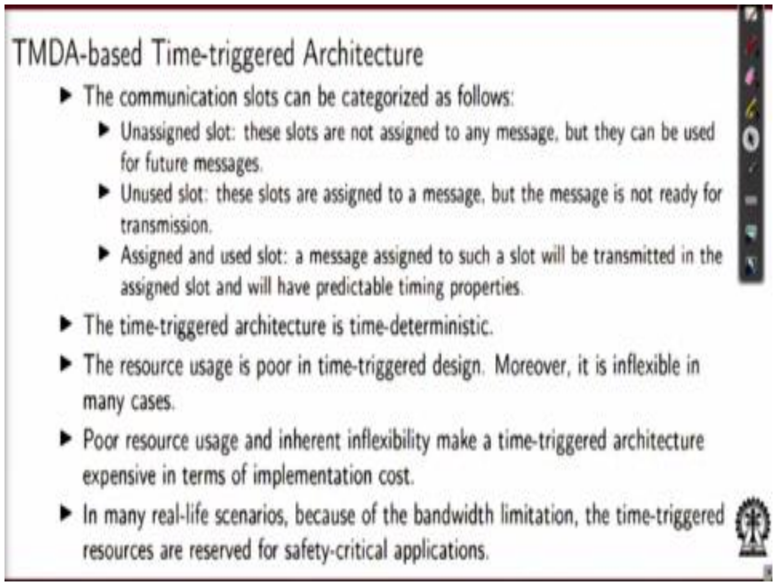
Foundation of Cyber Physical Systems

Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 16
Real Time Task Scheduling for CPS (Continued)

(Refer Slide Time: 00:29)

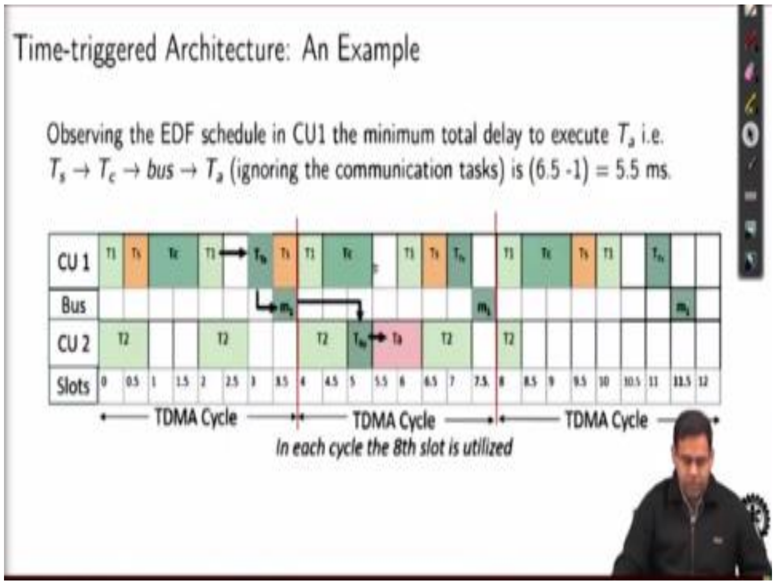


TMDA-based Time-triggered Architecture

- The communication slots can be categorized as follows:
  - Unassigned slot: these slots are not assigned to any message, but they can be used for future messages.
  - Unused slot: these slots are assigned to a message, but the message is not ready for transmission.
  - Assigned and used slot: a message assigned to such a slot will be transmitted in the assigned slot and will have predictable timing properties.
- The time-triggered architecture is time-deterministic.
- The resource usage is poor in time-triggered design. Moreover, it is inflexible in many cases.
- Poor resource usage and inherent inflexibility make a time-triggered architecture expensive in terms of implementation cost.
- In many real-life scenarios, because of the bandwidth limitation, the time-triggered resources are reserved for safety-critical applications.

Hello and welcome back to this lecture series of Foundations of Cyber Physical Systems. So, in the last lecture we had been discussing about, I mean, the bus slot allocation for TDMA-based time-triggered architecture. So, in these systems, the communication slots can be categorized as follows, I mean, you have unassigned slots, because there may be slots where you do not have any messages assigned. And you can have slots where you have messages assigned, but those slots may not be occupied in every cycle, okay. Or maybe the message is not ready for the transmission and you can have assigned a new slot, that means a message is assigned to such a slot and will be transmitted in that assigned slot and it will have a predictable timing property. So, you can have these different scenarios there. Now this time-triggered architectures are nice in the way that they are extremely time deterministic.

(Refer Slide Time: 01:22)



Time-triggered Architecture: An Example

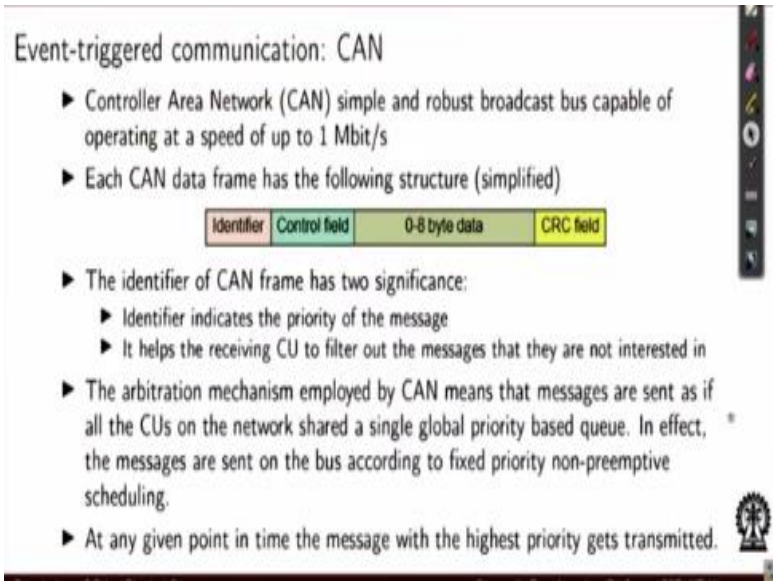
Observing the EDF schedule in CU1 the minimum total delay to execute T_a i.e. $T_s \to T_c \to bus \to T_a$ (ignoring the communication tasks) is (6.5 -1) = 5.5 ms.

