Corellium

Video Webinar

Episode 15: *Next-Gen Automotive Software*

Full Transcript

(00:10)
Bill Neifert: Hello, my name is Bill Neifert. I'm the Senior Vice President of partnerships here at Corellium, and with me I also have Markus Helmling who is the product owner of SIL Kit from Vector Informatik in Germany. And we're here today to talk to you about next generation software challenges for automotive and we're going to have two different demos that we go over in here. One about mixed criticality workloads and the other demonstrating the software test environments that Vector provides. Let's talk a bit about the agenda. We'll first talk with an overview of the software problem itself, some of the challenges being faced, introduce leveraging the cloud to solve these problems, and then a bit about the Corellium virtualization technology, which underlies our virtual model approach. Here I will then hand over to Markus who will do an overview of Vector's technology and some of the solutions that they have done together with us, and he will incorporate that. Then a demo basically showing our i.MX 93 working together with their SIL Kit and CANoe products. He'll then hand it back over to me so I can give a brief chat about mixed critical software and then talk about that, how that plays into a demo that I will then give basically showcasing the autoware ADAS demo. And then there will be a bit of a wrap up followed by Q&A.

(01:39)
So onto the hardware software problem. And this is not a problem that's new to anyone attending this and that's that auto software is lagging in complexity. And this slide, which I stole from Arm here, talks about that and it basically shows the gap between what is possible with today's methodologies and what is needed by the cars of tomorrow as they incorporate more and more sensors, more automated driving features and such like that. And you can basically see that we're already behind and what's needed and by 2030 it's just going to be a gap that can't be closed if you apply standard methodologies in order to solve this. So if there isn't a change, we're just going to not be able to meet the needs of the next generation of vehicles, which obviously no one wants. So let's talk about how we solve that problem and when we can solve this by leveraging the methodologies that others are already doing.

(02:39)

I mean, others have had complex software problems before and it's been addressed largely by moving to the cloud. And the cloud is great because the cloud has a number of fantastic methodologies, repositories and flows that can be leveraged out there and if you virtualize the representation of a device, you can move it to the cloud and leverage these flows and methodologies. Of course clouds also come with APIs, ways to control debug and how these devices are interfaced too. So once you virtualize a device and place it in the cloud, you can take advantage of a lot of these flows. Another advantage of course is that you can scale up and down quickly and efficiently. Cloud instances are typically used as you need it and put aside when you don't, and this gives you the ability to scale the size of your solution based upon the size of your problem.

(03:35)
It also means that you get to scale your costs accordingly as well. Typically when you turn off cloud instances, you're no longer paying for them. Your mileage may vary depending upon your cloud package of course, but typically when you're not using cloud instances, they greatly go down in costs. Of course, the cloud is also fairly ubiquitous. You have the ability with virtual devices running in the cloud to share them amongst teams working worldwide. This means that a matter of working hours, location, et cetera. From there, everyone can be working with the same rev of the same instance and when fixes are created, they can be deployed instantaneously to teams throughout the world. You no longer need to worry about shipping boards around the world and what happens when they get out of rev or what happens when one gets lost in customs or the little blue wire changes that you made on there fall apart because they got roughly handled by someone.

(04:32)
And finally, by doing this and virtualizing these devices, you are no longer limited by the availability of the device itself. This means that you can start using a device as soon as that device model is available. Some of the models that we're going to be talking about today are available today, but the silicon that corresponds to them won't be available for another two years. This gives you the ability to start your software development much, much earlier in the process and ensure that by the time you actually do gain access to a physical device, you've got fully mature software which you're able to take advantage of. One thing that you'll see on some of the technology that we're talking about today though is that we're also giving you something that is available to shift right and still be available after physical hardware is available. And this is important because as you look at the complexity and cost of things like automobiles, you don't want to have a whole fleet of these against which you validate all of your software and all of your revs and do your OTA testing. What happens when you upgrade over the air from a version three versions ago to today's version? Did you test out what happens in that path and what happens when it breaks because it probably will. Did you just break a car? That's probably an expensive problem to have. Doing this entire thing with a virtual flow lets you take advantage of virtual devices even after you've shipped a device and have a more cost-effective and in some cases more performant platform as well.

