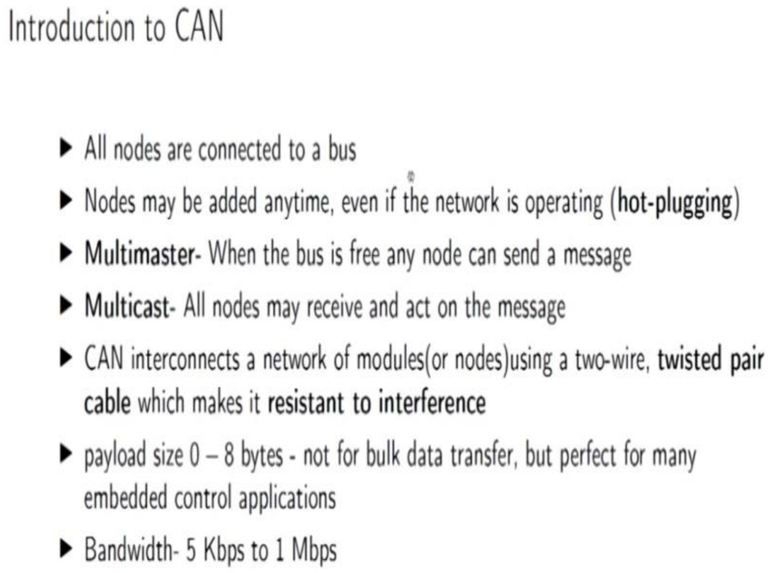
Foundations of Cyber Physical Systems
Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Module No # 02
Lecture No # 08
Real Time Sensing and Communication for CPS (Contd.,)

Welcome back to this course on Cyber Physical System. So, in the previous lecture we have been taking about CAN bus and how such CAN bus work, right.

(Refer Slide Time: 00:40)

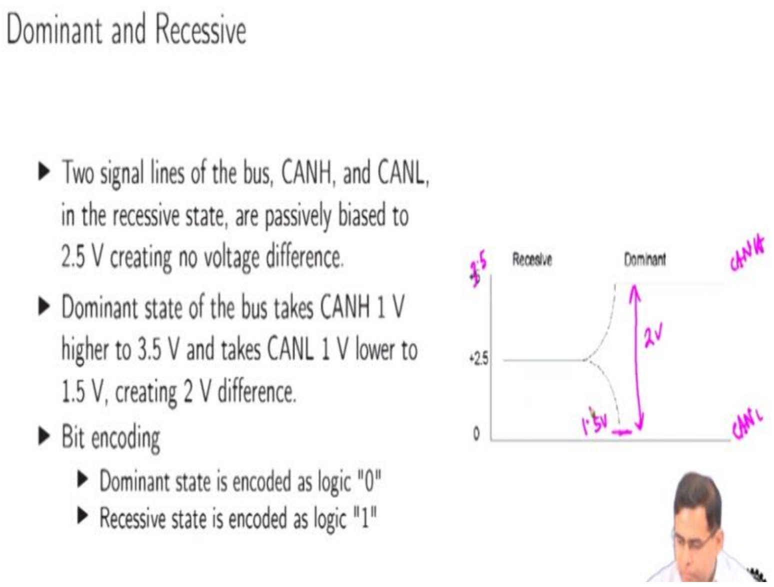


Introduction to CAN

- All nodes are connected to a bus
- Nodes may be added anytime, even if the network is operating (**hot-plugging**)
- **Multimaster-** When the bus is free any node can send a message
- **Multicast-** All nodes may receive and act on the message
- CAN interconnects a network of modules(or nodes)using a two-wire, **twisted pair cable** which makes it **resistant to interference**
- payload size 0 – 8 bytes - not for bulk data transfer, but perfect for many embedded control applications
- Bandwidth- 5 Kbps to 1 Mbps

So, let us switch over to the slides (refer time: 00:40) from where we left. So, we were talking about typically I mean the properties of the CAN bus, right. And we said that a CAN bus at a physical level, although it is not enforced, but the way the CAN bus is implemented is it’s a twisted pair cable, where there are 2 signal buses, 2 signal lines, the CAN high and the CAN low line, okay.

(Refer Slide Time: 01:09)