For example, if you look at our previous discussion, we assigned m_1 to this slot, okay. And in every cycle, it is occupying this slot. But maybe, I mean, the system of task is such that well, I have assigned m_1 to this eighth slot in the TDMA cycles, but the message periodicity is not really 4, okay. The message periodicity is 8 or 12. Then what would mean is the next TDMA cycle, this slot is unoccupied or if the message periodicity is let us say 12, then well, I am going to really use this slot, these 8 slots of the TDMA cycle once in every 3 TDMA cycles, right. So, that is the point that I am reserving slots and that is helping me to create the communication extremely deterministic, I know exactly when it will happen and exactly when the message should be received. All these things are known to me, but there may be situations where these slots are unused.

I just give an example. suppose I will just repeat if m_1 has a periodicity let us say here m_1 period is it is 4, which is same as the bus cycle. But let us say m_1's periodicity is 12, okay, then although I reserve one slot here. Message is actually sent in that slot once every 3 cycles in other. So, it is basically a 33 percentage of usage and 63 percent of wastage of the time slot, right. So, that is why we say that it is poor in terms of resource usage. TDMA is deterministic nice real time, but its poor in terms of resource usage, right.

And it is inherently inflexible it cannot allocate the slot at the run time at the run time I cannot allocate the slot when it is empty to somebody else. So, this makes it expensive also, in real life scenarios because of bandwidth limitation, they are reserved such time-triggered resources are reserved for safety critical applications. That means applications where the message sending has to be absolutely time deterministic. There we will have such reserve slots but otherwise we may look for more optimized systems.

(Refer Slide Time: 03:28)



Event-triggered communication: CAN

- Controller Area Network (CAN) simple and robust broadcast bus capable of operating at a speed of up to 1 Mbit/s
- Each CAN data frame has the following structure (simplified)

Identifier	Control field	0-8 byte data	CRC field

- The identifier of CAN frame has two significance:
  - Identifier indicates the priority of the message
  - It helps the receiving CU to filter out the messages that they are not interested in
- The arbitration mechanism employed by CAN means that messages are sent as if all the CUs on the network shared a single global priority based queue. In effect, the messages are sent on the bus according to fixed priority non-preemptive scheduling.
- At any given point in time the message with the highest priority gets transmitted.

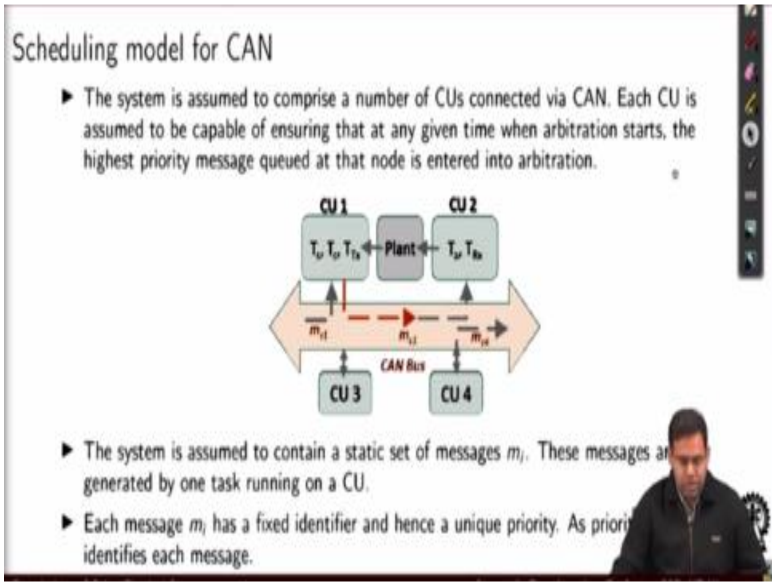
So, that is why people can also choose to do event triggered communication, that based on some event you choose that well I will send a message. One such example is our CAN bus, which we have discussed earlier also. Now we will go into timing analysis of CAN messages in this specific topic here. So, as we have found that CAN is simple robust and it is a broadcast bus, because any message you send over CAN is kind of visible to everybody, right.

And let us just recap how the CAN data frame look like. It has identifier, the control field and then the payload that is 0 to 8 byte data and then you have CRC, right. This identifier field have some significance there if you remember. So, it gives a priority to the message. The lower the value of the identifier, the higher is the priority. And it helps the receiving compute unit to filter out this message that they are not interested in.

Because based on the identifier the compute unit knows whether it wants this message or not. Now also if you remember the arbitration mechanism which we discussed earlier for CAN which is quite nice it is a simple physical mechanism. And messages are sent as if to all the CUs, the compute units or the control units. So, effectively what we have is like a global priority queue, right. In effect, these messages are sent on the bus according to a fixed priority non-preemptive schedule, right.

Because when you send, you are sending the message with the highest priority and priority of the messages are never getting changed. And while the message being is being sent, it will not be preempted. So, that is why its non-preemptive also, right. But initially whenever there is a contention the message with the highest priority will get sent as the bus arbitration mechanism will enforce.