(06:09)
Of course, Arm is well aware of this and with another slide that I've stolen from them here, and this is part of their latest IP announcement for the next generation of automotive IP. When they made that announcement, they highlighted four different areas inside of here. Of course you have the IP itself, but then you also had the system software, the virtual platform, and then the integrations that went along with all of this. And one of the reasons that of course Arm does this is because you have that ability to shift left. And so this is a slide from that same deck that talks about how virtual prototypes let you start that software development two years earlier and Corellium was at the heart of this announcement. Once again, this is a slide from Arm talking about what Corellium has to offer in this space. I wrote the right hand side of this and as part of this announcement, Corellium rolled out two virtual prototypes.

(07:13)
One is our Cortex-R82AE systems which have the ability to run OSs and hypervisors on them and enable you to do mixed criticality in the cloud. In fact, I'm going to be showing you one of our R82AE systems later on. And as you might guess, that system is available today. You can log into avh.arm.com and take advantage of this. The other is the Arm RD-1 AE system, which is their next generation. This is a full automotive SOC and I'll show you a block diagram of some of this later on. It's very complex. It has tens of processors inside of this representing across the arms portfolio. A, R, and M processors are inside of this and the partitioning inside of here gives you access to the full SOAFEE capabilities when it comes down to mixed criticality, partitioning, et cetera from there and like said we'll talk about that later on. Since this has a Neoverse V3AE inside of it, which is a V9 processor, it needs to run on a V9 core. And so we are waiting on commercial availability of the AWS Graviton4 in order to roll this out and make it commercially available to folks in the cloud. AWS has promised that to come out shortly, however, so we hope to have this remain commercially available shortly to folks as well. It also runs on AmpereOne systems for customers who want onsite offerings.

(08:46)
I've touched on Arm a few times and stolen a few of their slides and that's because Corellium has a very powerful relationship with Arm. Arm, a few years ago, decided to make an offering called Arm Virtual Hardware, which is basically virtual representations of Arm processors, subsystems and devices which run in the cloud. And here at Corellium we provide the technology for the devices and some of the subsystems that run out in the cloud. And for this we're leveraging our hypervisor technology to give you devices that typically run substantially faster than the actual hardware on here. With our auto parts, we're actually now rolling out systems well in advance of their silicon availability as well. So I talked a bit about Corellium hypervisor. Let me actually talk a bit about what Corellium does in this space. And if you'd come to us two years ago before the Arm relationship, you'd have seen a company focused entirely in the mobile segment.

(09:47)

And indeed we still have a good number of offerings here and are always expanding our presence here. We have the industry's only virtualized representation of Apple iOS devices as well as a large family of Android devices as well. And these devices include the latest and greatest devices, so the iPhone 15, running iOS 17 in jailbroken and non-jailbroken fashions and every Android release from 7 up to 14 in here with a whole array of features. These aren't just small areas of functionality. Things our Android devices have, Bluetooth camera, GPS, et cetera, inside of them in here and we provide these to folks who want to do security research, pen testing, vulnerability analysis and such from there and also take these and integrate them into a DevSecOps flow. So this has traditionally been Corellium core technology, but all of this takes and runs on top of a hypervisor and this hypervisor can be used to represent all of Arm's architectures, not just mobile phones.

(10:56)
And so this is a lot of what drove Arm to partner with us and leverage the functionality of our hypervisor. And this shows one of our technology stacks running on top of the hypervisor. And so you see at the bottom of this running directly on top of the physical hardware is the Corellium hypervisor for Arm or CHARM. And this hypervisor supports all our architectures A, R, M, V7–9, 32 and 64 bits and does this by mapping the processors one-to-one. So every processor that you have in your device maps to a processor on our underlying server. This is how we get a lot of our speed inside of here or processors that don't map directly in here. So R and M processors for example, don't have representations inside of servers. So we have the ability there to map those and basically dynamically remap those capabilities onto the underlying processor.

(11:57)
We do all of this by accurately modeling all of the peripherals inside of the design because our goal here is to not just let you execute the exact same binary as you would on your physical device but exercise the same binary paths as well. And that's extremely important for things like safety applications and testing because you always want to make sure that you're getting as much exact full code coverage in your virtual representation as you have in your physical one. As I mentioned earlier, we're able to do all of this while still delivering performance that is at or above real device speeds in most applications. And in fact, every application currently available on Arm Virtual Hardware, we are running multiple times faster than the board that we're representing. So one of the models that we're going to be talking about quite a bit today in our Virtual Hardware is the NXP i.MX 93 and this board of course is running the exact same binary as the real board.

