Foundation of Cyber Physical Systems

Prof. Soumyajit Dey

Department of Computer Science and Engineering

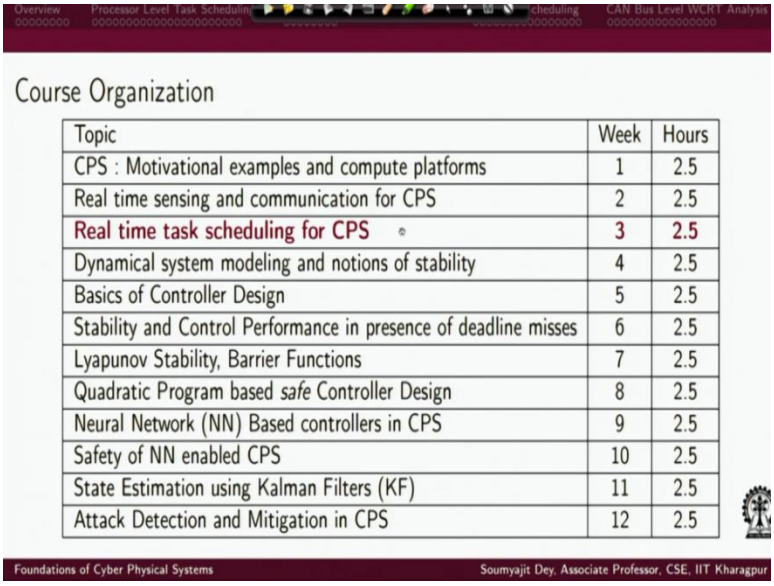
Indian Institute of Technology, Kharagpur

Lecture - 10

Real Time Task Scheduling for CPS

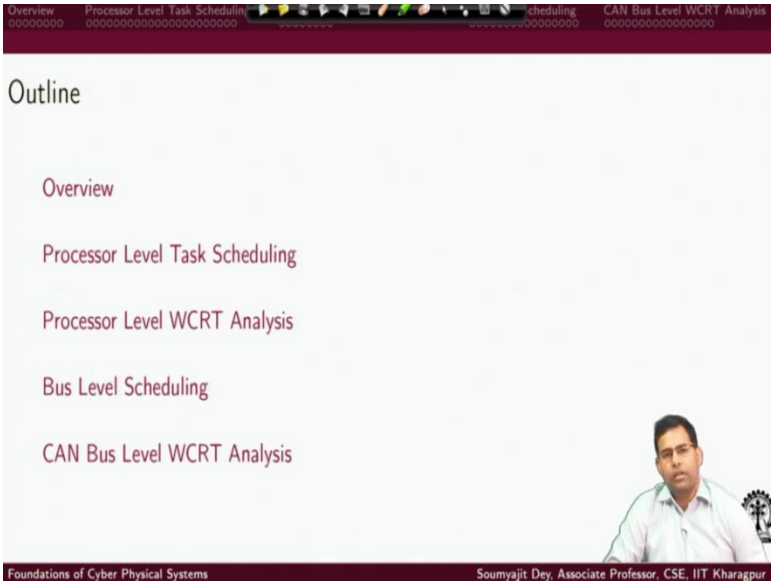
Welcome back to this lecture series on Foundations of Cyber Physical Systems. So, today we will be looking at our week three lectures.

(Refer Slide Time: 00:39)



Where the topic which we are going to start is about Real time task scheduling for Cyber Physical Systems.

(Refer Slide Time: 00:46)



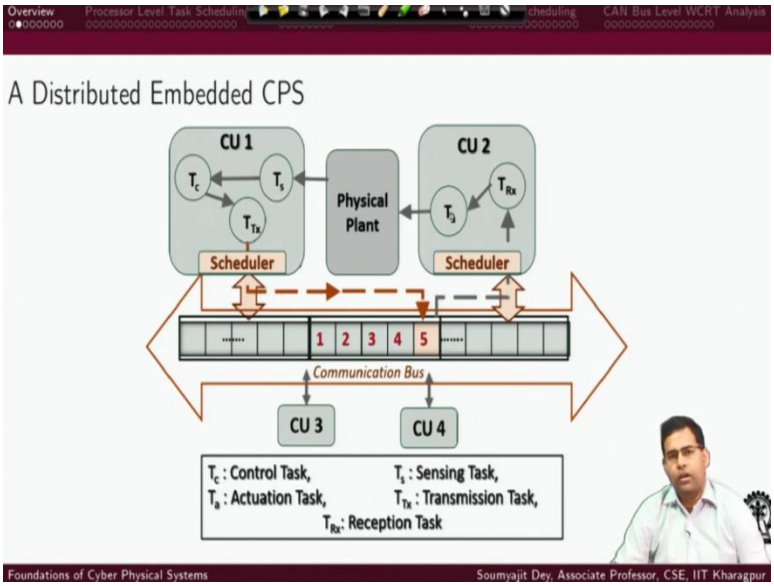
So, this is an overview what we are really going to cover are processor level task scheduling techniques. So, that basically means that you have a set of control tasks we and they may be shared with some other software tasks in a processor. And what are the mechanisms in which these tasks will be sharing the CPU and still they would be executing and providing some results inside some real time deadline.