(Refer Slide Time: 05:27)



Scheduling model for CAN

- The system is assumed to comprise a number of CUs connected via CAN. Each CU is assumed to be capable of ensuring that at any given time when arbitration starts, the highest priority message queued at that node is entered into arbitration.

- The system is assumed to contain a static set of messages m_i. These messages ar generated by one task running on a CU.
- Each message m_i has a fixed identifier and hence a unique priority. As priori identifies each message.

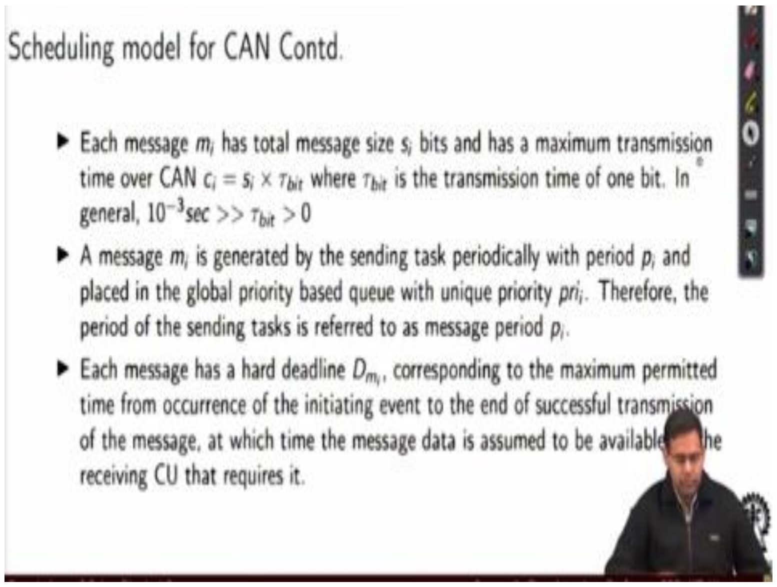
So, that is how CAN work. So, with that knowledge if we try to do some timing analysis let us progress on that front. So, let us assume a system now where you have multiple compute units and they are now connected by a CAN instead of something like TDMA we discussed earlier, right. So, each compute unit is assumed to be capable of ensuring that at any given time when the CAN arbitration starts, the highest priority message queued at that node is entered into the arbitration.

So, that means inside the compute unit, there may be multiple messages that are produced. But it will do an arbitration among those messages internally and drop the highest priority message into the CAN bus. So, that now, it will be a race among messages coming from multiple compute units, those are highest priority inside their compute units. But now they have to compute among messages from other highest priority messages in compute, in separate compute units, right.

So, let us say in that way for each compute unit you have a set of messages that have been dropped. I mean, each computer will send some message through their internal arbitration. And now they are going to win, choose among those messages on the bus, right. So, let us say you have a set of such static messages m_1, m_2 like that and every message is generated by a task running on a compute unit. Every message has a fixed identifier, based on the CAN protocol.

And that means every message has a unique priority. We do not have two messages of the same priority by design of the system, okay. And like we discussed earlier the priority tells me what is the message, say, some priority value will tell this is a fuel reading, (07:10) some priority will tell that, this is the air to fuel ratio reading things like that.

(Refer Slide Time: 07:14)



Scheduling model for CAN Contd.

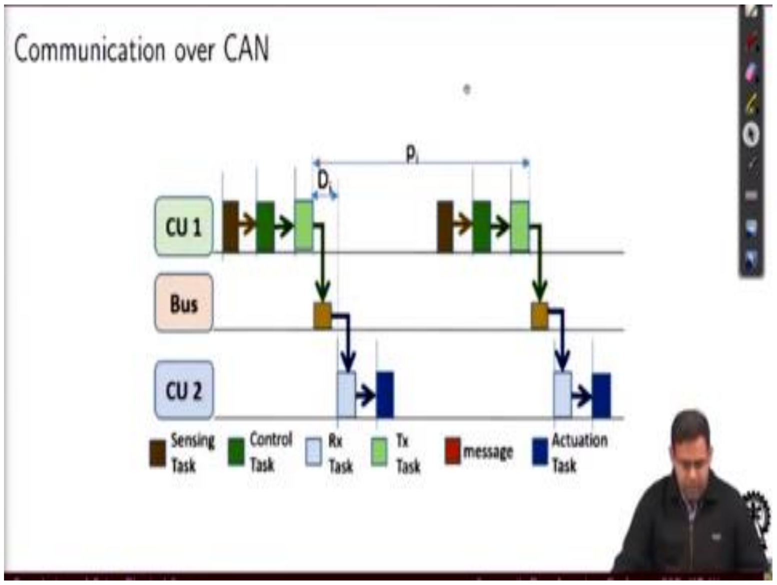
- Each message m_i has total message size s_i bits and has a maximum transmission time over CAN $c_i = s_i \times \tau_{bit}$ where τ_{bit} is the transmission time of one bit. In general, $10^{-3}sec >> \tau_{bit} > 0$
- A message m_i is generated by the sending task periodically with period p_i and placed in the global priority based queue with unique priority pri_i. Therefore, the period of the sending tasks is referred to as message period p_i.
- Each message has a hard deadline D_{m_i}, corresponding to the maximum permitted time from occurrence of the initiating event to the end of successful transmission of the message, at which time the message data is assumed to be available [illegible] the receiving CU that requires it.

So, each message will have a total message size let us say of s_i bits and the maximum transmission time over CAN. So, let us say tau bit is the transmission time of 1 bit, okay, which is of course very small negligible here. Message is generated by send, by the sending task periodically with a period p_i, that means whatever is the period of the task will be the period of the message also. And the messages are placed in a global priority queue with some unique priority.