(12:59)
We've used this and featured it in a number of demonstrations. We did one at Embedded World where we were showing of video and recognizing cars and identifying them and people and then keeping a count and then running that image side by side with a real physical board and we were running faster than the i.MX 93, but it shows some of this power. But on the left you can see basically a representation of the technology stack running on the side of this. So we've got the Corellium hypervisor for Arm down at the bottom being controlled with its REST APIs.

We've got the peripherals on the side that we're interfacing out to, and inside of this we've got basically the technology staff running on top of this. We've recently introduced the capability to run a hypervisor on top of our A processors. On top of this you can run one or more guest OSs and then run your applications inside of this.

(13:49)
Of course, the M33 is running its own OS inside of it with applications on here as well. The peripherals, we typically, we represent them very accurately, but we also understand that sometimes you may want to interact with the peripherals directly yourself. And so we have APIs here in order to let you leverage these. And in fact, Markus is going to be talking about this later on because the Vector integration takes advantage of our CoreModel API. We have the ability also to take some of these peripherals and remap them to the PC peripherals. In fact, I'll show you later on that the webcam for example on my PC, I can also redirect to be the webcam on the i.MX 93. The same thing with the audio and such on here as well. So you can see that we take advantage of a lot of the capabilities and our speed lets you do things that just aren't possible with any other virtual representation. So I talked to you about the i.MX 93. Why don't I show you the i.MX 93 and let me drag that from my screen over here onto the screen that I'm sharing and hopefully that's still sharing on here and expand this out.

(15:05)
And so this is the representation you see after you've booted the i.MX 93 itself. I created this from there, our device screen over here. I can create a variety of different devices. You see we've got the i.MX 93, Raspberry Pi, we have generic Linux so you can have your own test bench and things like that running there. Cortex-R52, another auto system that we have available. A number of FVPs from Arm, these are the 82AE. We'll be showing that later the NXP i.MX 8M and of course the ST Discovery Kit as well. So there's a large variety of devices available here. Let me go back to the i.MX 93 that I'm running and you see that I selected that from there. There are a variety of options that I have and let's walk through some of them here. The first thing you'd want to be able to do with your device is connect to it.

(16:01)
And so you have the ability to SSH directly into the device. We use a jump server in order to do this so you don't need to have any other authentication in place. If you want to network with it more robustly, however, we have open VPN drivers on here and so you can download an OVPN on here and once you've done that you can just SSH in without a jump server. I mentioned earlier we have CoreModel APIs to interface to all the peripherals. These can be downloaded here and it's being downloaded directly from GitHub. All of our interfaces like that are open source. Once you've launched a device and want to debug it, you can use GDB to simply interact with it. And we have accelerators here to basically cache some of the local responses and let you interact with it even more quickly. And this really comes in handy if you're not in the US since Arm Virtual Hardware is in the us. If you're not in the US, some debuggers might want to time out and things like that since they're dumping a lot of information. So this actually caches

the data, runs it locally and you can interact with it basically as fast as you do with the physical board.

(17:14)
If you want to interact directly with the land port of the device, you can of course do that as well and things like the GDB and other tooling is done over another ethernet interface as well. And those are typically presented here as well to give you the ability to do this. All of these by the way are also available to you via our API. So everything that I'm showing you here that I'm doing interactively can be using our APIs including finding out all this information when you're up and running, you have the ability to do a CoreTrace and basically trace the execution of any process thread inside of this. You can change other settings inside of your boot arguments that you have. You can upload your own firmware binary or you can do it at boot time as well, but you can do it now here and then reboot.

(18:04)
You can upload another Kernel to run with the M33. You can bring up a custom device tree. Say for example you don't want it to appear that there are certain devices, you can bring that in here. Or if you want to add your own custom extension in here, so you would add that to the device tree and then add in your own custom vMMIO, VirtualMemoryMap.io. This is the ability to add in your own behavior that basically appears as Memory map I/O to the device, but then actually generates network transactions which you would then attach to with your device and then generate and interrupt in order to do that interaction. It's all nicely documented on here. You can see the question mark here will take you to that and then you can modify the partition table as well. So we give you a good amount of configurability. On here, you have the ability to see the console on here, and you can see it performs basically just like the console as you'd expect.