Once that is done, we will go a bit further into timing analysis of such real time tasks. Essentially what we will be covering is techniques like how we can compute for such periodic real time tasks, what is their worst-case response time. And there are also related matrices, like some matrix, like worst case execution time and others. So, we will be introducing these terms and we will be learning how to calculate such response times.

Then we will also speak about that, the fact that well tasks are going to be scheduled in processors but such periodic tasks executing in processors are going to generate periodic messages. And these messages will be flowing along a bus through which, I mean, messages may be exchanged between multiple processors at other sensors and actuators. So, since this is a shared communication infrastructure how messages will be scheduled on the bus in real time things like this.

Of course, the primary application we are thinking of still is in the automotive domain here. But of course, these mechanisms are pretty much useful in many other real time embedded domains. And among these bus level scheduling protocols we will specifically talk about the CAN bus level timing analysis techniques like how to compute worst-case response time in CAN bus.

(Refer Slide Time: 02:44)



So first let us look at such a high-level architecture of a distributed embedded cyber physical system. So, typically in an automotive system you have this kind of an architecture. So, you have this CU 1 and CU 2, let us say they are representing two Compute Units or low power real time processing units, pretty much something like a microcontroller or a risk CPU like we discussed earlier in our introduction to basic computing architectures and stuff, if you remember from week one coverage.

So, let us say you have a Compute Unit one where you have multiple tasks that are executing. So, there is a physical plant from which some variables are getting sensed periodically through one software code which is executing periodically. Let us call it a sensor task. So, a piece of program when we make it execute periodically, we are calling it a task here, okay. Now this sensor task is generating some data on which the control task essentially a controller which is implemented as a software is going to act on periodically.

And it is going to generate some actuation command, this actuation command will be, I mean, it needs to be transmitted. Now it can be transmitted to a physical plant. So, here we have a candidate example where it is getting transmitted to another ECU which is actually going to, I mean kind of, physically it is interface is connected to the plant. So, if you see here the output of the control task needs to be compute, it needs to be communicated through this communication bus.

So, that is typically taken care of by a separate periodic task which we, let us label it as a transmission task. So, the job of this transmission task is it is going to wake up periodically and it is going to transmit the, it is just going to interface with the bus through some control, some, if it is a CAN bus then there is a CAN controller, things like this, which basically a physical chip which will understand the bus protocol.

And it will packetize the message, and as per the communication protocol implemented here it will drop those messages or drop those message packets which are going to contain the actual information that is the actuation information. So, in effect if I just look at CU 1, it is going to run multiple tasks periodically and they have dependencies. In general they may or may not have dependencies.

And there must be a real time scheduler here or this task can be interrupted through, these tasks can be executed using an interrupt mechanism, or there would be a very simple scheduler here which is going to run these stuffs periodically, because there is a one CPU. So, it is basically time shared among these tasks. Now when these tasks run, like we discussed this transmission task is going to drop packets here, on this bus.

Now this bus will have multiple transmission slots, right. For example, you can see we have drawn some transmission slots their slots marked as 1 2 3 4 5. Let us say this message goes to slot five, okay, and so the bus will have a delay after which its output is kind of stored in a buffer here. So, let us say compute unit two is also snooping on the bus and whenever it sees something interesting

here this message, it can just, the bus will be interfaced with this compute unit through another controller hardware, let us say it is a CAN bus you have a CAN controller.

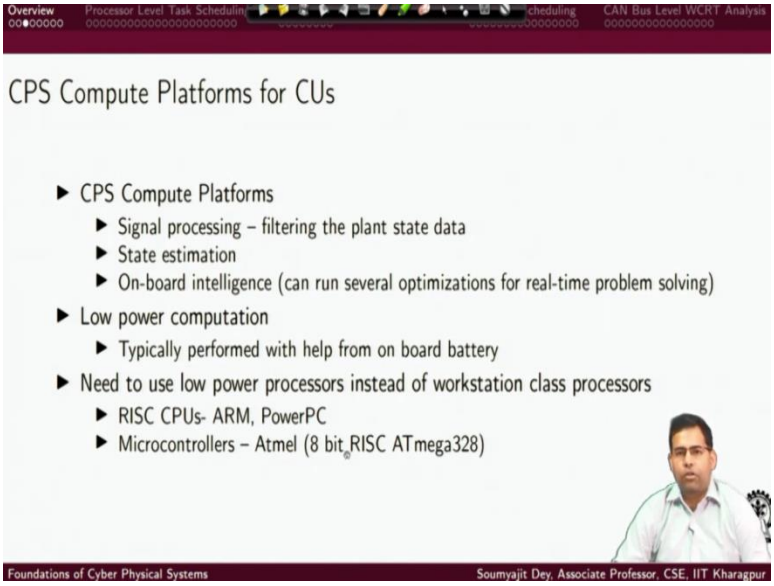
So, this kind of a message will be, which is of interest to this real time CPU it will be enqueued in the FIFO queue that is there. Now once this message is enqueued, here this message will eventually be read by another real time task here. Let us call it the reception task, which is also some piece of software which is running periodically here in this Compute Unit 2.