So, for the ith message I have a period p_i and priority pri_i something like that, and each message will have a hard deadline, like this D_{mi}, okay. And that means the message transmission must happen inside this hard deadline. And corresponding to this maximum permitted time for occurrence of this initiating event to the end of the successful transmission. That means inside this time, this message must get transmitted, it is that means, I mean, when do I say the transmission is finished?

So, that is marked by the time when the message data is available to the receiving control unit whichever is supposed to, I mean require that message and act on the message. So, that is essentially a response time of the CAN message. That means the initiating event from when the message got generated up to the time the message has been successfully received by the target compute unit. So, this interval is what is the response time, right, and it must be less than this given hard deadline for the message.

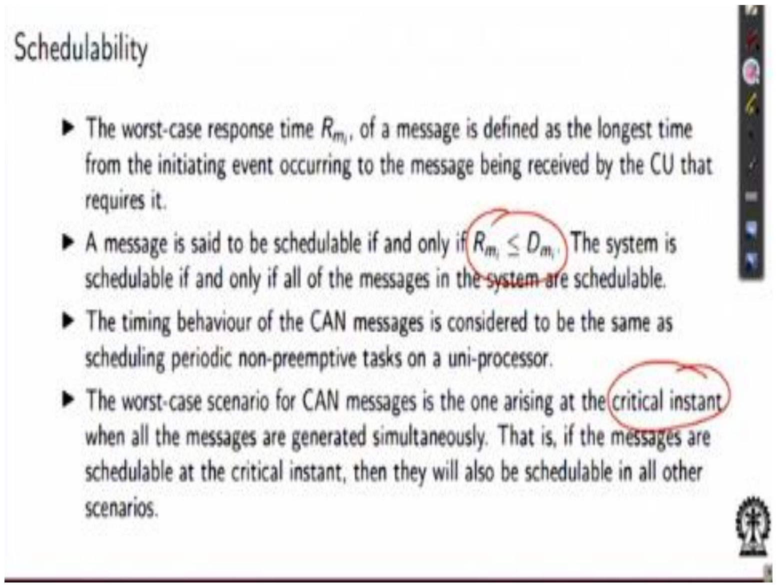
(Refer Slide Time: 09:00)



So, here we again have that pictorial picture which is just trying to tell you that well, let us say in compute unit 1 you have some sensing then some control task and then the control command is generated and then there is a transmission task, this guy, and then there is the transmission over the bus of the message and this is the deadline of the transmission that is D_i, right. And then once this is received there is a reception task which should run and then some actuation task run.

So, that same figure where we are just trying to tell you that well if the tough control task has period p_i these messages sending will also have a period p_i, I mean, the time between two messages of same id getting generated on the bus will also have the same period of course. And for each of them there will be a deterministic deadline provided inside which the bus, the bus must be able to transmit the message, okay.

(Refer Slide Time: 10:03)


Schedulability

- The worst-case response time R_{m_i}, of a message is defined as the longest time from the initiating event occurring to the message being received by the CU that requires it.
- A message is said to be schedulable if and only if $R_{m_i} \leq D_{m_i}$. The system is schedulable if and only if all of the messages in the system are schedulable.
- The timing behaviour of the CAN messages is considered to be the same as scheduling periodic non-preemptive tasks on a uni-processor.
- The worst-case scenario for CAN messages is the one arising at the critical instant when all the messages are generated simultaneously. That is, if the messages are schedulable at the critical instant, then they will also be schedulable in all other scenarios.

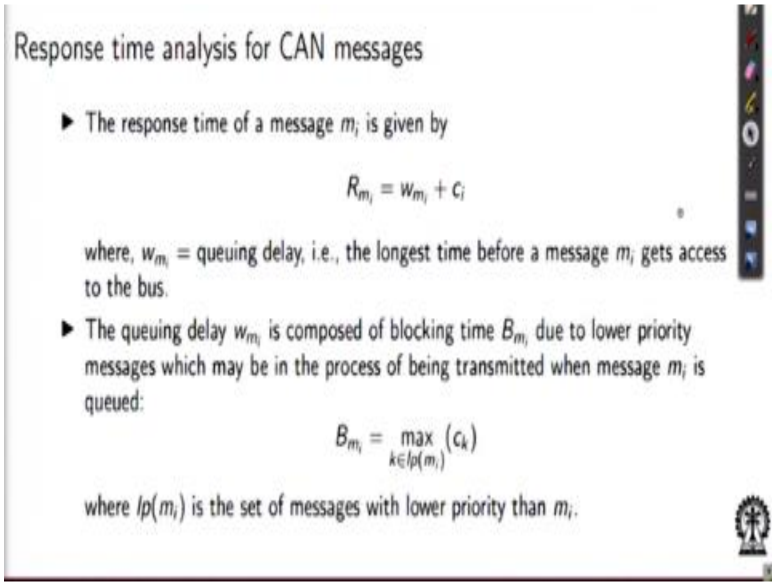
So, the worst-case response time for the message m_i, let us call it R_{mi} is defined as the longest time like we discussed from the initiating event when the initiating event here is this, this task executing will generate this message. So, from that up to the time when the message is finally received and we will require this thing to be satisfied, right, like we have already discussed. And we will say that the entire system is schedulable if all the messages are schedulable.