(19:07)
And like I say, it does run multiple times faster than the physical board, so you won't have any issues interacting with this. The Cortex-M console is here as well. I'm not running anything on the Cortex-M, so it doesn't represent here. And then we have the sensors as I mentioned earlier, we have the capability of basically taking over the camera on my device here and you'll see that I've only got my laptop camera here available today. If I had more, it would list more. Similarly with my microphone, you'll see that I've got a few different microphones available that I can choose from or I can upload a file, bring it in here and I have this so I can repeat with that or I can direct to a file somewhere out on the network. I can modify the environmental sensors here interactively and set a different temperature.

(19:59)
I can modify the acceleration in Gyroscope sensors on here as well. Once again, I'm doing it here via the GUI, but of course these can be modified via the API as well. We finally have the capability of generating snapshots, which of course comes in very handy for debugging and sharing with other members on your team or creating a known good snapshot that I can use. I just mentioned sharing with your team. This is a new function that we're rolling out with our next

release that will give you the capability to create a snapshot and then share it with another member of your team. Maybe you found a bug that you think they're better qualified to work on and so you hand it off to them or maybe you get to a certain point and want to hand that known good device off to someone else as well. So there are multiple values that you can get out of this. So that is the demo here. Let me now bring the presentation back on across and I think this is actually now where I hand off to Markus. So Markus, take it away and I will be back after you're done to talk about our technologies from mixed criticality.

(21:16)
Markus Helmling: Hello. This is Markus Helmling from Vector, Germany. I work at Vector as a product owner responsible for the Vector SIL Kit. Today I'd like to give you a brief overview. Who is Vector, what is Vector CANoe and what is Vector SIL Kit? And then it's my pleasure to show you our current activities, our current proof of concept that we set up together with Corellium showing a software test scenario using the Corellium Virtual Arm platform together with our Vector tools and I will demonstrate this in a short live demo. And afterwards I will wrap up what we have seen. So who is Vector? Vector has been in the automotive market for almost 40 years. We have a quite wide spectrum and application areas that we cover mainly to assist and simplify the development of embedded devices spreading from embedded software that we provide to several types of tooling.

(22:48)
Today we will focus on testing, namely software testing using the product CANoe. So what is CANoe? CANoe was designed for analysis, logging and measurement in the beginning of CAN-based networks, but it was extended over time. So it covers a wide range of different physical layers that can be attached to CANoe and establish the communication to the system under test. Besides the analysis locking and measurement capabilities, it comes along with an environment simulation and then test executor. Today we are going further than that. CANoe is not restricted to physical setups or hardware in the loop setups. It can also be used for pure software testing. Besides that, the OE in CANoe stands for open environment. That means CANoe is not just a tool, it's an open platform that allows to interconnect or to interact with third party toolings and solutions. As you can see in the gray outline, what is Vector SIL Kit?

(24:23)
Vector SIL Kit is a rather new development from Vector. It's an open source library connecting software in the loop environments. SIL Kit connects test tools, simulators, emulators, virtual issues, models and software applications, meaning it connects almost anything. It's communication middleware that can be used on different levels and different stages in your development. It provides communication infrastructure for automotive and non-automotive applications and we develop SIL Kit in C++ without any platform dependencies. That means we have a great interoperability. Here that means you can run it on your Windows PC as well as in Linux or other UNIX derivatives. We focused on a stable API and API for long-term network layer compatibility and you can check out SIL Kit which is available, open source and free for everyone on GitHub on the link below.

(25:44)
Now we come to the actual proof of concept. It's an experimental SIL setup using the Corellium Arm platform. You can see the Corellium solution on the left-hand side running in the cloud. We selected a i.MX 93 model and our application that we want to test and that we are communicating with is running on some A core. We connect my local machine via some open VPN tunnels. So we have a private setup and on the right hand side we see the CANoe running on my local machine using SIL Kit to interconnect with the application using a virtual canvas. We did some integration over here so that Vector SIL Kit makes use of the CoreModel API provided from Corellium. The actual demo is quite small and easy. It consists of a small CAN network that spreads from the cloud via the virtual canvas to my local machine. We have two nodes located in the cloud. It's the engine, the lights, ECU. And on the right hand side we have a control and a display node running in CANoe. Then let's come to the actual live demo. So as you can see we fired up already the i.MX 93 model in the Arm platform

(27:32)
And when I switched to my local setup over here you can see that I connected to the cloud instance via an VPN tunnel. And if I start the measurement on CANoe side, we can already see some CAN communication happening and then if I switch on the ignition in the control panel, then the ignition info is transported via some cyclic CAN message to the engine node and the engine node in turn provides some RPM signal and you can see on the left hand side in the cloud and on the right hand side on my local machine, this is quite synchronous. The signal going up and down in this sawtooth manner as an example, it's quite responsive. You can switch on the lights. So you see lights. Engine also indicates that the lights is switched on, same as on the CANoe side. Same applies for the hazard indicator. So you can see that this is quite synchronous. If I switch off the engine then RPM goes down to zero and we can also see at the graphic window that the signals are off.