Dominant and Recessive

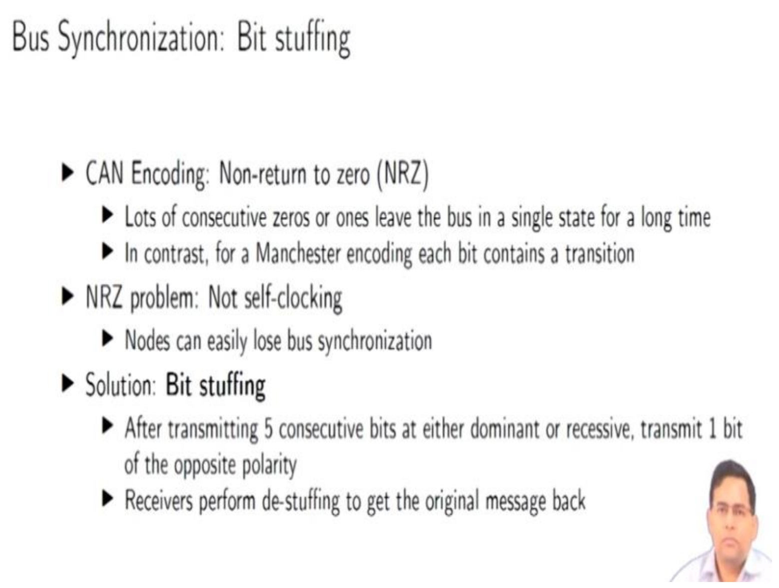
- Two signal lines of the bus, CANH, and CANL, in the recessive state, are passively biased to 2.5 V creating no voltage difference.
- Dominant state of the bus takes CANH 1 V higher to 3.5 V and takes CANL 1 V lower to 1.5 V, creating 2 V difference.
- Bit encoding
  - Dominant state is encoded as logic "0"
  - Recessive state is encoded as logic "1"

So, what happens is, they maintain, I mean, the overall this bus, this twisted pair has 2 states. Its either in dominant state or it is in the recessive state. So, when it is in recessive state, that means, this CAN high and CAN low, they are both biased positively with a voltage of 2.5. Hence, there is no voltage difference between them. So that is the recessive state. And when we say that the bus is in a dominant state, that means, CAN high will assume a voltage value which is 1 volt higher, so that is 3.5 volt, and CAN low will assume a voltage that is 1 volt lower, that is 1.5 volt, and this will create a 2 volt difference.

So, this is something like, 3.5 here, and let us say that when this guy who is in the CAN low. So, this is CAN high. This is CAN low. Here it will come down by 1 volt to 1.5, okay. And this will create this kind of a 2 volt difference, okay. So, this is the dominant state. Now what is the corresponding bit encoding? So, when you that the bus is in a dominant state, we interpret these as a logic 0.

And when we say that the bus is in a recessive state, we interpret that as a logic 1, okay. So, when actually there is not voltage difference between the 2 lines in the bus, so that is the logic 1. It is recessive, and when there is a voltage difference of 2 volts, then we have this as a logic 0 and that is the dominant state. Dominant state because you really have a signal difference here. Here we will just write it in better way.

(Refer Slide Time: 03:39)



Bus Synchronization: Bit stuffing

- CAN Encoding: Non-return to zero (NRZ)
  - Lots of consecutive zeros or ones leave the bus in a single state for a long time
  - In contrast, for a Manchester encoding each bit contains a transition
- NRZ problem: Not self-clocking
  - Nodes can easily lose bus synchronization
- Solution: **Bit stuffing**
  - After transmitting 5 consecutive bits at either dominant or recessive, transmit 1 bit of the opposite polarity
  - Receivers perform de-stuffing to get the original message back

Now something very important, that how the CAN voltage signal is going to represent a bit pattern. So, the CAN encoding type is called NRZ or Non-Return to Zero. So, as we know that in our digital signal, you have 2 possible ways of representing bit patterns. So, let us say this time and for each bit you have an allotted time period for the transmission. So, let us say you are writing a signal which is high from here up to this.

So, you can say that I have a 1, then a 1, like that. So, this is a Non-Return to Zero encoding because whenever it is high it is a 1. That is how am I interpreting it, right. The alternative would have been, this goes high and it comes down. Then for the next one again it will go high and come down. It is always going high at the edge of the clock, okay. Again, this going high here, and then coming down like that. So, you can see that we have a positive edge at the clock when the signal is going 1.

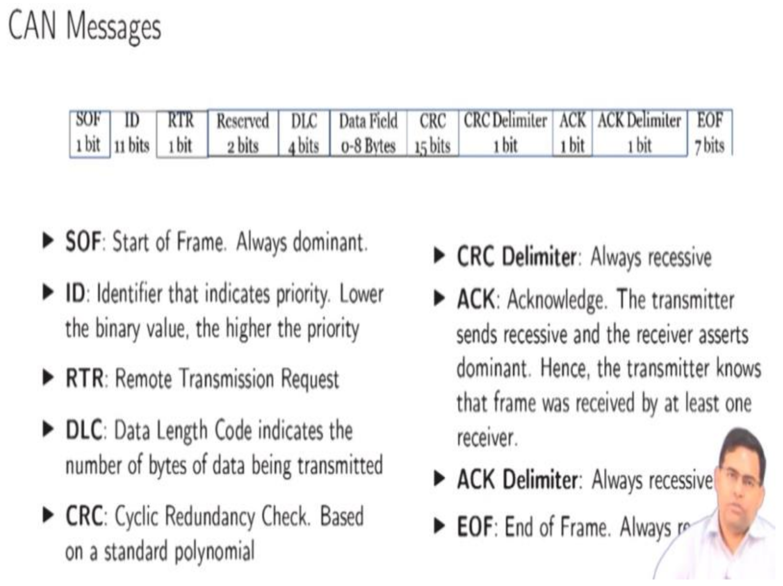
So, which is actually good for synchronization. So, that is a Return to Zero type encoding and the previous one that I showed is a Non-Return to Zero type encoding and CAN actually followed the Non-Return to Zero type encoding. But the problem with that kind of encoding is that it can it is easy to implement, it is cheap, but the bus, I mean, you may lose synchronization. The nodes may lose synchronization with the bus, for the simple reason that the signal is continuously 1 or the signal is continuously 0 and there is no switching activity happening.