I mean, now the timing behaviour of this CAN message is considered to be same as the scheduling of non-preemptive (10:48) tasks. So, if you see what we are trying to say is well how do I do a timing analysis on the CAN bus. If you remember we have already established that the CAN bus messages are fixed priority and non-preemptive, right. So, whatever is our method of doing scheduling and timing analysis for such task set fixed priority non-preemptive, the same will apply for can messages also. So, we can apply the same method to compute the worst-case response time for messages on the CAN, okay. But there is a catch here. We have to identify what is the worst-case scenario that the message can, the message can face, right. Because if the message is schedulable on the CAN bus, that means the message transmission, the response time is inside the deadline in the worst-case scenario, then it should be schedulable and it should be responding within the deadline for any other scenario. Now this worst-case scenario for every CAN message is when it is facing maximum interference from other messages. Like a task faces interference from other tasks. And this maximum interference can happen when we have that critical instance.

So, we call this critical instance, there is the time when all the messages from other tasks let us say they are generated simultaneously, okay.

Now if we can show that at that critical instance, we can satisfy this thing, then we are done, right. We can say that well the CAN bus will be able to schedule this specific message at all other instances, when such other interfering messages will not be present.

(Refer Slide Time: 12:35)



Response time analysis for CAN messages

- The response time of a message m_i is given by

$$R_{m_i} = w_{m_i} + c_i$$

where, w_{m_i} = queuing delay, i.e., the longest time before a message m_i gets access to the bus.

- The queuing delay w_{m_i} is composed of blocking time B_{m_i} due to lower priority messages which may be in the process of being transmitted when message m_i is queued:

$$B_{m_i} = \max_{k \in lp(m_i)} (c_k)$$

where $lp(m_i)$ is the set of messages with lower priority than m_i.

So, let us do this CAN bus level we worst-case response time analysis. So, a few things here. Response time of a message, we are saying that well there would be a queuing delay, that means, other higher priority messages may get transmitted. We have to figure that out. And then what is the exact time when this message will get transmitted, its own transmission time, which is this the message of size (13:02) S_i bits would get, take C_i amount of time, real time on the CAN bus.

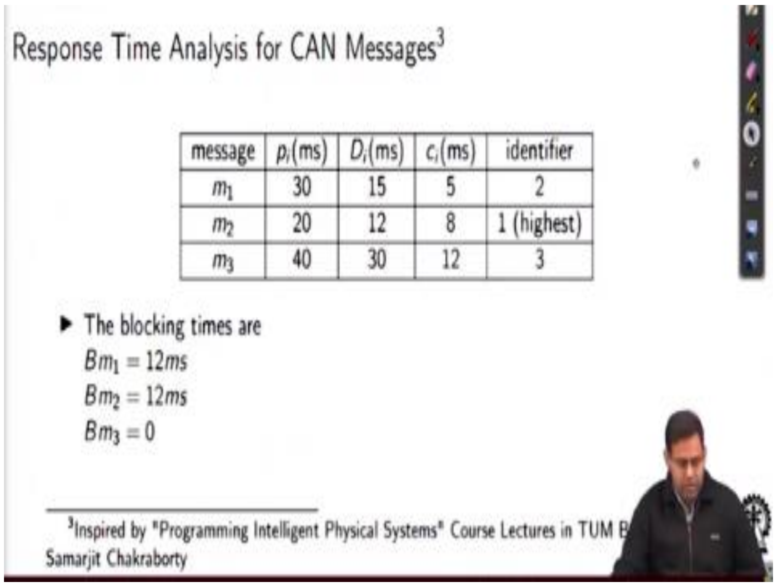
So, whatever is the delay that it is getting that $W_{mi} + C_i$. So, C_i we know from the message size and the transmission time etc. etc. So, we need to figure out this W_{mi}. This queuing delay that a message can get it can be its composed of two things. One is a blocking time and then the interference from other higher priority type. So, what is the blocking time? Blocking time means suppose when this message instance came at that time, some lower priority message is already executing.

Now if you remember, we said that this is fixed priority non-preemptive scheduling. Because in CAN, if something is already undergoing transmission, nothing can preempt it, right. So, this specific thing that somebody of lower priority is executing that means, a task which is blocking me, right. If somebody of higher priority is executing that is not blocking me because its right to execute, right. So, this queuing delay will have two parts, one is the blocking time.

And blocking time is nothing but with respect to this message, this message m_i, right, you figure out what are all the lower priority tasks. So, that set is given by this $lp(m_i)$. So, this is containing for the ith task what are the tasks which are of lower priority than this. So, any one of them can be executed right now, so for every such case task its transmission time is c_k, and what is the maximum among them, right.

So, that is the maximum blocking I can get, right. So, among all tasks which are of lower priority whichever has the highest execution time will block me most. So, that is the blocking time this specific message can face. So that is why that is how I get the amount of time I may be blocked, right.

(Refer Slide Time: 15:00)



Response Time Analysis for CAN Messages[3]

message	p_i(ms)	D_i(ms)	c_i(ms)	identifier
m_1	30	15	5	2
m_2	20	12	8	1 (highest)
m_3	40	30	12	3