So, whenever reception task gets to execute on Compute Unit 2 his job is to sample that queue where messages of interest are getting dumped, okay. And it will, since this reception task is designed to understand this message protocol it can actually depacketize the data and it can get the raw data to some other task. In this case as we have drawn we have an actuation task Ta.

So, this will use that data to decide what will be the actuation value for this physical actuation interface, okay. So, this physical activation interface will get the suitable actuation command finally and that will engage the physical plant with the control input. So, that is what is happening here and so we have lot of periodic tasks which are executing periodically, they are exchanging messages, and multiple periodic tasks in the same compute unit are need, they need to be scheduled by a scheduler. Now the scheduler will be another piece of program which is basically executing a scheduling algorithm. The job of the algorithm would be to decide when which task gets access to the CPU and for how long the task, I mean, gets to execute on the CPU. So, these are the scheduling decisions which will be implemented in the scheduling layer.

So, we will be following these notations of Tc as Control Task, Ta as Actuation Task, Ts as Sensing Task and TTx as Transmission Task, and TRx as Reception Task.

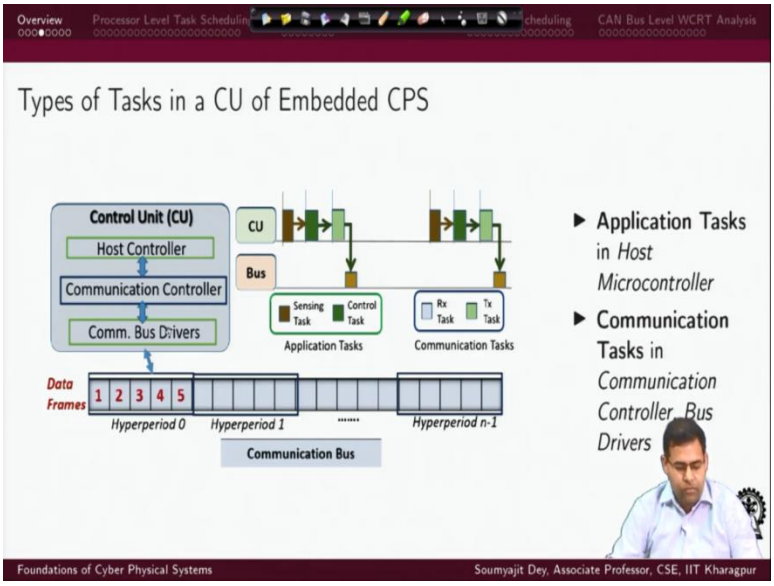
(Refer Slide Time: 08:50)



So, what can be these different tasks apart from them being some Control Task or Sensing Task for Actuation Task. Well, you can have some signal processing tasks which may be implementing some filters that denoise **(09:04)** the plant data. And there can be also task implementing some state estimation algorithm. There can also be some onboard intelligence like there can be some optimization methods running for doing some real time problem solving, okay.

Now if you remember you already told that these are going to be low power CPU, right. Because in many cases for specific embedded devices, they will be running on battery power, right. So, since they are going to run on battery power, they have to be of low power CPUs or microcontrollers, I mean, low power CPUs like ARM or the power PC family or microcontrollers like Atmel. Infineon has its own set of microcontrollers targeting the automotive domain and there are other vendors also.

(Refer Slide Time: 09:51)



So, if you look at this picture it is kind of a bit more detailed one. So let us say I have this kind of a Control Unit or Compute Unit, okay. So, in the vehicle context we will be calling the ECU or Electronic Control Units, okay. So, just like I was saying that this kind of an ECU will need an interface with the bus. Now there are there is a there is a layered architecture for that. So there would be some bus drivers the extreme lower level software through which they interface with the bus, okay.

And on the top of that there will be a communication controller and the host controller, okay. Now you have multiple application tasks running here, okay, and the communication tasks will be running in the communication controller which is sitting on the bus driver, I mean, communicating with the bus driver, okay. So, in the control unit you will have this kind of task like we already discussed, Sensing Task, Control Task, they would be running.

