

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

**Lecture - 15**  
**Real Time Task Scheduling for CPS (Continued)**

Welcome back to this lecture series on foundations of CPS. So, in the last lecture we have been talking about this response time analysis of multiple tasks which are scheduled in a uniprocessor.

**(Refer Slide Time: 00:38)**

Processor level Response Time Analysis Example

- ▶ Since  $R_3 = 38 \geq D_3 = 30$ , the deadline is not met for the control task.
- ▶ If we increase the period and deadline of the control task to 50ms then it will meet the deadline.
- ▶ If the priority is fixed (it might change based on the deadline though), change in deadline or period does not affect the response time of the control task i.e.  $R_3 = 38 \geq 50$ .
- ▶ Notice that the utilization is also  $> 1$  which denotes the task set will be unschedulable.
- ▶ In that case we need to check will the other tasks remain schedulable?
  - ▶ Yes. Since the control task has the lowest priority, it will not affect the response times of the other tasks.
  - ▶ But if we change the parameters of other tasks, that might affect the response time of the control task.

**(Refer Slide Time: 00:42)**

### Processor level Response Time Analysis Example

message	$p_i$ (ms)	$D_i$ (ms)	$e_i$ (ms)	Type	Priority
$T_1$	30	15	5	Security Task	2
$T_2$	20	12	8	Sensing Task	1 (highest)
$T_3$	30	30	12	Control Task	3

Response time for  $T_3$

So, from our previous analysis, if we just recall this was our task set, there was a control task, somehow, we gave it a lower priority and we saw that well this response time is violating the deadline, so then the question comes that well what to do in this case? One issue can be that well if I can increase the deadline and period of the task to let us say 50 millisecond that means this task instance of the control task starts coming at a period of 50 instead of 30.

Then of course the response time is well within the period, so that is not a problem. Question is can I decrease the response time? If I have the priority fixed the response time will not change. Because the task being of lower priority than other tasks will continue to have the interference that we calculated and that interference will always keep on creating the same response time for us, right. So, unless the priority is changed this response time will not change, okay.

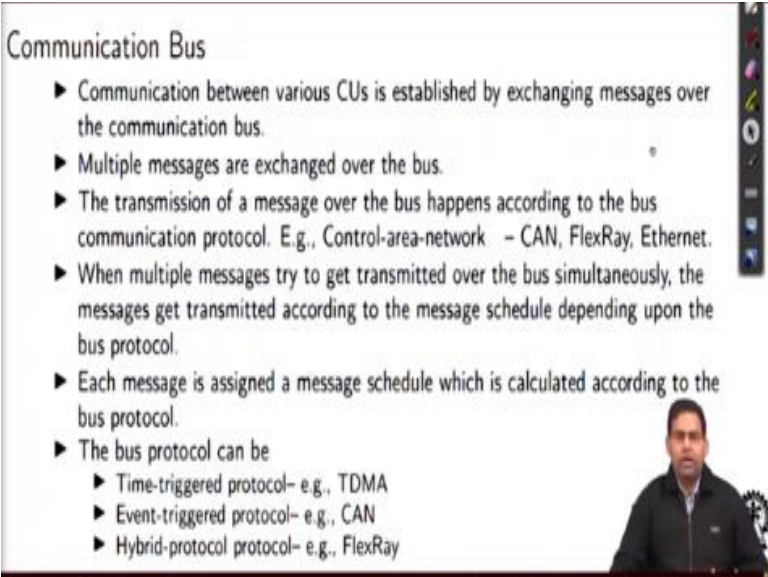
But of course, if I increase this to 50, then this should be less than then of course 38 being less than 50 it will satisfy the deadline requirement here, okay. Now of course if you see that in the previous setting if we calculate the utilization, that also makes it unschedulable. It will be the utilization will be greater than one. So, unless we do something about changing this period from 30 to 50 or something larger than 30, this utilization will also not go down. So that is one issue, okay.

