(frame):
    frame = bin(frame)[2:]
    n_s = (len(frame)//4 + 1)*4
    frame = frame.rjust(n_s, '0')
    pointer = 0
    id = ''
    if frame[pointer] != '0':
        print('Start of frame bit error')
    pointer +=1

    for i in range(11):
        id += frame[pointer]
        pointer +=1

    if frame[pointer] != '1':
        print('SRR bit error')
    pointer +=1

    if frame[pointer] != '1':
        print('IDE bit error')
    pointer +=1

    for i in range(18):
        id += frame[pointer]
        pointer +=1


    if frame[pointer] != '0':
        print('It is not a data frame')
    pointer +=1

    pointer += 2 # reserved bits we dont care about

    num_bytes = int(frame[pointer: pointer+4], 2)
    pointer +=4

    data = frame[pointer: pointer+(num_bytes*8)]
    pointer += (num_bytes*8)

    crc = frame[pointer: pointer+15]
    pointer +=15

    if frame[pointer] != '1':
        print('crc delimiter error')
    pointer +=1

    ack = frame[pointer]
    pointer +=1
```

```
    if frame[pointer] != '1':
        print('ack delimiter error')
    pointer +=1

    if frame[pointer: pointer +7] != '1111111':
        print('end of frame error')
    pointer +=7

    return id, data, crc, ack
```

In [ ]:
```
id1, data1, crc1, ack1 = decode(0x527EAAAA0618B3200003FF)
id2, data2, crc2, ack2 = decode(0x527EAAAA063166400002FF)
```

In [ ]:
```
print(crc1, crc2, sep = '\n')
```

```
000000000000000
000000000000000
```

In [ ]:
```
print(data1, data2, sep = '\n')
```

```
000011000101100110010000
000110001011001100100000
```

We have gotten all the data from the frames, now let us see what happens when we check if our data is correct using the crc. The data is obviously different, so we should expect an error to occur.

In [ ]:
```
generating_poly_binary = '1100010110011001' # this is the binary represen
```

From the underlying principal of CRC, we expect to get a reminder of 0 when we divide the data with crc bits added by the generating polynomial

In [ ]:
```
print(int(data1 + crc1, 2) % int(generating_poly_binary, 2))
```

Out[ ]:
```
0
```

The original data gives zero as expected,

In [ ]:
```
print(int(data2 + crc2, 2) % int(generating_poly_binary, 2))
```

```
0
```

We expect to get a non zero answer here, because the data has been changed and errors have occured, but in this case our CRC fails to detect this and gives a reminder of 0. We want to find the likelihood of this happening.

In [ ]:
```
num_bits_changed = 0
a = data1 + crc1
b = data2 + crc1
for i in range(len(a)):
    if a[i] != b[1]:
        num_bits_changed +=1

print(num_bits_changed)
```

This case happens only when the data has been modified in a special way that it happens to fool the CRC. In this case it requires 8 bit swaps. So the chance that this happens is $10^{-2^8} = 10^{-16}$. That is there is a one in 10^{16} chance that this will happen.

2 Sources

1. https://en.wikipedia.org/wiki/CAN_bus
2. https://forum.arduino.cc/t/filtering-and-masking-in-can-bus/586068/3
3. https://911electronic.com/what-is-can-bus-protocol/
4. https://en.wikipedia.org/wiki/Cyclic_redundancy_check

3 NOTE

To access all the source files, screenshots and ipynb files please visit this github repo:

https://github.com/MachanHoid/AbhiyaanElecApp.git