So, here we have a coloured representation we are trying to show these tasks on a timeline. So, we have this kind of timeline-based representation popularly known as gant charts **(11:08)**. So, you have this Controller Unit, where you have let us say the Sensing Task whose output, at some later point of time will be taken and being used by a Control Task whose output will again be used by the Transmission Task and all of them will execute again periodically.

That means, let us say this is the period of the Control Tasks, which means inside this period one instance of the control task must execute. And then there is the second instance. So, this is the Sensing Task of course. The sensing should happen once in the period. The Control Task should happen once in the period. The Transmission Task should also happen once in the period, like that. So, if you look at this analogy of a period let us call this period h.

Now if you just go back to this picture. So, each is the period after which the sensing is happening here, right. So, after every h period you have values from the physical plant being sensed and the rest of the things happening. So, just a small refinement here, in a typical such processor like we have also told earlier, let us say you are interfacing with a CAN kind of bus, okay. So, what we will require is another extra hardware like a CAN controller which is going to understand the interface of the bus, okay.

So, in general we call it a communication controller. And inside that CAN controller you will have suitable software programs which are going to run, right, which does this kind of interfacing. So, when you use the CAN controller you have to give you suitable commands, through which it is going to interface with the CAN bus and send messages with certain priorities. I mean, by the end of this lectures and also in our tutorials we will be showing you some examples of such CAN-based communication how such communications can be performed through some simple programs running on Atmel **(13:21)** or some other microcontrollers.

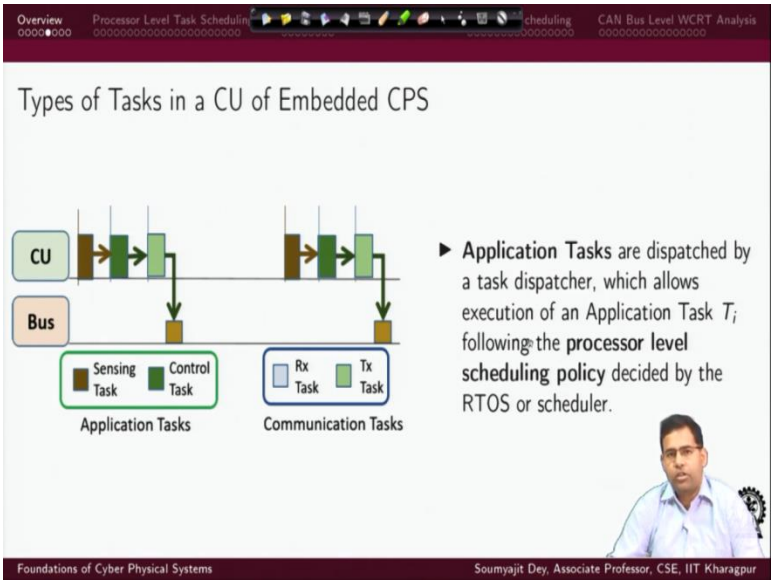
So, in modern automotives, what happen is, at the physical level you will have these ECUs and you will have such the ECUs being interfaced with the bus through this kind of communication controllers, and so that, that is the hardware part. And inside the ECU you will have a set of software layers at the lowest level you will have drivers, right. So, you will need drivers for all the ECUs peripheral interfaces.

For example, you will need the bus drivers which are going to with the lowest level software layer for interfacing with the bus. And on the top of that you will have the communication layer. And the top of that you will have some host operating system which may be an RTOS **(14:09)**, or some

other very lightweight scheduler which is kind of tasked with running the real time task on the ECU. So, you have this kind of layered software architecture in. I mean, there is a name for that the AUTOSAR architecture, AUTOSAR adaptive, AUTOSAR. And they have got their own specifications and modern standards agreed on by automotive companies. So, just like I said that modern automotive software engineering is quite advanced and sophisticated.

They have this kind of a layered architecture and on the hardware side they have this ECU's interfacing with the bus through specific physical layer controller.

(Refer Slide Time: 14:55)