So, these are the two important points we need to remember, that since the task has the lowest priority, it does not affect the response time value, okay. And so, one option can be to increase the period, so that the utilization goes to be less than one, and also 38 less than 50 will satisfy the deadline requirement. And unless that is not possible, for example, if I am controlling less frequently because if I increase the period, then I am executing the control less frequently that may not be a good thing to do for the system, okay.

So, if this choice of increasing the period and deadline is not okay for the system, then we have to look at and change the parameters of other tasks, and we have to see that well if that is admissible. So, if we change the parameters of other tasks, that might affect the response time of the control task. So, that is what we have about our example and stuff on processor level response time analysis.

And let us just move on to the next topic here inside this processor level and bus level scheduling. So, we have completed kind of this processor level scheduling algorithms and processor level response time analysis. The next thing we have to understand is bus level scheduling.

**(Refer Slide Time: 03:48)**



### Communication Bus

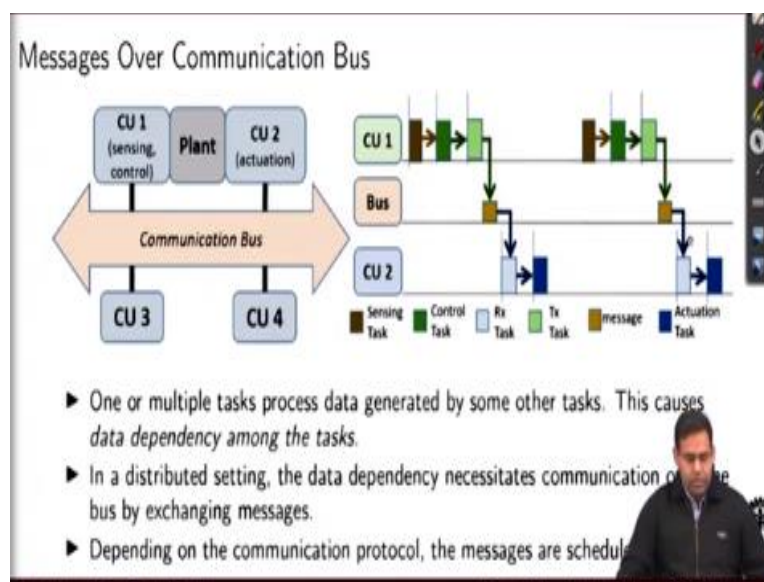
- ▶ Communication between various CUs is established by exchanging messages over the communication bus.
- ▶ Multiple messages are exchanged over the bus.
- ▶ The transmission of a message over the bus happens according to the bus communication protocol. E.g., Control-area-network – CAN, FlexRay, Ethernet.
- ▶ When multiple messages try to get transmitted over the bus simultaneously, the messages get transmitted according to the message schedule depending upon the bus protocol.
- ▶ Each message is assigned a message schedule which is calculated according to the bus protocol.
- ▶ The bus protocol can be
  - ▶ Time-triggered protocol- e.g., TDMA
  - ▶ Event-triggered protocol- e.g., CAN
  - ▶ Hybrid-protocol protocol- e.g., FlexRay

Because if we remember our earlier discussions, we are considering an automotive or similar kind of CPU system where there is a real time bus communication and it is connecting multiple processors, each processor is running multiple tasks. So, once those tasks are scheduled in the processors, they are going to generate periodic messages and those messages now need to be scheduled on the bus, right.

So, this transmission of messages over the bus will happen following some specific real time communication protocol. The popular one is CAN and there is also FlexRay and we also know it as Automotive Ethernet and other standards coming up. So, we have this same scheduling issue now occurring over the bus because multiple messages may try to get transmitted over the bus simultaneously and then the messages will need to be scheduled.

The bus protocol should select which message to communicate to the target system at exact what start times. So, this message schedule will be calculated now according to the bus protocol and bus protocols can be of several kinds. It can be a time trigger protocol, for example a time division multiple access or TDMA. It can also be an event triggered protocol like CAN or it can be a hybrid protocol like FlexRay, which has both times triggered as well as event triggered slots.