(29:12)
OK, that's it from this short demo. So what are the takeaways in this short demo? We could demonstrate that together with the Corellium Virtual Arm platform we can form a hybrid co-simulation setup and Vector SIL Kit can be used to interconnect the Corellium cloud-based Arm emulation platform with Vector CANoe using a virtual canvas. I just want to make clear that we are not limited to cloud, but we see this as a valid setup. We are not limited to CAN or system test. Of course we can also use other protocols or we can test on application level and we are not limited to this manual testing. We can also apply our automated tests and headless mode as possible too. For further info you can check out the links below. I think Bill will also summarize all the links in the last summary slide. Thanks for your attention. And then back to Bill.

(30:38)
Bill Neifert : Thank you very much, Markus. That was a fantastic demo. I love what you guys have done here and how great of a partner that Vector has been to us throughout this process.

The Vector integration goes beyond just the stuff we're doing with them with SIL Kit and CANoe. We're working with their embedded team on bringing up their OSs on various of our Cortex-R devices as well. So you can expect to see more information on that in future announcements and probably webinars. So let's talk about software criticality. And this is an area that a lot of the attendees here are probably much better versed in than I am on here. But I want to give an overview on this before I talk about running some mixed criticality workloads. And so software criticality as defined by everyone's favorite ISO 26262 standard basically gives around various automotive software integrity levels or ASIL levels here.

(31:29)
And basically they come down to how much harm they can cause when something goes wrong, it's rather gruesome when you look at it that way, but it kind of makes sense that you'd want to look at the impact on this and then assign to that what happens when things go wrong inside of your, or what level of testing do you need in order to ensure that nothing goes wrong. And so at the top of this is ASIL D Basically when things go wrong here, lots of people can die. Lots of impact to life and limb in here. So the highest potential for fatal injury, think of things like your airbags or braking systems. Obviously you want these tested to the very highest level inside of here. Stepping down in here you have things that are a little less severe. Think of cruise control. Cruise control goes bad and you start racing in theory you'll be able to press your brakes and it disengages, but still your cruise control can get you into some pretty bad situations in here.

(32:29)
Stepping down, you've got things like your headlights and brake lights, necessary safety features in here, but I've accidentally driven with my headlights off before in here. Thankfully someone flashed the lights, I turn 'em on, but obviously you want to make sure these are running but it's at a lower criticality. And then stepping down from that, you've got your running lights. Do I want to be seen at night for example inside of here? And of course there is a final standard setup for basically regular quality management. Everyone wants to have quality software, but this is allocated for something that has no safety impact in here. Obviously everyone wants to have the good stuff, but if there is no threat to life and limb for this, this is typically factored down into what they call quality management. How do you leverage all of this and why have a mixed critical?

(33:23)
Well, it's because you don't want to have to test everything to the highest level of criticality inside of this. As I mentioned, it's not just the establishment of these standards. What you need to do in order to meet these standards. If you've got something at ASIL D, you've got to do all sorts of extraordinary steps to validate that functionality. You've got to perform form analysis, fault injection, do flow analysis, et cetera on here. And in fact for ASIL D, it's got to run on an RTOS. You've got to be running this on something that has ECC and lockstep and all these other features. So you've got redundancy inside of this. Imagine trying to apply ASIL D standards to your regular workload inside of here. It basically puts a huge amount of burden on

the software development and you think we've got gaps between productivity and throughput now you start having to certify everything to ASIL D.

(34:20)
And so for a while now they have taken the ability to basically separate out and partition it such that you only run as much as you need for at the various certification levels and partition it out using containers or different processors, different processor types, et cetera like that. And so this would be the typical anatomy of a mixed critical system. And you'll see on the left hand side we've got things that are on the lower end of the criticality scale. And so I'm able to run them on regular compute course running with a hypervisor on top of them in order to run a number of guest OSs. Each one of these guest OSs is basically forming a nice partition inside of this. And on that I'm able to have software running at different levels of criticality inside of here tested to the required level of testing for their various settings over on the right hand side where things start getting a little more critical.