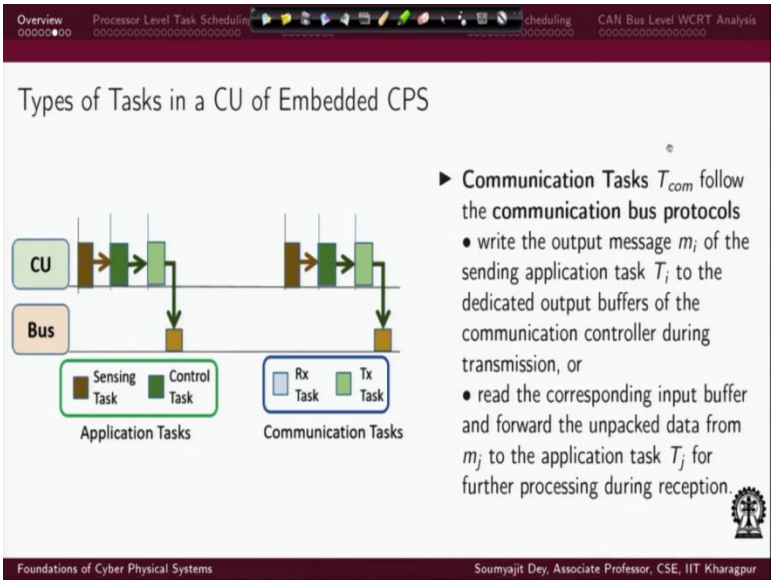
So, coming back to our discussions on scheduling here. So, like we have been discussing that we, let us talk about the tasks which are running on the Compute Unit or the Control Unit. We have the Sensing Task and Control Task and the communication is being handled by the Transmission Tasks and the Reception Tasks. So, whenever you have this sequence of sensing control and transmission tasks getting executed in the Control Unit, you will get some message being loaded on the bus.

For example, here you see that. So, if this is your period, inside this period you have one instance of sense, control and transmission happening. This is going to be later followed off by actuation

which may be done through another ECU here, right. So, this entire thing should happen inside this sampling period h, which is the sensor sampling period. After every h, the plant will be, from the plant side some data values will be sensed.

And similarly at every h interval, the plant side we will expect an actuation command update. So, the Application Tasks as you see they can be dispatched by a task dispatcher which will allow the execution of an Application Task following the processor level scheduling policy. So, like we have been saying that there can be a vanilla simple scheduler or there can be a full-fledged real time operating system which will implement the scheduling algorithm following which the tasks will be executed in the Control Unit.

(Refer Slide Time: 16:43)



Now this communication tasks they need to understand the bus protocol, okay. So, their primary task is to write the output message, let us say some message mi, of the Application Task to the dedicated output buffer of the communication controller. So, like we have been saying that there is a layered architecture here, right. So, the transmission task which is executing at this communication layer, right, through the corresponding software interface, it will communicate with the underlying communication controller hardware.

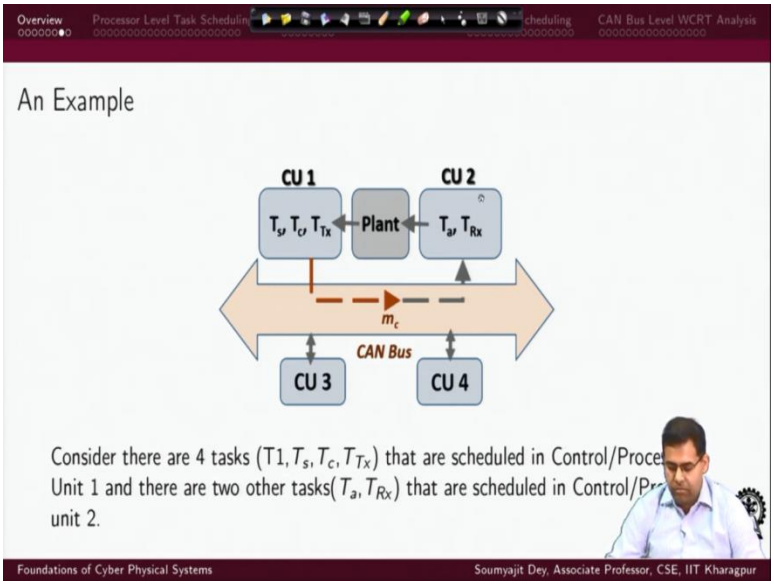
So, that is containing this kind of a queue, right, a message queue and this tasks job is to write this message to this buffer or the queue of this communication controller. And the communication controller's hardware is going to take care of whenever there is a, whenever its cycle time will come it will see if there is a message in the queue and accordingly it will just put the message in a corresponding slot in the bus.

On the reception side the job of this task, that is the Reception Task, will be to read a corresponding input buffer. So, you can understand that every communication controller, I mean, both of the hardware and on the top of it the layer software level, it will implement the input and output buffer. So, that means, I mean, the input buffer will simply be same sampling messages that are received through the bus and the output buffer is just storing messages which need to be transmitted through the bus, right.

So, it is going to the transmission the Reception Task is going to read the corresponding input buffer and it is going to forward the data from to the next Application Task whichever wants to read it in that compute unit. Now the primary job of this tasks and the software layer is to packetize and depacketize the data as per the communication protocol that is followed. Because whatever is your bus it will have some format for sending messages, right.