**(Refer Slide Time: 05:13)**



So, if I look at these messages over communication bus, let us take an example. So, you have a compute unit one here, which is running let us say sensing task and a control task and it is getting some messages from the plant. And it is sending the control command as computed by the control task over the communication bus, and that is going to some control unit which is going to control the actuation by using this control command that is being calculated.

So, one or multiple tasks process this data that are generated by some other tasks and this would create the data dependency or the precedence constraints among such tasks. So, for example, in compute unit one you can have a dependency like the sensing task will execute and then it will pass over some sensed information from the plant to the control task. It will compute the control command and then the control command would be sent to somebody else, some transmission task.

And this transmission task, when this transmission task is executed, I mean, it will submit the message to the output queue of this compute unit. And through this transceiver chip, this message will eventually be communicated through the bus as a packet and it will be received by a reception task, okay in compute unit 2. Because when the bus is transmitting the message, the message will be submitted to some incoming message queue of compute unit 2, right.

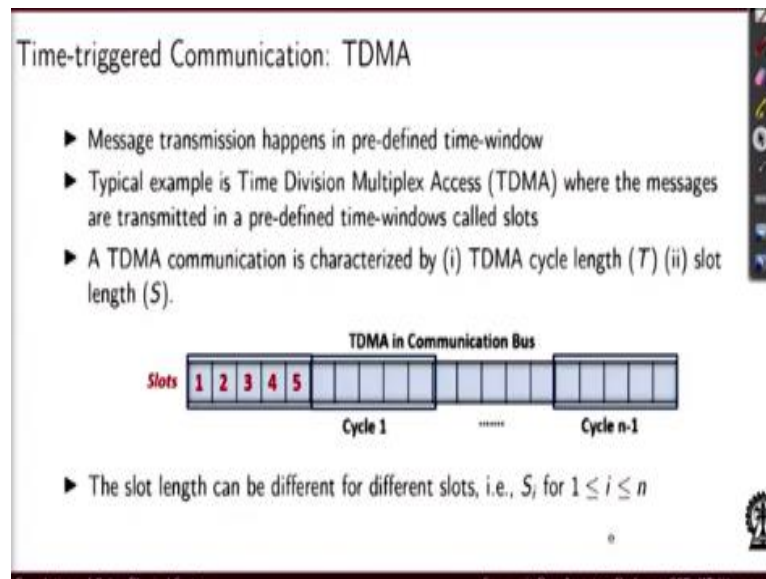
And when in compute unit 2, this reception task or  $R_X$  task as you can see here, we based on the colour code we are following here. When this guy is executing, it will collect the message and it will pass over the message to the actuation task and the actuation task will process this message and do the actual actuation command computation when this guy executes. So, if you see this colour code that we are following here.

So, this is the sensing task then the control task and then the transmission task here, right. The green one, the light green one, the deep green one being the control task. And here we have the message over the bus, right and then the message is received in compute unit 2 by the reception

task right there the light blue one and the deep blue one is the actuation task. And this thing is going to happen periodically here once and here again like that.

So, the bus scheduling part has to deal with deciding when to transmit this message at what periodicity and at what offset and it has to do this for many other control loops together, okay. So, in a distributed setting this data dependency will necessitate communication over this bus and it and there will be multiple messages that will be getting exchanged here. And depending on the communication protocol it will be decided that when messages corresponding to which control loop are getting scheduled on the bus.

**(Refer Slide Time: 08:13)**



So, we will start with a simple protocol that is TDMA. So, that I mean this is Time Division Multiple access that means every communication of message is kind of time slotted. So, there is a TDMA cycle, inside a TDMA cycle there are multiple transmission slots. What you have to do is we have to map each message to some specific slot and that slot will be kind of booked by this message and in every TDMA cycle in that slot that message is expected to be sent.

So, that is how it works. And as you can see that we call this time triggered communication because the slots are kind of defined by a clock signal, right and based on the timing of the slot the message

has to be same, right. So, it depends on in which slot you have mapped the message. So, messages are transmitted in a predefined time window and these are known as slots, right. So, the total TDMA cycle length will be the collection of these multiple slots that is the sum of the slot lengths, okay.