The good thing about the switching activity is that the switching happens at the clock edge so that gives me some signal to synchronize with. As a node, I get a signal to synchronize with the bus. That well, this is where the bus is one transmission cycle is beginning, something like that.

So, what is the solution? The solution, so that it is not continuously enforced that I can be allowed that I am always a 1, where the rule says that well, if you have 5 consecutive bits, which are either dominant or recessive, you must have 1 bit in between of the opposite polarity. So, this is known as bit stuffing rule. So that means, that bit does not any related information. It is just a bit stuffing rule of CAN, which is breaking this pattern of continuous dominant or continuous recessive state.

And one bit in between of upwards polarity is also helping you to get the system synchronized back, because, and fundamentally why does the synchronization problem come? Because different nodes are having their different local clocks. And of course, like we know that clocks may slowly drift, right. So that leads to this whole problem of synchronization. But coming back here, let us just restrict ourselves to understanding that for long periods, if there is no clock edge that is happening on the bus, there is no switching activity, the individual nodes may lose their synchronization with the bus.

That means, they will be unaware of the exact times when one cycle of transmission starts or ends in the bus. That is why, to break this thing, this issue, break out of this issue, this rule exists, okay. Now of course this stuffed bit needs to be understood. That means the protocol will also do suitable de-stuffing of this bit from the original message at the reception side. **(Refer Slide Time: 07:49)**



CAN Messages

SOF	ID	RTR	Reserved	DLC	Data Field	CRC	CRC Delimiter	ACK	ACK Delimiter	EOF
1 bit	11 bits	1 bit	2 bits	4 bits	0-8 Bytes	15 bits	1 bit	1 bit	1 bit	7 bits

- **SOF**: Start of Frame. Always dominant.
- **ID**: Identifier that indicates priority. Lower the binary value, the higher the priority
- **RTR**: Remote Transmission Request
- **DLC**: Data Length Code indicates the number of bytes of data being transmitted
- **CRC**: Cyclic Redundancy Check. Based on a standard polynomial
- **CRC Delimiter**: Always recessive
- **ACK**: Acknowledge. The transmitter sends recessive and the receiver asserts dominant. Hence, the transmitter knows that frame was received by at least one receiver.
- **ACK Delimiter**: Always recessive
- **EOF**: End of Frame. Always re

Now let us open up a CAN packet and see what are its contents. So, this is typically how a CAN packet would look like. So, initially you have the Start of Frame bit. So, it is one bit which is always dominant, and it is telling you that one CAN frame has started. Then you have an identifier. So, identifier is an 11 bit field and that is, I mean, kind of indicating the priority of the message. So it will have a bit pattern that would also mean that if the bit pattern is having a lower value in binary then the priority of this message is higher, okay.

Now after that there is one bit called Remote Transmission Request, requesting other nodes. And then we have 2 bit in reserve for future usage. And then we have the Data Length Code, the DLC. Now, the DLC has a usage. If you see after the DLC you have a field called the Data Field. Now this is the actually packet, the payload, that you are sending inside this CAN packet, okay. So, this Data Field, it is written 0 to 8 bytes. That, I mean, we are not fixing a fixed number of bits in every packet for data transmission. We are saying that, well the data may be large, the data may be small.

I mean, it depends on the application, so let it vary from 0 to 8 bits. This is good for, from an embedded resource constant system point of view, because if there is less amount of data to process or send why book more number of bits in this field, right. And the data length code is actually telling me it's a 4 bit, its length is fixed, right. And it is telling me, that well, what is the size of the actual Data Field.

So, this is the smart decision to have some bits for DLC and having this some bits for DLC is actually helping you to keep the original Data Field, I mean, kind of variable in its length. So, after this you have the CRC, Cyclic Redundancy Check. So that is a standard CRC algorithms to a kind of, I mean, store the integrity of a message, that well, if you have a valid message.

Let us say your message is x, using a CRC polynomial you can compute some p(x) and these 2 things together you can send. The reason being suppose somebody is perturbing this x then at the reception side, somebody, who will be knowing the CRC polynomial, he can actually compute this p(x) for this modified x. Let us say it has become some x prime. And what will happen is whatever p prime he computes they will not match right from their it will be known that where whatever was sent has lost its integrity, okay.

So, this kind of checks, that is why, they are standard and they are implemented in CAN buses. Also, but it known that now a days that this kind of security measures are quite easy to break also. Now, the next thing is a CRC delimiter. So, this is one delimiter field. After this delimiter field there is an Acknowledgement bit. Now the transmitter sends recessive and the receiver will assert in dominant.