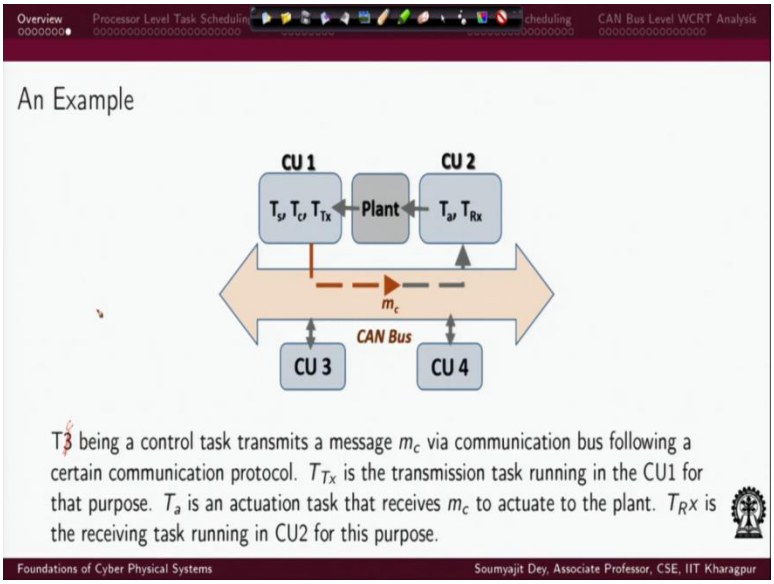
So, accordingly expert that bus protocol the message has to be prepared and similarly when a message is received in that packet, in that protocol, following that protocol, from that message the payload or the data part needs to be unpacked.

(Refer Slide Time: 19:05)



So, let us take a small example. Suppose you have four tasks. You have T1, some other task. And then you have a Sensor Task and the Controller Task and the Transmission Task. So, this sensor controller are with respect to some control loop, okay. And they are scheduled in Control Unit or processing unit one here. And let us say the Reception Task and the Actuation Task corresponding to the same Control Unit they are scheduled in some other control some other CU here, okay. So, as we have in this figure.

(Refer Slide Time: 19:42)

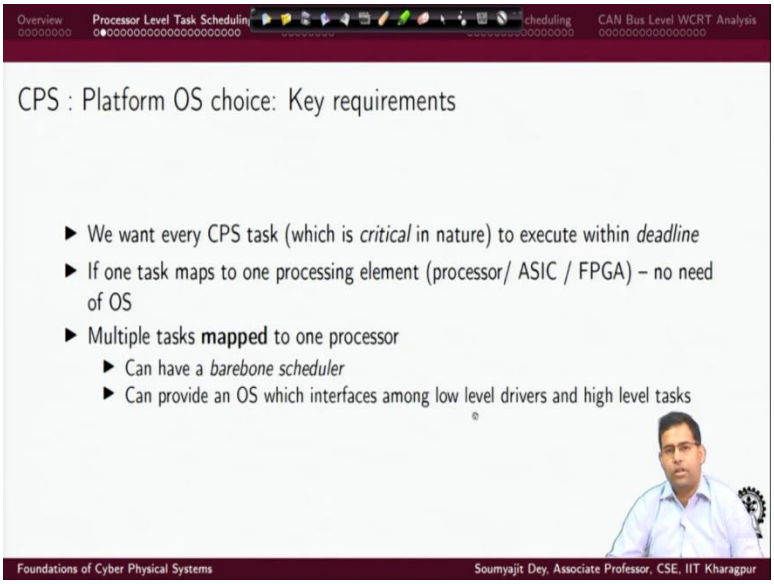


So, this control task is going to transmit some message via the communication bus following the whatever is the protocol in the bus in this case it is the CAN protocol. And like we said that Tx is

denoting the Transmission Task it is also running on CU 1, right. And Ta is the actuation task which is supposed to receive the message and the reception part is actually done by this TRx. So one thing here this is Tc, not T3. Please note that.

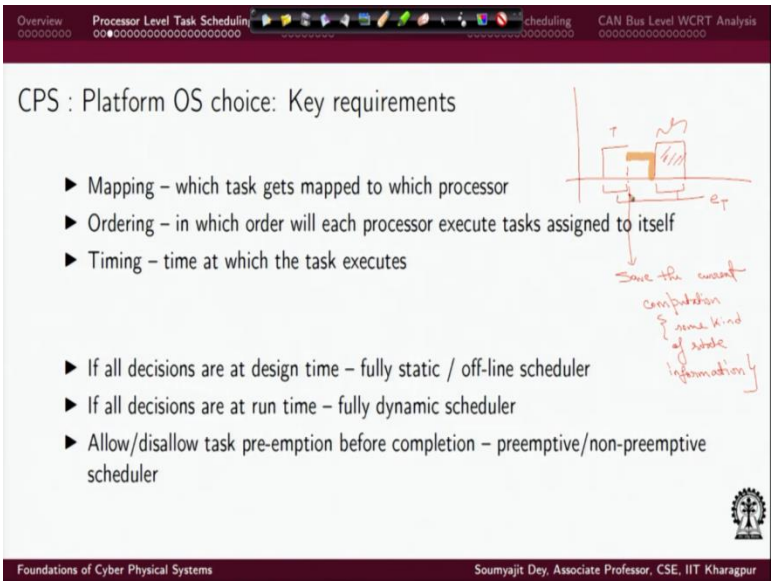
So, that is how high-level architecture of automotive system may look like. Of course, in a very complex system the bus will contain more such interfaces and more number of processors which will be connected to it. So, well, we understand that how the system is there physically in the architecture. But then what we need to focus on is how multiple tasks here are going to be scheduled together and how multiple messages here with different priorities are going to be sent.