So, the slot lengths can be different for different slots in general for a simplistic assumption we can have all the slots of same length, right. So, that means each slot is going to take the same amount of time inside one TDMA cycle. But overall, what we need to compute is a mapping of messages to slots and then the TDMA cycle when it gets repeated the message will be going as per that message to slot mapping that we decide.

**(Refer Slide Time: 09:52)**

**TDMA: Timing Properties**

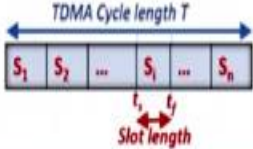
- ▶ The slot length and the TDMA cycle length is related by:

$$T = \sum_{i=1}^n S_i$$

- ▶ If the slot length is equal and there are n number of slots in the TDMA cycle:

$$S = \frac{T}{n}$$

- ▶ If a message  $m_i$  is scheduled to slot  $S_i$ , the start time  $t_s$  and finish time  $t_f$  of message transmission are given by:  $t_f = t_s + S_i$



So, the slot length and the TDMA cycle length is defined like this. Suppose there you are dividing that TDMA cycle into n slots and there are slot lengths of size  $S_i$  for ith slot. So, that summation is the TDMA cycle length,

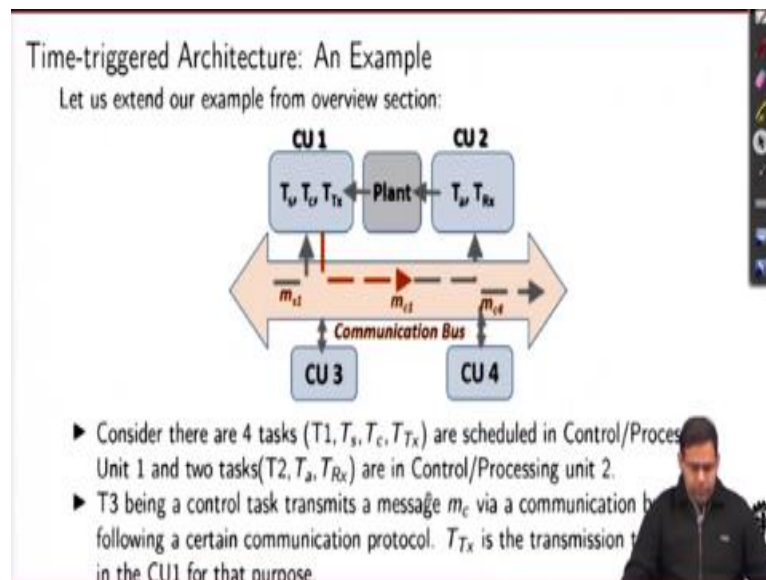
$$T = \sum_{i=1}^n S_i$$

Assuming that the slots are of equal size. Then of course if you are doing such a simple, that I mean slot length calculation it is just  $T/n$ , right. Now if a message  $m_i$  is going to be scheduled to a

slot  $S_i$ , then the start time and finish time  $t_s$  and  $t_f$  of the message can be computed just like this, right.

I mean, the start time is there, so there is a start time at which the message starts the transmission and then the slot length is  $S_i$  that supposed to the time when the transmission of this message will finish, right. So, we will have always  $t_f = t_s + S_i$ , something like this.

(Refer Slide Time: 10:51)



Simple example of a time triggered architecture here. You have multiple compute units. So, you have one task  $T_s$  sensing task, control task  $T_c$  and then the transmission task. And then let us say and I mean so, we are getting the sensed message from the plant. So, that is why we have this arrow and then this message that is the control command is moved over through this communication bus to compute unit two where it is being received by  $T_{Rx}$ .

And then that is used by  $T_a$  to decide some actual actuation value and that is passed over to the plant. So, that is how this thing is working here and we are just trying to show pictorially that there can be multiple messages on the bus and this is the message of our interest that is  $m_{c1}$  for compute unit 1 the control message that is coming. So, we have a four-task system here, okay. So, we have

let us say some task  $T_1$  and then we have the sensing task, the control task and the transmission task.