- The blocking times are
  $Bm_1 = 12ms$
  $Bm_2 = 12ms$
  $Bm_3 = 0$

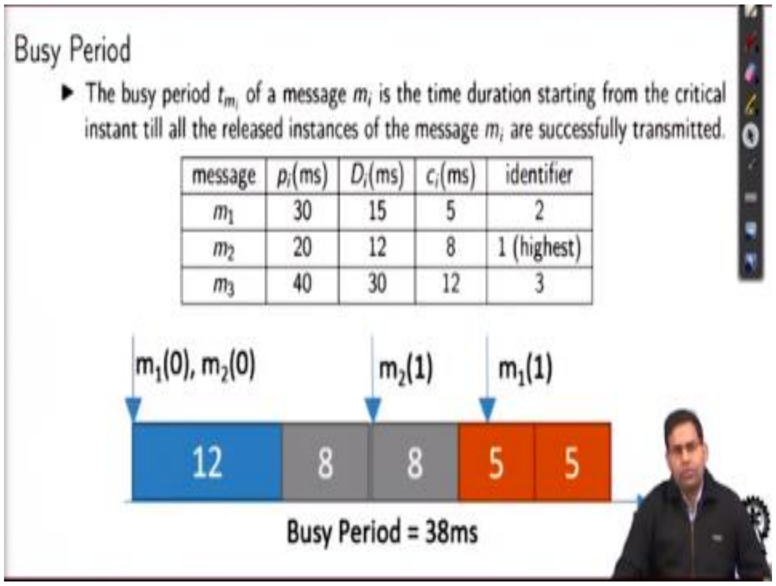
[3]Inspired by "Programming Intelligent Physical Systems" Course Lectures in TUM B Samarjit Chakraborty

So, let us take this example of messages. So, I mean, this example has been taken from some lectures by Professor Samarjit Chakraborty of TUM, I mean, unit and from the course Programming Intelligence Physical Systems. And if you see these messages have their periodicities it is 30, 20 and 40. And these deadlines are like 15, 12 and 30, right. So, inside these times these messages should be transmitted and these are the periods.

And for these messages you have the transmission time given by 5, 8, 12, like that. And let us say these messages have this identifier to 1, that is the highest. So, this is the highest priority message and this is the identifier 3, okay. Now these blocking time values, so how do I compute? So of course the message with the lowest priority that will not have, that won't be blocked by anybody, right. So, the message with identifier 3 for each blocking time is 0.

The message 2 has the highest priority. It can be blocked by any of them, right, any of the lower priority ones. So, message two if you see. So, message 2 can be blocked either by m_1 or m_3 so whichever has the highest execution time is 12. So, blocking time is 12. Similarly, message 1 can only be blocked by the lowest priority 1, m_3. And that is that also has, that is what the execution time 12, as the transmission time 12. So, for that also the blocking time is 12.

(Refer Slide Time: 16:45)



Busy Period

- The busy period t_{m_i} of a message m_i is the time duration starting from the critical instant till all the released instances of the message m_i are successfully transmitted.

message	p_i(ms)	D_i(ms)	c_i(ms)	identifier
m_1	30	15	5	2
m_2	20	12	8	1 (highest)
m_3	40	30	12	3

So, if you take this kind of an example, what we now do is we compute the busy period of a message. So, if you remember, when we talked about queuing delay, there are two parts into it. The first part is the blocking time and we talked about the how to compute the blocking time. The next part is the busy period. That means the amount of time this message can get, transmit, I mean, get interference from other messages, okay, of higher priority.

So, the busy period for this message is the time duration starting from the critical instant, that means when all other messages have arrived, up till all the release instances of m_i are successfully transmitted, okay. So, let us understand this very critically. We have already understood what is this queuing delay. That means, the message is delayed by this previous low priority messages which may be this just started executing when this message came, okay.

And then let us take this task system m_1, m_2 and m_3. And let us see we are trying to create a busy period situation here. So, we have this m_1 and m_2 arriving here, okay. But let us say right now, m_3 is already executing, right. So, m_3 will eat up this time of 12 milliseconds, right. And after this, of course m_2, which has the highest priority, that going to execute. So, m_2 which has the highest priority that is executing so.

And that has the execution time of 8. I mean, executing means is getting transmitted on the bus, sorry. So, it will have this transmission time of 8 millisecond, right. Now see, so m_2's instance is getting transmitted here. That means m_1 which has the second highest priority, its 0th instance is still pending, right. So, now once m_2's message transmission is done, but exactly at that time you see this 12 plus 8, 20 and m_2 has a period of 20 also. So, m_2's second instance or m_2^0 and m_2^1, is the second instance that is arrive.

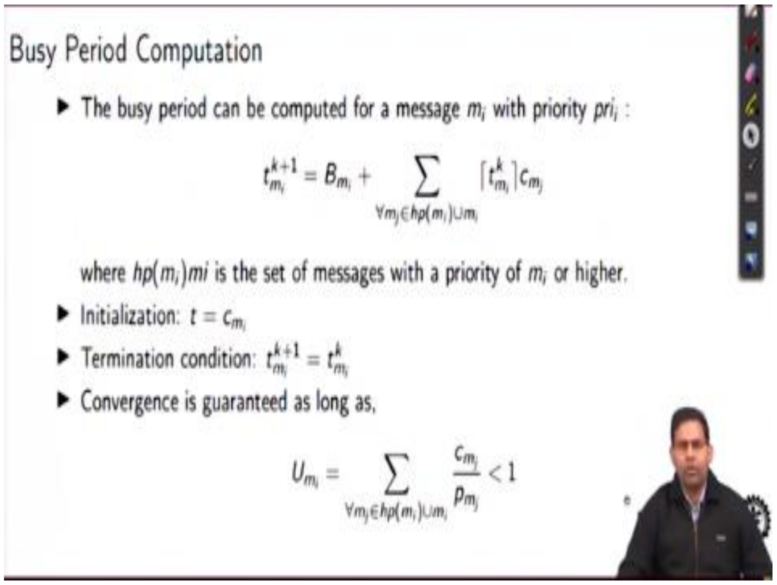
So, being of highest priority that will again execute, right. So that will again execute and now you see, once this is done after that what is pending, well, m_2 has nothing pending again. Because it is again going to come after another 12 millisecond time from this point. m_3 is not going to come right now. But m_1 has an instance, I mean, m_2 has an instance which was pending here, right. Sorry m_1 has an instance which is pending right from here, right.