(35:18)
This is where we would take advantage of running on a real-time core inside of here and running an RTOS of that in order to leverage that. Modern real-time coreswcan run a hypervisor on top of that. So this one real-time core could be capable of running multiple RTOSs on the side of it and partitioning out as accordingly. And in fact Arm is well aware of this. So I talked earlier about the RD-1AE system that Arm announced when they announced their next generation IP and we are working on a model for this ourselves. In fact it's up and running in our test systems now. And you can take a look at this and this is the software architecture on the side of this. You can very quickly see that Arm has partitioned this straightforward mixed criticality inside of here. We've got on the left hand side, we've got our high-powered application processors.

(36:16)
These things are running a hypervisor button, then running multiple high-powered high level OSs inside of them. They're running Linux over here and this is going to handle our regular compute type stuff in here. When it comes down to safety application stuff that needs real time, your higher highest ASIL levels, well that's in the middle and we've got that partitioned out to run R82AE clusters in here and I think we've got one, two and four processors in these clusters and they're running Zephyr as an RTOS on side of this. Further to the right—obviously we've got a bunch of complexity on the left, so let's coordinate that with some control processors. And so Arm's architecture has that specified over here and this entire system, as I say up and running today on our development servers and will be available on Arm Virtual Hardware very soon.

(37:14)
This architecture is underlying a lot of the stuff that's being touted by SOAFEE. You can go to soafee.org on this. It's the Scalable Open Architecture for Embedded Edge and this is an organization set up by Arm and a number of other high-powered players in the automotive industry to basically establish standards around how to do a lot of the things that I've been

talked about here. And one of the things that they have introduced as part of this is the Open AD Kit. Basically a way to package software to take advantage of all of this.

(37:55)
And when RD-1 AE comes out, you'll be able to take Open AD Kit binaries and load them directly on top of that. And this is a diagram of one of these systems and in fact it's showing what I'll be showing in my demo today and that's a software package from Autoware and Autoware is the open source ADAS stack out there that's being commercialized by tier four and in this case in the primary compute which will be running on the Neoverse V3AE, we've got the interface software on top of all of this, the orchestration level running inside of here. And basically the presentation layer of all of this is communicating with the safety island, which is where our realtime cores are running over here to basically run the control features. When we show this, I'm going to showing you laying out a map with the compute cluster but then transferring that map over to driving instructions. And think about it driving is something you really want to control by real-time processors in the Safety Island over here.

(39:00)
And so it's showing a great example of that mixed criticality. And this plays into the demo that I'm going to be showing you because as I mentioned the RD-1 AE itself is not available yet today. But the demo that I'll be showing you shows a lot of these same concepts. So on the left hand side running the Autoware and the orchestration layers on here will be a regular EC2 instance Graviton3 running in the Amazon cloud, AWS east-1 for those of you that care, it'll be communicating over ethernet to the Corellium model for the Cortex R82AE, which will be running Zephyr. And then basically the control features running from the side of this. So the whole thing's going to be executing in the cloud. So mixed virtualization running entirely on the cloud, but let's say it'll be partitioned out such as you're aging generic EC2 instances for one part and then Corellium instances for the other.

(39:58)
Unlike Markus, I'm not going to do a live demo on this. I've had that go wrong on me way too many times in my years of doing this, but I did set this up earlier and so I've got that demo prepared here and this gets me the ability to overlay and et cetera like that on here. And before I press play, let's talk a bit about what we're seeing here on the left hand. Two thirds of the screen is Autoware. This is their GUI basically showing all of the controls on the left, the map that's going to be in the center and we will place a car on that here in the middle. And on the right is basically the console display from the R82AE. It doesn't need much of a display so it doesn't have much of a display on it.

(40:38)
So as I go through and press play on this, you'll see that it's going to start out and we're going to do a pose estimate, which is basically going to involve placing the car here to establish its starting location. And you'll see we're going to be placing that on the left-hand side of the road. Autoware is being commercialized in Japan where they drive on the left-hand side of the road.