And they are run together in control unit or processing unit one. And let us say in the other processing unit we have the tasks of our interest that is  $T_a$  and  $T_{Rx}$  and along with that there is some other task  $T_2$ . It may be a security task that means a task which is looking for whether there is an execute issue there may be a monitoring task, whether the variables are maintaining some correlation etc.

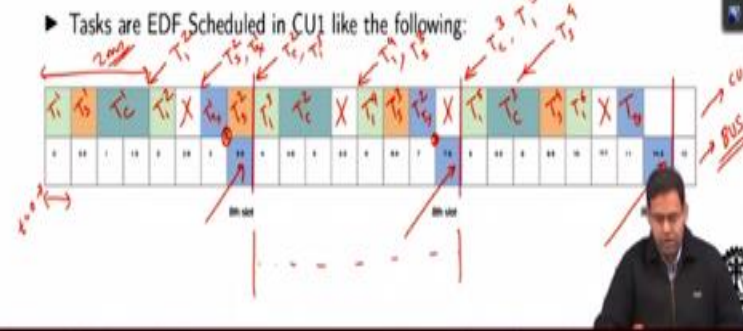
So, we will talk about such security and monitoring tasks later on in detail. Now let us assume that these are tasks which are also running in these shared processors and they are creating interferences for the tasks of interest to us, which are these ones, okay. So, this  $T_3$ , I mean we can call this  $T_3$  or  $T_c$  like we discussed. So, this is a control task and this is going to transmit a message  $m_c$  over a communication bus following a certain communication protocol.

And like we discussed earlier that the corresponding transmission will be handled by this task which is also going to be periodic and they are both executing in the control unit one.

**(Refer Slide Time: 13:14)**

## Time-triggered Architecture: An Example

Task	$p_i$ (ms)	$o_i$ (ms)	$e_i$ (ms)	Type	Message
$T_1$	2	0	0.5	Security Task	-
$T_s$	3	0	0.5	Sensing Task	-
$T_c$	4	0	1	Control Task	$m_1$
$T_{Tx}$	4	3	0.5	Transmission Task	-



So, for compute unit one we have let us say this kind of a task model. So,  $T_1$  is the extra task let us there be a security monitoring task, it has a period two there is no offset that means it arrives at  $t = 0$ , the execution time is 0.5 all of these are in millisecond. And you have the sensing task and you have the control task, okay. So, the sensing task is with periodicity three, execution time 0.5. Control task has a much larger periodicity of 40, okay and it has this execution time of one.

And then we have this transmission task periodicity of 4 and it has an offset of 3, okay. So, one small error here. So, this control task is actually of 4 and not 40 because of course you are going to execute this and then the transmission tasks together, okay. So, as you can see from this figure what you have is these three tasks the  $T_1$ ,  $T_s$  and  $T_c$  they have arrived to the system. But we are going to execute them in this order here.

So, what we are trying to show that well what is going to be a possible mapping of the messages and the and the tasks in the processor that the schedule of the messages on the bus and the schedule of the tasks on the processor that is what we are going to show here, okay. So, first let us try and create an EDF like schedule of the tasks in the compute unit one. So, here we are trying to have a pictorial representation.

So, this part here is where we show the CU1 task execution timings, okay. This is  $t = 0$  and here at the lower part of the diagram we are going to show the bus and the timing diagram of the bus with the messages belonging to some slots. So, at the bus level we have done this kind of partitioning of slots of size 0.5 millisecond here, okay. Now let us say this three messages  $T_1$  this task  $T_1$ ,  $T_s$  and  $T_c$  they just start executing together, okay.

So, we can have  $T_1$  here following the EDF schedule and then we can have  $T_s$ , okay. It has a smaller deadline with respect to  $T_c$  the control task, right. So, we can have  $T_s$  here and then the control task is remaining, right. So, let the control task come over here, right. And at the end of this once up to this point we see that the execution times are 0.5, 0.5 and here the execution time is 1, right. So, at this point we have covered about 2 milliseconds of time.