So, now what will happen is m_1's this instance will start executing. m_1's zeroth instance will start executing. So, now m_1's zeroth instance is executing. But if you see m_1 has a periodicity of 30, right. So, 12 + 8, 20, + 8, 28, and 5, 33. That means, somewhere in between m_1's zeroth instance getting executed, m_1's first instance or basically, I mean, you can call zero and one or one or two whatever the next instance of m_1 has also got released.

So, by the time m_1 zeroth instance has finished, we already have another of m_1's release instant still pending. So that would now execute. So, if I am now trying to tell, well what is the total busy period of this message m_1 is basically this entire duration. It is not up to this. If you look at this definition, the busy period of the message is the time duration starting from the critical instant which is this till all the released instances of the message are successfully transmitted.

So, if I am talking about the busy period for m_1, it is starting here and it is going to end at the point where all the released instances are successfully transmitted. So, when m_1's first instance actually got to execute, I mean, before it got completed there was another released instance, and that then executed. So, up to this which is 20 +, I mean, 20 + 8 + 10 that is 38. So, that is the entire busy period here.

(Refer Slide Time: 21:30)



Busy Period Computation

- The busy period can be computed for a message m_i with priority pri_i :

$$t_{m_i}^{k+1} = B_{m_i} + \sum_{\forall m_j \in hp(m_i) \cup m_i} \lceil t_{m_i}^{k} \rceil c_{m_j}$$

where $hp(m_i)mi$ is the set of messages with a priority of m_i or higher.

- Initialization: $t = c_{m_i}$
- Termination condition: $t_{m_i}^{k+1} = t_{m_i}^{k}$
- Convergence is guaranteed as long as,

$$U_{m_i} = \sum_{\forall m_j \in hp(m_i) \cup m_i} \frac{c_{m_j}}{p_{m_j}} < 1$$

Now when I am going to compute this entire delay that the message can have is going to be this blocking time plus the busy the, this entire period, right. So, effectively this is the blocking time, right. And then you have the rest of the interfering time and they together is the your overall busy period for the message, okay. That entire queuing delay of the message. So, now how do I compute the busy period in general? If you see it has a similar kind of idea.

So, you have an estimate for some message m_i with a priority pri_i, right, and periodicity p_i, message m_i, ith message basically. It has a periodicity pri_i, period p_i, like that. Let us say I have made an estimate. In the kth iteration of this equation,

$$t_{m_i}^{k+1} = B_{m_i} + \sum_{\forall m_j \epsilon hp(m_i) \cup m_i} \lceil t_{m_i}^{k} \rceil c_{m_j}$$

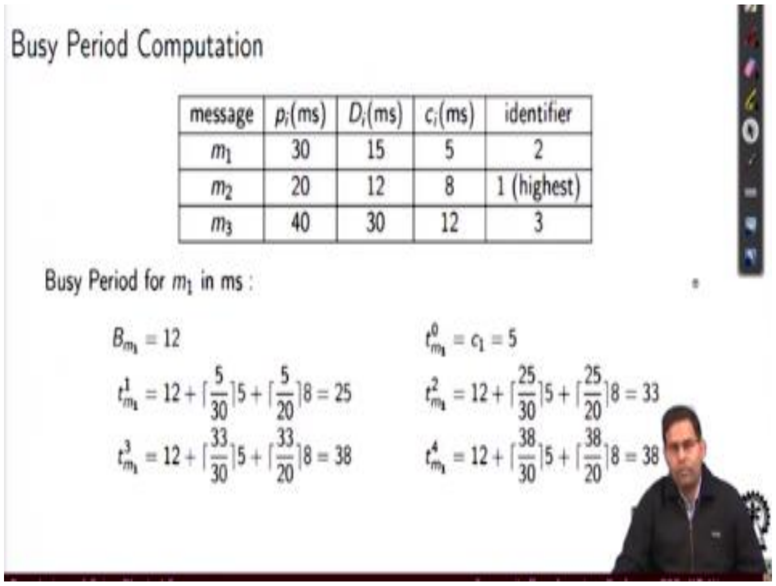
the estimate is $t_{m_i}^{k}$ and I am trying to update it with this $t_{m_i}^{k+1}$, right. So, they will be related like this blocking time for this example the blocking time is 12 here, the blocking time plus, so, this blocking time is actually. So, this busy period or my total queuing delay is comprising two things here as you can see the blocking time and the interference I am getting from the higher priority messages. Because why this is happening first when I am talking about m_1 it will be getting blocked by something having lower priority than it, which is m_3. And then again it is it is unable to get to the CPU, because the CPU, sorry, the bus, because the bus is busy transmitting messages of higher priority. And then it got the bus. But then before its own completion another of its instance came and that also has to be accounted for. So, that is how the total busy period is to be computed, first account for the lower priority message delay and then account for messages which are of higher priority, okay. And also account for your own release instances, so this is a bit different as you can see.

So, what you do is here, this is for messages. So, you consider all messages which are of higher priority that is given by this set $hp(m_i)$, and then you also include yourself, right. So, that is why you include yourself also. So here you have a union. So, basically what you are doing is, you are considering all messages of priority higher than you and yourself, okay and for such all such messages you compute, you get this estimate of t^k, and you multiply that with the execution time, sorry, the transmission time of that corresponding message. So, see, we are making this set of higher priority or this same message and equal priority messages, okay. And then you are accounting for such messages, what is the current estimate. So, these messages are represented by m_j, right, and then you are multiplying this transmission time estimate that you have currently with this c_{m_j}, okay.