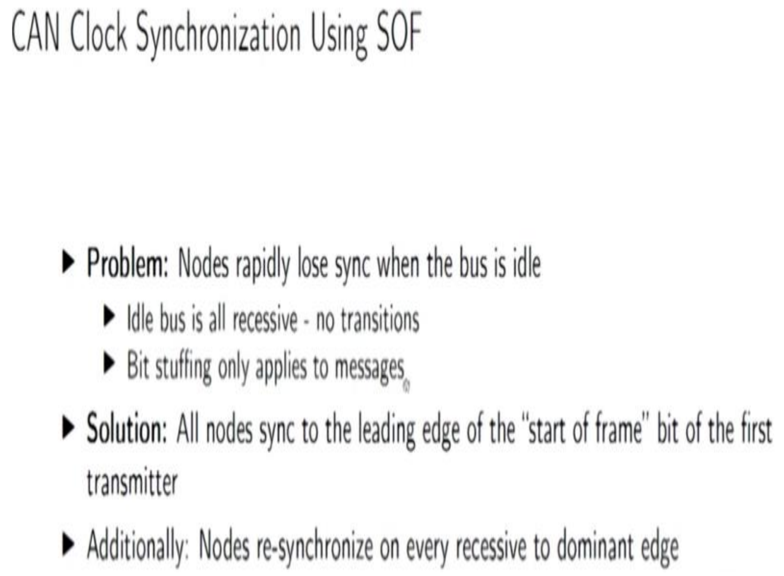
So, this has a standard. That whenever the transmitter is sending, in the acknowledgement field he will send a recessive signal. That means a voltage difference, I mean, that means a voltage difference of 0 would be set, right. Or a logic 1 would be set. And the receiver will assert a logic 0 that means a voltage difference of 2 that is the dominant bit. So, when the transmitter sees that well I have seen this in a recessive mode but it has become dominant then we you would know that well the receiver has indeed received it and acknowledged it.

So, in this way the transmitter knows that the frame has been received by at least one receiver. If whatever he is sending as an acknowledgement bit in this field is immediately getting, I mean, negated, I mean flipped in that CAN line. Because as we know that in the CAN line everybody can see and read the message and whenever somebody is sending, he can also simultaneously see that whether the message value is getting changed inside that same transmission cycle or not.

Now after this there is a delimiter, which is the Acknowledgement delimiter and this is always recessive. And after this there would again be as End of Frame of 7 bits and that is also recessive, okay. Now one way question what is the use of the delimiter? This delimiter tell you that there are certain fixed position in this bit pattern which would be recessive or which would be something different. So any change there would also tell you that well this packet has somehow been spoiled by some attacker or by some environmental noise or something.

You know that the end of frame has to be 7 bits recessive. If that is not so, then you know that the packet has issue. You know that there is a delimiter CRC at this position and then there is going to be a recessive bit if it is not so you know that there is an issue so and so forth.

(Refer Slide Time: 13:16)

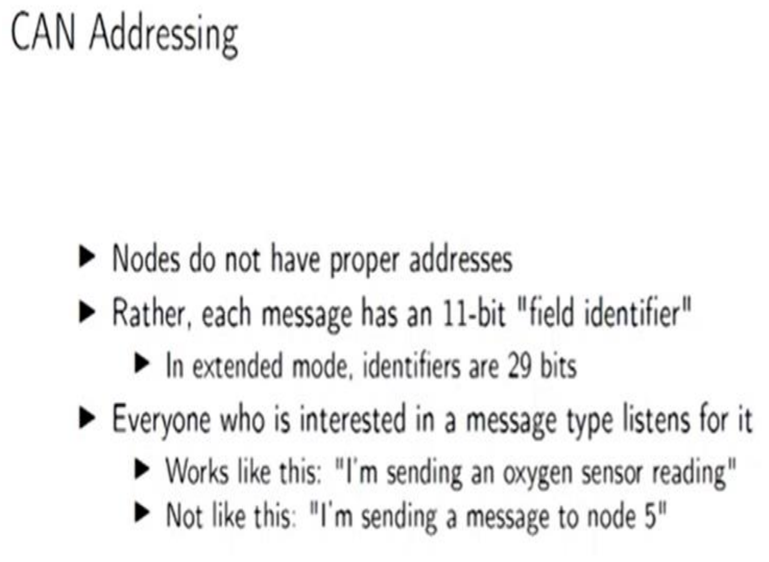

CAN Clock Synchronization Using SOF

- **Problem:** Nodes rapidly lose sync when the bus is idle
  - Idle bus is all recessive - no transitions
  - Bit stuffing only applies to messages
- **Solution:** All nodes sync to the leading edge of the "start of frame" bit of the first transmitter
- Additionally: Nodes re-synchronize on every recessive to dominant edge

So, the CAN clock is usually synchronized using this Start of Frame bit. Now this is nodes rapidly lose synchronization when the CAN bus is idle. Now the idle bus has no transitions and the idea of bit stuffing applies only to when the real messages are sent. So, when there is no message sent then the bit stuffing rule does not even apply, right. So, this is an issue, right. So, the nodes will also re-synchronize on every recessive to dominant edge.