So,  $T_1$ 's next instance will be arriving soon and at this point, right.  $T_1^2$  has come. So,  $T_1^2$  will get in here these are all  $T_1^1$  like that, right. And then at this point you do not have anything to execute, because let us identify when the other tasks are coming. So, if you see at  $T_s$ ,  $T_s$ 's second instance is going to come at a periodicity of 3, right. So, this is length 2, this is 2.5, so this is 3. So, let us just write them down first.

So,  $T_s^2$  is supposed to come here, right and then if you try to think of  $T_c^2$  and the second instance, right, so that should be arriving somewhere here. And at the same time  $T_1^3$  is going to come, right. So, let us just write down these instances first. And you will also have the sensing task now repeating. Now the sensing task comes at instances of 3, right. So, the sensing task also comes here once  $T_s$ , it is third instance, right.

And now if I am going to compute the schedule of this task. So, we have already done up to this, right. So, next after this point what is going to execute let us try to figure out. So, at this point you

have this sensing task which has come but also let us not forget the transmission task, right. So, the transmission task will have an offset of three, right. So, its first instance is here and that is quite because the transmission is following the control command, right.

So, you have this transmission task coming up right here, right. So, you will have this transmission task executing right here. Let us see the schedule now. So, the question is why should it execute right here? Its deadline is more urgent. Because its deadline is going to expire inside these 4 timelines, the period, right. So, I mean, we will have it here the transmission task and then we can put the sensing task here. If we continue like this, then we can soon see that this will be  $T_1$  again.

$T_1$ 's third instance and then we will have control task's second instance, right. Then again do we have anything left? Because up till now whatever has come, we have executed them.  $T_1$ 's first instance came here and we executed it, right. And sorry I did not show the arrival time of  $T_1^3$  is this. And we have executed it  $T_c^2$ , we have executed sensing task is also executed right here. So, at this point inside this box you again do not have anything to execute just like this. So, these are blank.

So, again we have  $T_1$  coming. So,  $T_1^4$  executes here, and then  $T_s$  third instance executes here, right. And I did not show the arrival of the next transmission task but it will be inside this period. So, this transmission task  $T_{Tx}$  second instance will execute here. And then again here you do not have anything, right. But this is again the point where you have the send this  $T_1$ , sorry, so this is  $T_1$ 's fifth instance that is going to come here, right.

And so,  $T_s$ 's third instance came here and then it is supposed to come again somewhere here. That is where  $T_s$ 's this third instance. And then this is the fourth instance that would come. And if we look at  $T_1$  so  $T_1$  is repeating with a period of 2, so  $T_1$ 's fifth instance would come here at eight. So,

because at two at 4, and then 6, and then 8 and that is how  $T_1$  is coming, right. So, at this point, at the end of the eighth slot you have the transmission done.

Now this is again a blank slot nothing to execute. You will have  $T_1$ 's fifth instance executing here. Then the control task's third instance executing here, then you have this sensing task fourth instance executing here. It will be followed by  $T_1$ 's sixth instance and so on so forth. And then there will be a blank slot, and then again you will have the transmission. Now observe that the transmission is coming one later here again, because before that it is not scheduled to execute.

So, this is kind of how the messages are going to be EDF scheduled in compute unit 1. And now the question comes that well how am I going to send the messages over the bus, right. So, if this executes here that means right now at this point the message is in the output buffer of CU 1. So, I can kind of schedule this message right here for sending, similarly I can schedule the messages again available in this buffer.

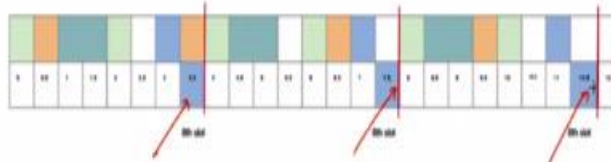
So, I can schedule it for sending right here, and similarly right here. So, this is the slot using which so basically, I can send it, right. Now then if you see a possible slot mapping. So, there are eight possible slots and the eighth slot can be reserved for sending this message on the bus for this specific controller, right. So, that is how it will work here.