So having laid out the initial position on here, we will then basically mark where it's going. So once again laying that on the left-hand side of the road, we'll place this up here and basically lay out a path of it taking that turn. And you'll see as soon as we placed it, and I've just pressed pause on this. As soon as we placed it, the actuation processor over here on the right started being active.

(41:25)
It's basically now being pulled by this to basically start doing some of the calculations on what's going to be needed to be done in order to start out this route. Now pressing play on this and continuing on, you'll see that it's doing the calculations and we've lit up the auto button over here and we're now going to click the auto button to basically say automatically drive this route. And so now you'll see the output from the actuation processor is changing as we calculate our speeds and our turning angle is good on this and we're showing this on the screen as well. And basically this is all being then executed out by the car being rendered out and displayed by the application processor as the car goes through the trajectory that was mapped out and determined by the Autoware software. When it comes to the end of its route, it reaches that end, it straightens out the steering wheel, it pulls off the accelerator as you can see over here because the angle in radiance has gone down to zero degrees and the media per second squared has gone down to zero as well, basically indicating that we're no longer steering, the car has come to a full stop.

(42:30)
So this is a nice dramatization of what's possible when you run mixed criticality inside of the cloud.

(42:40)
So let's go on to our next slide. I think I can do that here. Yes. So as I mentioned the R82AE here is running Zephyr and Zephyr is an RTOS running on a representation of a real time processor. And one of the important things that's always important when you're running an RTOS is predictability. You want to be able to execute the same amount of software in the same amount of time and basically have the interrupts that everything aligns to take the same amount of time. And if you look at the timing predictability slide that I've got here, you can see that the CHARM model, which is represented here in blue, delivers that. What I've got here is a representation of how long each system tick takes while booting Zephyr. And for this instantiation of Zephyr, a real system tick in the real system should take about six tenths of a second.

(43:38)
That's what it is in the real system and we're lining up all of our interrupts on that. So obviously predictability on this is very important and that's what we deliver. If you take a look at the classic approach on this and doing it with traditional virtual models, however, you'll see that it does not deliver tick predictability. It varies anywhere between half a second on here to nearly seven seconds per tick on the side of here. And it's basically varying based upon the workload that it's

executing. So you'll see that it's not delivering on the promise of real time because it's not able to deliver the same amount of execution and align things back up in a predictable amount of time. And I'm not meaning to malign existing technologies on this, they just didn't have a throughput in order to take advantage of the real-time capabilities. Something that we're able to do with the CHARM models.

(44:34)
So let's change gears here a bit on this. I've talked a bit about the Virtual Hardware and everything we've done so far has been up and running in our cloud and indeed Corellium's own offerings as well as Arm Virtual Hardware run in the cloud on AWS. And if you want to engage with that, you can go sign up today for free@avh.arm.com and start up and get up and running and you can get your 24 free device hours to spend over 30 days and you'll have a QR code for that link at the end if you don't want to run in the cloud for whatever reason—security, desire to run things on site, IP protection, et cetera. From there we have the ability to take these same workloads and run them onsite as well. We have a variety of NPR appliances, which we supply for this and give you 64 processors upon which to run your designs or these same designs.

(45:30)
Similarly, if you do want to run in the cloud, but we want to run in your cloud, we have the ability to take what we're doing with AVH or our own offering and run it in your own VPC. So basically now it's an API that can run on your own bare metal instances. Finally, we have the ability to run on a desktop appliance, which is a handy little about an eight-inch square device that can sit there on your desktop and execute any device that will fit inside of it. And I believe there are 10 available processors inside of this. We take that same desktop appliance and make it available with a Developer Kit on top of it as well if you want to develop your own models. All the models that I've shown to you here today have been developed by Corellium, but we have the ability and have partners and technologies that enable you to develop these same models yourself.

(46:26)
So, we can license to you one of our server desktop appliances that comes along with training model source and things like that to let you develop your own code that runs on top of this. So that wraps up our talk today. I definitely want to thank Markus for joining me and for the fantastic demo and overview that he gave. I've taken and generated QR codes for a lot of the links to more information there on the right. I've given a number of links for some of our representations as well. I talked about AVH, I talked about CoreModel as well as there's also a link to our auto page and an app note on how to use the CoreModel interfaces that we laid out. If you would rather interface with any of us directly, you can feel free to send an email to either myself or Markus as well and we'd be more than happy to get back to you. Thank you so much for your time today and I look forward to working with you.

###