So, if you see the Start of Frame is always dominant and dominant means that suppose there is a positive voltage difference that would get created, right. So, that is kind of a signal which is used for the nodes to get together synchronized and make that as a start of a transmission cycle in the system.

(Refer Slide Time: 14:11)


CAN Addressing

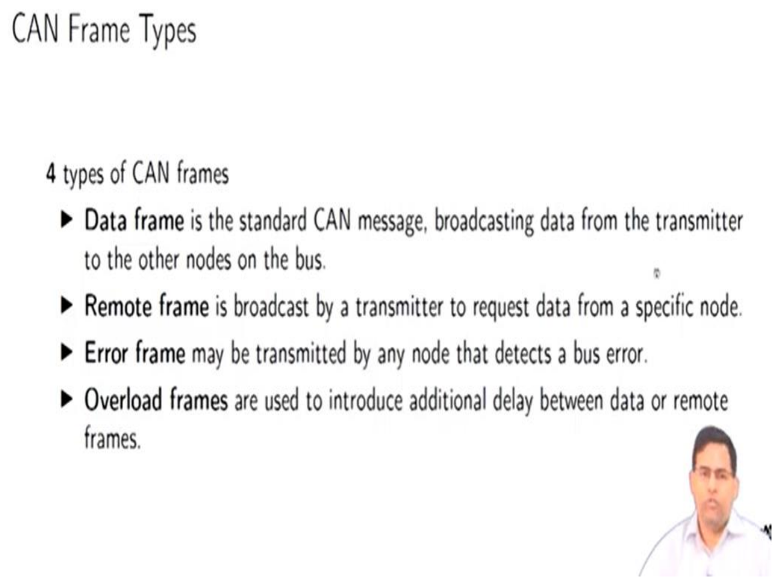
- Nodes do not have proper addresses
- Rather, each message has an 11-bit "field identifier"
  - In extended mode, identifiers are 29 bits
- Everyone who is interested in a message type listens for it
  - Works like this: "I'm sending an oxygen sensor reading"
  - Not like this: "I'm sending a message to node 5"

Now how do I address nodes? In computer science or any embedded real-time system design, whenever you have multiple things to connect, you would like to number them as 1, 2, 3 etc., like that, right. So, in this case you do not have addresses for nodes. Rather you have identifier fields for messages. What does a message represent? So, the way we interpret CAN messages is a bit different. So, it works like this, that, so suppose I am sending a message.

So, the thing I am trying to say is that well I am an ECU which is sending an oxygen sensor reading, okay. So, its not like that, I am saying that well this is the message and I am sending it to that node. So, this is all multi-cast communication. Everybody can see what everybody is transmitting. So, whenever I transmit, something I do not give this an address. That you send this value, I am sending this value to node X. Instead, I am just advertising that well this value I am sending is of this type and that type is kind of understood from that 11 bit identifier field, okay, which in an extended CAN system, it can be a larger.

Suppose and, why do I need a larger identifier? Suppose I have so many different type of messages right that the identifier field needs to be enhanced then there is also an option of having this kind of a 29 bit identifier. So, what is important is, what is the message type and the message type is defined by the identifier. It is not important that who is sending and to whom it is going things like that, okay. From the message type everybody knows that well this message of this type and accordingly he or she will get it read.

(Refer Slide Time: 15:53)



CAN Frame Types

4 types of CAN frames

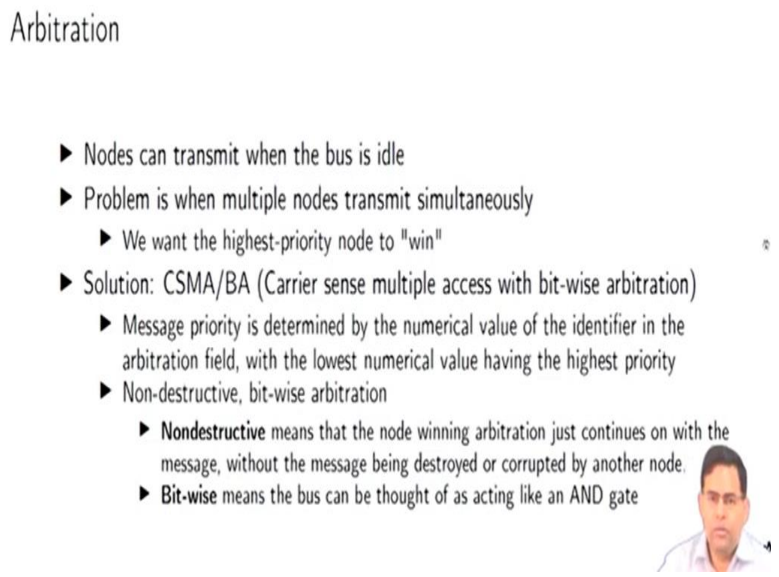
- **Data frame** is the standard CAN message, broadcasting data from the transmitter to the other nodes on the bus.
- **Remote frame** is broadcast by a transmitter to request data from a specific node.
- **Error frame** may be transmitted by any node that detects a bus error.
- **Overload frames** are used to introduce additional delay between data or remote frames.