Now you will initialize this with your own transmission time and of course the termination condition is this, that is after two consecutive iterations you get the same estimate, right. Now do I have a convergence guarantee on this computation? Well, as long as you have a feasible schedule of these things, that means, as long as you have a utilization feasible, that means, this summation I mean of utilization for the messages of the current message.

And all other messages which are of higher priority than it for all of them, their transmission time by the period this summation is less than 1 you are guaranteed to get a convergence of this system of equation.

(Refer Slide Time: 25:35)



Busy Period Computation

message	p_i(ms)	D_i(ms)	c_i(ms)	identifier
m_1	30	15	5	2
m_2	20	12	8	1 (highest)
m_3	40	30	12	3

Busy Period for m_1 in ms :

$$B_{m_1} = 12 \qquad t^0_{m_1} = c_1 = 5$$

$$t^1_{m_1} = 12 + \lceil \frac{5}{30} \rceil 5 + \lceil \frac{5}{20} \rceil 8 = 25 \qquad t^2_{m_1} = 12 + \lceil \frac{25}{30} \rceil 5 + \lceil \frac{25}{20} \rceil 8 = 33$$

$$t^3_{m_1} = 12 + \lceil \frac{33}{30} \rceil 5 + \lceil \frac{33}{20} \rceil 8 = 38 \qquad t^4_{m_1} = 12 + \lceil \frac{38}{30} \rceil 5 + \lceil \frac{38}{20} \rceil 8 = 38$$

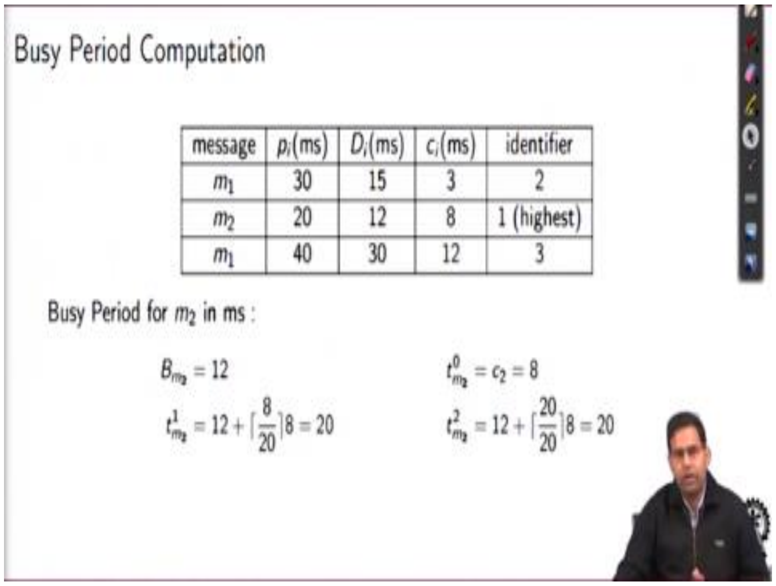
So, let us take an example of this busy period computation. So, essentially when I get to compute this busy period, what am I really doing is I am trying to identify the entire time inside which, I mean, how many I mean, all the released instances of the message get transmitted, and this will have an effect on my calculation of response time. We will see how. So, let us first get through the busy period computation.

So, now for this system of messages, if I compute the busy period, you see what comes. So, first of all I have a blocking time, that is 12 and I have my initial busy period estimated as 5 which is the execution time of m_1, sorry, transmission time of m_1. So, then you apply this form the m_1 and the higher priority message from that is m_2, right, which has a period 20. So, if you see, you have this T_1, problem here sorry, so this should be otherwise there is no meaning of the ceiling.

There is this period the current estimate must be divided by the period. And you take the ceiling over that of course that is how it is, sorry for this mistake here and then we go forward. So, if you see this transmission times, okay, this is my initial estimate. And then you have this initial estimate divided by the period of this message m_1. And similarly, the initial estimate divided by the period of the message with higher priority that is m_2, right, multiplied by the transmission times 5 and 8 you get 25, right.

Now you use this estimate again and then again you compute with 25 you get 33, you compute with 33 you get 38. And then you see that it is getting fixed at around 38, right. So, mind this, problem here sorry, if this is divided by pri_i and then you are taking the ceiling like I said earlier. So, you get this busy period and it tells me from the start time of the message critical instant when all messages are released onto the bus what is the time inside which if I am targeting message m_1 all its release instances are being executed, right.

(Refer Slide Time: 28:15)



Busy Period Computation

message	p_i(ms)	D_i(ms)	c_i(ms)	identifier
m_1	30	15	3	2
m_2	20	12	8	1 (highest)
m_1	40	30	12	3

Busy Period for m_2 in ms :

$B_{m_2} = 12$ $\qquad t^0_{m_2} = c_2 = 8$

$t^1_{m_2} = 12 + \lceil \frac{8}{20} \rceil 8 = 20$ $\qquad t^2_{m_2} = 12 + \lceil \frac{20}{20} \rceil 8 = 20$

So, that is how I compute for m_1 and similarly I can compute for m_2 its busy period, right. So again for m_2 as you can see, it has this blocking time again if its blocked by m_1, right, 12 and m_2 have only its own release instances to interfere, no nothing of higher priority. So, there are only one interfering term here, so it by a period 20 gives you 20 here and then again you put that estimate of 20 you get it.

So, its busy period is 20 only, right. And now if I am going to compute the busy period for m_3, it has a zero-blocking time, which is nice, because it does not have anybody else which is of lower priority but it is going to get interference from all the other tasks and also its own release instances. So, you start with the initial estimate of 12 and you compute this interference equation, you get to 25, 33, 38 and then that is it, right, the busy period.

So, we will stop here for the interest of time and we will continue in the next lecture from this point. Thank you.