(Refer Slide Time: 20:57)



So, we will start with the processor level task scheduling. Now let us first understand what are my requirements there. So, what we really want is that every CPS task which is critical in nature, it should execute within a deadline. So, if one task maps to one processing element, then I do not need any kind of scheduler, right, because there is only one task and one processing element. But whenever I have multiple tasks which are being mapped to one processor, then I have this requirement of scheduling algorithm to decide which task should run, when on the processor. Now it can either be a very simple barebone scheduler, or there can be a full-fledged operating system which will interface along with low level drivers, and it will also have a scheduler which is tough to run, I mean, decide when these other high-level tasks were going to run.

(Refer Slide Time: 21:51)



So, the primary things that need to be done in this case is that well, the first decision that is to be made is which task will get mapped to which processor. Suppose you have a multi-processor environment, or you have a processor with multiple cores. Then the primary job as a designer is that if you have multiple tasks you have to decide which task gets mapped to which processor, right. Because if tasks are mapped to different processors, you also have to resolve this issue of how the tasks are going to communicate through the bus.

Then the next thing is you have to decide, that well, for each processor which is currently going to run multiple tasks in which order the task needs to be executed. So, that is the scheduling step in the uniprocessor scenario. And when I consider the previous problem and this problem together, they represent what we call as scheduling in a multi-processor scenario, okay. And then that important thing what do we really mean by scheduling is a very simple idea, that in a single processor if you have multiple tasks scheduling simply means, what are the time stamps at which a particular task gets to execute on the CPU. So, deciding these time stamps and giving them a periodic behaviour is finally what we call about scheduling. Now the question is when will these decisions about scheduling be taken? Are they be going to be taken at the design time?

If so then what we have is offline scheduler. So, on board there is no practical scheduler. You just have a schedule table and that scheduled table may have been computed offline using some optimization approach. And it just tells that well what are the exact times when which, when certain tasks are going to start. Or it may say that well this task is going to run when these previous tasks have just ended, immediately they can just start running.

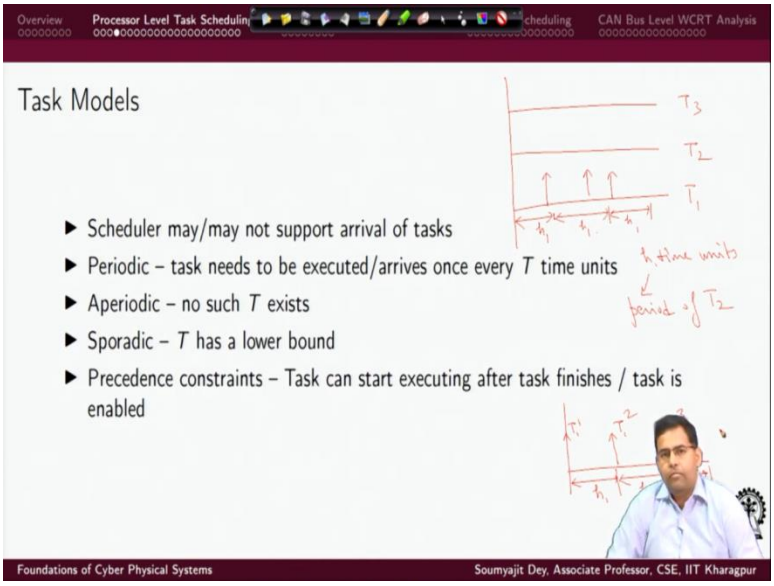
So, in some such format the, such an offline fully static scheduler will be specifying the start times of the task. The other option is to have all the decisions taken at the run time that is what we call as a fully dynamic scheduler. That means you have the set of tasks and essentially depending on runtime scenario, runtime situations, like for each task what is the execution time that is remaining?

What is the deadline? How much time is left till the deadline? Based on various such possible scenarios the system is deciding which task to execute. So, that is what we call as a fully dynamic scheduler. Now there is another important scheduling terminology which is pre-emption. Whether or not we want to allow a task to be evicted from its current execution in between. That means, let us say you start a task T.

So, suppose you started executing a task T here. And then at this point you decided to just remove it from the **(24:50)** system. And you bring some other task here. And when these other task ends, then you decide to start T again. And then, this gives you the total execution time of T. So, the point is if I am going to preempt at this point, whatever computation of T has been done, I need to save the computation.