So, there are multiple different types of frames in CAN. So, you have the Data Frame which is the standard CAN message, broadcasting data from the transmitter to the nodes on the bus, okay. And then you have a Remote Frame which is broadcast by a transmitter to request data from a specific node. So, Data Frame is just the standard CAN message that we discussed.

Remote Frame means, I mean, suppose some node I mean request data from a specific node so that would be a Remote Frame.

Error Frame is a special type of frame which is generated by a node when a bus error is detected, and Overload Frames are kind of frames which are used so whenever the system detects that this too many number of data packets from the bus. And they are not getting used and there is lot of I mean the data has no real consumer, one would like to insert some delay between future transmission and that is done by transmitting this kind of Overload Frames.

(Refer Slide Time: 16:56)



Arbitration

- Nodes can transmit when the bus is idle
- Problem is when multiple nodes transmit simultaneously
  - We want the highest-priority node to "win"
- Solution: CSMA/BA (Carrier sense multiple access with bit-wise arbitration)
  - Message priority is determined by the numerical value of the identifier in the arbitration field, with the lowest numerical value having the highest priority
  - Non-destructive, bit-wise arbitration
    - **Nondestructive** means that the node winning arbitration just continues on with the message, without the message being destroyed or corrupted by another node.
    - **Bit-wise** means the bus can be thought of as acting like an AND gate

Now we come to the issue we pointed out earlier. Like we said that CAN has a very nice arbitration scheme, right. So, this arbitration scheme will determine that which of the nodes attempting to transmit together will actually control the bus. And like we said earlier, that this is not so in CSMA/CD, Carrier Sense Multiple Access or Collision Detect type systems. Because in such systems anyone can transmit when the medium is idle and then if there is a collision that would be detected and then the current packet, I mean, this transmission will stop.

And then each of the nodes would back off for sometime, right. And this may also need to I mean then again they can start probabilistically at a future point of time. Now this may lead to collisions recurring, right. I mean in certain cases, also the issue is, what I mean, there is some random exponential back-off, so which does not tell me that, well is there a real guarantee that inside this time a message from here to here we must be going.

I do not really have such mathematical guarantee by design, okay. So, if there are repeated back-offs, if I am trying to get a worst-case guarantee which is often required in a real time system, that worst case guarantee will be pretty bad, right. So, repeated back-offs are not a good thing. Also there is no access priority. So, there is no well-defined rule like, and that was very much the reason why this problem happened.

Because there is no well-defined rule which says that when 2 parties are trying to transmit, who exactly can. Because in this case both of them can. So, in arbitration, nodes can transmit when the bus is idle and the problem like we said that happens when these multiple nodes transmit simultaneously. And what we want really is that well, there should be a priority-based arbitration happening and the node with the higher priority should win, right.

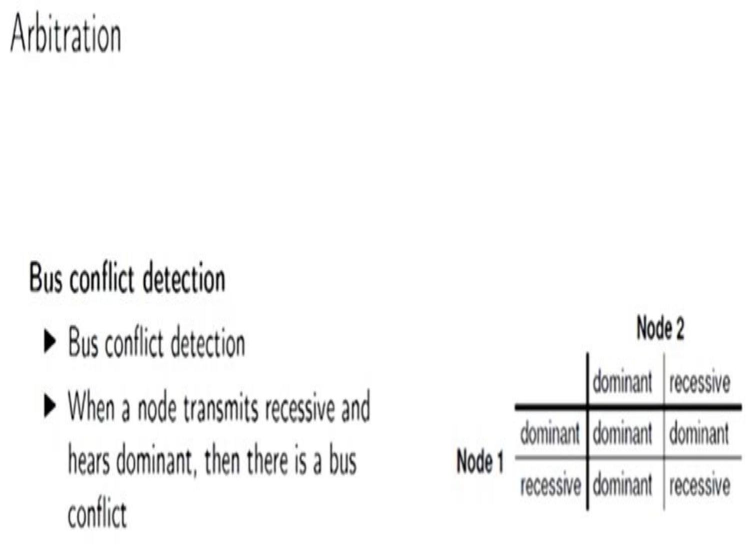
So, in case of CAN what we can do is, something like this. Carrier Sense Multiple Access with bit-wise arbitration and this is the things that we actually do here. So, priority will be determined by the numerical value of this identifier. If you remember we talked about this 8 bit identifier, right, in the arbitration field. And we will assign that well the lowest numerical value will have the highest priority.

And this should make things non-destructive and the arbitration also process will be also very elegant. So let us understand what is non-destructive. It means that well, eventually only one node would be sending the message. So first multiple nodes will try to send the initial identifier bits. And at that point of time, one of the nodes will win the arbitration. And that guy will continue to send this actual message and the other node will not see this message being sent and destroyed or corrupted by some other node. That thing will not happen.