**(Refer Slide Time: 23:51)**

## Time-triggered Architecture: An Example

Task	$p_i$ (ms)	$o_i$ (ms)	$e_i$ (ms)	Type	Message
$T_1$	2	0	0.5	Security Task	-
$T_s$	3	0	0.5	Sensing Task	-
$T_c$	40	0	1	Control Task	$m_1$
$T_{Ts}$	4	3	0.5	Transmission Task	-

- TDMA cycle length is 4ms and it takes a 0.5ms slot to transmit  $m_c$ , then let us draw the TDMA cycle:



So, in this case you see that the TDMA cycle length is 4. Like we discussed and it we have split the slots in this 0.5 milliseconds, I mean, slots, right. So, you have inside a cycle of size four you have these eight slots, and we can use this eighth slot for sending this message corresponding to this control loop right here, right. So, that is how it can be done and I mean it can be a design decision also we can also decide that well, I mean, if we send this message here, then there is a corresponding dependency that when should that reception task run in the compute unit 2 which is supposed to receive this message. So, let us see what can be done about that.

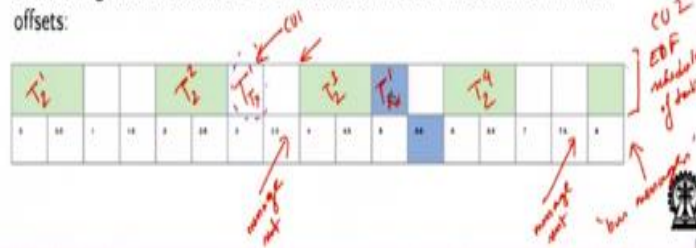
(Refer Slide Time: 25:08)

## Time-triggered Architecture: An Example

$T_s$  is an actuation task that receives  $m_c$  and actuates ~~the~~ the plant.  $T_{Rx}$  is the receiving task running in the CU2 for this purpose. Following are the details of the tasks.

Task	$p_i$ (ms)	$o_i$ (ms)	$e_i$ (ms)	Type	Message
$T_2$	2	0	1	Security Task	$m_2$
$T_s$	8	x	1	Actuation Task	-
$T_{Rx}$	8	y	0.5	Reception Task	-

- Considering that the tasks are EDF Scheduled in CU2 find out the minimum offsets:



So, now we focus on the compute unit 2, right. So, here you have let us say this following message is executing. So, you have this actuation task, you have the reception task. Let us say both are

executing with a period of 8, right. And their execution times are 1 and 0.5. They have interference from another task which is co scheduled in this compute unit 2, and this is a period of two and execution time of 1, okay.

Let us say this that the purpose of this task is again for doing some security monitoring, okay. And then we are trying to figure out well I mean what should be the exact timing of these messages and let us try to figure that out. So, this message, this task  $T_a$  we will receive this message  $m_c$  and it will actuate the plant. And this task  $T_{Rx}$  is the receiving task which is running in compute unit 2 for this purpose, that means, receiving the message precisely and let us look at it now.

So, first we will progress similarly, we will find an execution schedule of the tasks assuming EDF schedule. So, with EDF we have,  $T_2$  will have the higher priority. So, we will have  $T_2$  executing here and then again, it is coming after every period of 2, right. And then, if you see, let us just go back to our previous slide, where that was in the picture, right. So, if you remember from the discussion, we figured out that what is the time when the transmissions could happen, right.

And if you remember from my drawing of the schedule in compute unit time, compute unit one, we figured out that, that can happen at this slot, right, at 3.5. Somewhere at 3, this is where I had the task  $T_{Tx}^1$ . But this is not here really, this is happening in compute unit one. So, this is not really here I am just showing the timing, right. And based on that, we reserved this slot in the bus for sending the message.

At this 3.5, we were sending the message. That means another message would be coming at 7.5. So, these are the timings when the messages are being sent, right. Now we are trying to figure out well, when to execute this  $T_{Rx}$  task, because that is the task which is supposed to receive the message. So, I mean, let me just recap this again. So, we are drawing the timing diagram of messages and tasks in compute unit 2.