And when I am going to restart this task, I am going to just start as if, I mean, from this state of execution the rest of the computations which are needed to be done by this task instance, they will be done here, okay. So, if this is allowed, we call it as a preemptive scheduling policy and otherwise we call this a non-preemptive scheduling policy.

(Refer Slide Time: 26:17)



So, in general tasks, I mean, in real time system tasks, I mean, execution of tasks can be pertaining to such different models. So, we will just familiarize ourselves with some of these terminologies. For example, such tasks I mean the schedulers that we are talking about they may or may not support arrival of tasks. Now what does this mean supporting arrival of tasks. So, let us understand. Suppose, you have a set of tasks T1, T2, T3 and maybe some more they are given.

And you are asking the scheduler is that well you decide, what should be the start time for T1, what should be the start time for T2 etc, etc. So, the scheduler will run and based on certain characteristics of these tasks, it will decide when to start whom, right. Or the other model can be that well these are tasks which can arrive on the system. So let us say you have the CPU here. And right now you have task instance T1, and task instance T2, and task instance T3, which are pending.

So, you are deciding to run T1, okay. T2 and T3 are pending. At this point there is a new task instance of T1 that came. So, we just call this as the first task instance of T1. This is the second structure instance of T1. So, then your set of tasks to be scheduled has to be updated. And similarly in future again let us say T2 is also repeating with some period or T2 there is no fixed period. But T2 can come sporadically or T2 can come randomly. Whatever may be the case, more task instances keep on coming, right.

So, if your scheduler is able to handle this scenario. Then we call it that well this supports arrival of tasks and otherwise it does not. Hope this is clear. Now if I assume that all the tasks which I am going to schedule, they are all periodic, that means they are going to be executed once every T time units where T is the period of the task. So, for example, let us say I am supposed to schedule tasks T1, T2 and T3. And T1 has instances arriving.

So, this is your period h1. And let us say from here again the period starts, h2, h3. So, if that is so, that I need to execute an instance of T1 once. These are all same. This is the period of T1. that means, I need to execute an instance of T1 once every h1 time units, okay. Then h1 time units, this is the period of T1. So, if I characterize the period by an arrival time, so it basically means this, that well, I have an instance of each one. Let us say which is coming right here, then coming right here, then coming right here. Or rather I would say h1 has one instance, sorry, T1 has one instance. The first instance being released right here, T1 as another second instance being released right here, T1 as the third instance being released right here, like this. So, these are the release instances which are separated exactly by h1. And in case of a real time periodic system or rather a hard real time periodic system the requirement to the scheduler is, that, I need to execute an instance of T1 once every h1 period in the system.

So, they are going to be released at exact h1 in difference, but it does not mean they need to be executed at h1 difference. It means that inside every h1 interval, one instance of T1 must be executing. I get some result from T1's instance, and updates, once every h1 time units. So, that is the interpretation with arrival time and that is the interpretation with scheduling. I hope this is clear.

So, that is how periodic tasks are modelled and periodic tasks are supposed to be executed, fully periodic tasks. Now you may also have aperiodic tasks in a system, which means that well there are no such characterization of periods. The tasks can come anytime there is no timing model of the tasks arrival, okay. Then we would simply say that it is aperiodic. And there is another alternate thing called as sporadic task model which means that well it is not periodic but it is something like a quasi-periodic thing.

So, before going to that let us complete this picture. So, just like we have T1, we can have some serious requirements of T2 also, that will there is a period h2, and once somewhere inside this period h2, T2 must have an instance executed. Once somewhere inside this h2 interval, the next stage two interval T2 must have an instance executed etc, etc. And similarly, for T3. So, that is how that is how the scheduling requirements are.

And the scheduler S needs to satisfy the requirement by suitably choosing when to execute each of these tasks in a single CPU so that all these given requirements are satisfied together, okay. Now coming back to this situation of sporadic tasks. When I say that well T has a lower bound. So, let us understand what T as a lower bound means. It means that well, suppose an instance of the task T comes. So, any second instance of the same task must come after a separation Tmean.

So, this is the lower bound. I am not saying that the second instance must come after this interval. I am saying that the second instance must be separated by minimum this value. So, that is how typically a sporadic task model tells that what is the timing constraint on the task. Now apart from this there can be other situations like real time tasks can have precedence constants. For example, it may so happen that will task T1 and task T2, once they execute and produce some results, they, those results need to be available then only then a task T3 can execute.

So, these are known as precedence constraints. They can be modelled using or directed acyclic graph. So, that means that this T3, it can start executing only after the tasks on which it is dependent. That means its producers are task has completed their execution and their output is available. So, with this we will be finishing this lecture, with this discussion on task model. See you soon. Thank you.