So, this is a well-defined rule based Non-destructive scheme. And the arbitration decision is taken by looking at these identifier bits one after another in a serial manner, okay. So, this is why, it is called the bitwise arbitration. So, what is essentially done is something like you are looking at each of the bits one after another pairwise and doing something like a logical AND. The CAN bus, the way it is designed, the way we are conceived the dominant and the recessive bit, the recessive bits, if you remember, we took the dominant as to be 0, logical 0.

And logical 0 is represented by a voltage difference, whereas logical 1 is represented by I mean absence of any voltage difference between the twisted-pairs and that was a 0, right. So let us see that how this implements this kind of logical AND.

(Refer Slide Time: 21:02)



Arbitration

Bus conflict detection

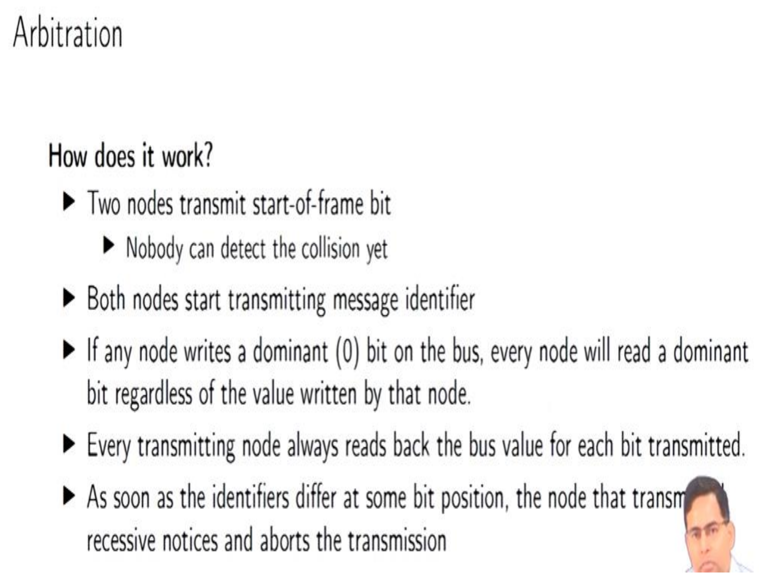
- Bus conflict detection
- When a node transmits recessive and hears dominant, then there is a bus conflict

		Node 2	
		dominant	recessive
Node 1	dominant	dominant	dominant
	recessive	dominant	recessive

So let us say, Node 1 and Node 2 both are in a dominant state. So that would mean the bus would also remain in a dominant state because there is a voltage difference. Now Node 1 is being recessive state, but Node 2 is creating a voltage difference. So that is what will be dominating the bus, okay. So, the result would be that the bus would be in a dominant state, right. Now if again similarly for Node 1 being recessive and Node 2 being dominant again the bus would be dominant and if the Node 1 output is recessive that means its sending a logical 1, which means 0 voltage in bus.

But node 2 is giving as logical 0, right. So that is already covered. But let us say both of them are recessive. That means both are giving a logical 1, then output is logical 1, right. So, as we can see that since we are interpreting this dominant as a logical 0 and there is and there is a voltage difference situation. So essentially its like a AND that is happening. The operation we are doing is AND with dominant being logical 0, right. And result is 1 only when both of the nodes are giving a logical 1, right.

(Refer Slide Time: 22:19)


Arbitration

How does it work?

- Two nodes transmit start-of-frame bit
  - Nobody can detect the collision yet
- Both nodes start transmitting message identifier
- If any node writes a dominant (0) bit on the bus, every node will read a dominant bit regardless of the value written by that node.
- Every transmitting node always reads back the bus value for each bit transmitted.
- As soon as the identifiers differ at some bit position, the node that transm recessive notices and aborts the transmission

So how does this really work? So, first of all you see I hope we have been clear that how the way the scheme is designed, how the way we have set the dominant and the recessive bits. It is logically implementing kind of an AND connection without really having any digital infrastructure or anything, right. So that is a nice thing about CAN and how it works is, the 2 nodes transmit the Start of Frame bit. Now both the Start of Frame bit are the same right I mean both are having the same polarity. So, there is no collision.

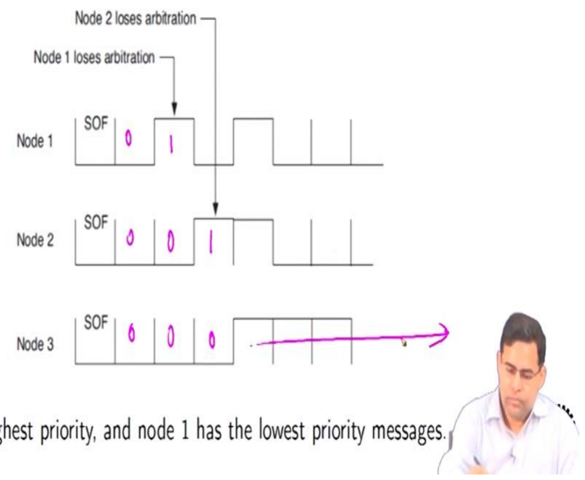
Now both nodes will send their transmitter start transmitting their message identifiers, right. Now whenever the one node keeps a dominant bit, what will happen is, every node will read a dominant bit regardless of the value written by the other node. Because of this property. One is sending dominant and this like a logical 0 and like an AND gate it is operating. By the physics of the system, now, there will be an output voltage on the bus.