In compute unit 2, at the upper part of the diagram, we have EDF schedule of tasks in CU2, right. And here I have the message schedule. These are the messages, the bus messages. So, if you remember from the previous timing diagram of the computer unit 1, the message was sent right here and right here, at 3.5 and 7.5. These slots have been used. Now what does this mean? That means I have to execute Rx, right.

Now for executing Rx, there is no point in executing it anywhere before this and then again Rx does not have priority over  $T_2$ , right. So, it can only execute when  $T_2$  is not executing or  $T_2$  has not arrived, right. So, Rx can execute after this and then it has the message which it is supposed to receive from the queue of the transceiver chip in CU2 and send it to this task  $T_a$ , right. But it cannot start right now, because then it would be kind of preempted by  $T_2$ , right, which has a higher priority.

But the message is available so it can start any point after this, which means I can actually start  $T_{Rx}$  right here, right. So, I can actually start  $T_{Rx}$  right here, and I will execute it accordingly, okay, and similarly this thing can continue. So, similarly at 10 again, from 8 to 10, I will have  $T_2$ 's second instance executing and then I will have  $T_{Rx}$  executing like that, right. Now you can figure out where  $T_2$ ,  $T_a$  will execute.

So, these slots are going to be taken by  $T_2$ , I mean  $T_2$  we have already figured out. Then we can also figure out where  $T_a$  will execute, somewhere in between these bubbles. So, if we now take up all these things together and we try to create a complete schedule of messages. Let us see how it is going to work. So, if you see our discussion here, this is where we have pushed in the messages that is  $T_2$ , and we decided that well here the message arrives. So, we can execute  $T_{Rx}$  here, right, and so that is at 5, right.

**(Refer Slide Time: 31:30)**



$T_a$  and  $T_a$  can execute and then again  $T_2$  can continue. And that is how if you see we have built an entire system where we have some tasks executing in compute unit 1 following a scheduling algorithm. Some tasks are executing in compute unit 2 following similar EDF scheduling algorithm and we have chosen a suitable message slot on the bus inside the TDMA cycle, so, that the message slot satisfies the timing requirements of compute unit 1 and compute unit 2 as well as the applications design requirement. That well you send the message then you receive the message and then you actuate. You get the message only when  $T_{Tx}$  is executing and  $T_{Tx}$  gets the message only after  $T_c$  has executed. So, you see there are lot of dependencies from control to transmission then bus, then bus to reception, and then actuation.

And there are lot of interferences from other tasks and those things have also been satisfied nicely. So, if you look at this original problem definition of task models, we had two unknowns here. We had two unknown parameters here, which are this  $x$  and  $y$ , right. And we are trying to say that well, what should be and because these are both periodic tasks. We know there are periods we know their execution time of  $a$  and  $R_x$ .

We are saying that well we do not know their offsets. That means I mean the offset of  $R_x$  will actually correspond to its slot in the TDMA cycle, and the offset of  $T_a$  we will correspond to its start time inside a period, inside a periodic instance of the task. But here once we are trying to do the scheduling following the EDF and the other constraints. We see that well, like this message has been allocated a slot, we can allocate a start time to  $T_{Rx}$  which should be somewhere here.

And we can also allocate a start time to  $T_a$  which is 5.5 and then those are the offset values of this task, because that is when their first instance comes. So, if you see the period of  $T_a$  and  $T_{Rx}$  are both 8. So, we are supposed to execute one instance of  $T_a$  and  $T_{Rx}$  inside this time line of eight milliseconds and we can choose them like this. So, for  $T_{Rx}$  the offset is 5 and for  $T_a$  the offset is 5.5.

So, that would be the start times for the first instance and then they can just continue periodically and the entire system should work nicely with all the different constraints being satisfied. So, that is an example of how to design a bus and multiprocessor system with multiple task co-scheduled and task saving dependencies and messages having to be mapped into specific slots and choosing suitable slots for messages and choosing suitable start times for periodic tasks in different ECUs.

So, with this we will end this lecture and we will resume from here in the next lecture. Thanks for your attention.