So, every node we will see that output voltage and that means every node we see that bus being in dominant state. And it is independent of whether what are the other, if there is any other node which is sending recessive bit, okay. So, every transmitting node will always read back, the CAN read back the bus value, which is being transmitted. This is the property of, that means, what I transmit I can see what is happening in the bus.

So whenever 1 node finds that I am sending a recessive value but it is becoming dominant on the node on the bus. That means there is another high priority signal which is there in the bus and it will just abort its transmission. So as soon as the identifier differs at some bit position the respective node will just back off.

(Refer Slide Time: 24:10)

Arbitration: An Example

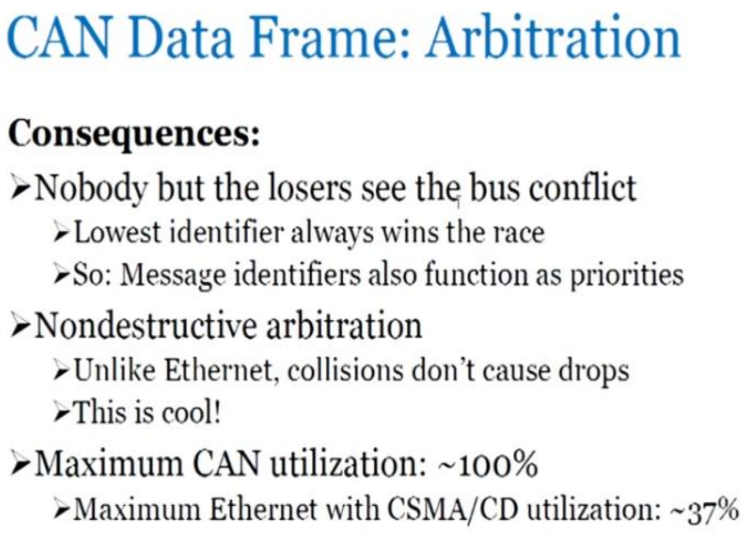


Node 3 has the highest priority, and node 1 has the lowest priority messages.

So here is an example picture here. So let us say you have multiple Node 1, Node 2, Node 3. So, Node 3 has the highest priority. And Node 1 has the lowest priority. How do we know? So, all of them are transmitting 0s here. So that is fine we progress to the next bit. This guy transmits the 1, they are transmitting 0. So, Node 1 has the lowest priority. That means its identifier value is high and Node 3 has the highest priority. That means its identifier value is the least, right.

So here we get a 1, and this is losing the arbitration and this guy will back-off. So, these 2 will remain and now let us say this guy sends a 1 and this is in 0. So again, he will lose the arbitration and Node 2 will back-off. So, after this the others will not send any future message and whatever message Node 3 will send, the highest priority node will send, that will continue. And that is how the arbitration system will work in CAN.

(Refer Slide Time: 25:28)



CAN Data Frame: Arbitration

Consequences:

- Nobody but the losers see the bus conflict
  - Lowest identifier always wins the race
  - So: Message identifiers also function as priorities
- Nondestructive arbitration
  - Unlike Ethernet, collisions don't cause drops
  - This is cool!
- Maximum CAN utilization: ~100%
  - Maximum Ethernet with CSMA/CD utilization: ~37%

So, what are the consequences here? So, nobody but the loser can see that there was a bus conflict, right. Because you see if you are the higher priority node, you do not even know that there was somebody whom against whom you won the arbitration, right. Because whatever you are sending you can see that is going. Only that how you lost he saw that well you are trying to set something as recessive but it become dominant, right.

So, you do not even see, but the loser sees that well, there is a difference in the polarity. So according the loser will abort respecting the protocol. The lowest identifier that is the highest priority will always win the race. So, message identifiers, in this way, they are also functioning as priorities. And this is non-destructive. Because there is no eventual collision that is happening and this is like a very cool feature here, right, because it costs you nothing. The voltage signals have been set in a clever manner so that it is coming this kind of arbitration is just coming for free.

That is a good thing. Now this helps you to have a protocol with very high utilization because you do not have collisions, right. So, you do not have collision related transmission etc., until and unless there is an error, packets will be transmitted. Only in case of errors in the transmission due to some other physical issue there can be the requirement of retransmissions.

So that is why, in CAN you can have a pretty much high utilization which is not so in other cases of let us say CSMA/CD based things, like maximum is an Ethernet, where it will be maximum like 37% or something like that. So, with this we will stop the lecture here and we will start from this point again in the next lecture. Thank you.