A SOFTWARE AND HARDWARE ARCHITECTURE FOR NEXT-GENERATION AUTOMOTIVE SYSTEMS

DR RICHARD WEST
CHIEF SOFTWARE ARCHITECT
DRAKO MOTORS

PROFESSOR
BOSTON UNIVERSITY
Drako GTE
Vehicle Growth in Electronics

• Electric vehicles, ADAS, IVI, V2X driving up cost and complexity of electronics

• Modern luxury vehicles have 50-150 ECUs
  source: Strategy Analytics, IHS Markit

• Global ECU market $63.6 billion (2018)
  source: grandviewresearch.com

• Electronic share of total vehicle cost is rising exponentially

source: Statista 2017
Vehicle Growth in Electronics

- Electric vehicles, ADAS, IVI, V2X driving up cost and complexity of electronics
  
- Modern luxury vehicles have 50-150 ECUs
  
- Global ECU market $63.6 billion (2018)
  
- Electronic share of total vehicle cost is rising exponentially

How do we reverse the trend?

(source: Statista 2017)
Automotive Software Complexity

Growth in automotive electronics has given rise to growth in software complexity

Source: https://informationisbeautiful.net/visualizations/million-lines-of-code/
Software Explosion

Software growth driven by increased vehicle functionality + increased ECU count

- 1970: Fuel injection
- 1980: Airbags, ABS, OBD
- 1990: Infotainment, traction & stability control
- 2000: Driver aids, connectivity
- 2010: Advanced Driver Assistance Systems (ADAS)
- Fully electric, autonomous & connected vehicles

- Cost, Complexity & Security Risk
- ECU count: <10k lines of code
- 100M+ lines of code

DRAKO
ADAS – SAE 6 Levels of Driving Automation

0  No Automation  Manual control
1  Driver Assistance  Single automated function e.g., active cruise control
2  Partial Automation  Automated steering and acceleration; Human can take control

Human monitors the driving environment

Based on: https://www.synopsys.com/automotive/autonomous-driving-levels.html
ADAS – SAE 6 Levels of Driving Automation

0. No Automation
   Manual control

1. Driver Assistance
   Single automated function e.g., active cruise control

2. Partial Automation
   Automated steering and acceleration; Human can take control

3. Conditional Automation
   Environmental Detection; Human override required

4. High Automation
   Vehicle performs all driving tasks under specific circumstances; Requires geofencing and human override is still possible

5. Full Automation
   Vehicle performs all driving tasks under all conditions, without human interaction

Based on: https://www.synopsys.com/automotive/autonomous-driving-levels.html
Hardware & OS Evolution

AUTOMOTIVE DOMAIN

• 8 → 16 → 32 bit microcontrollers

PC DOMAIN

• 64-bit CPUs, integrated GPUs
## Hardware & OS Evolution

<table>
<thead>
<tr>
<th>AUTOMOTIVE DOMAIN</th>
<th>PC DOMAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 8→16→32 bit microcontrollers</td>
<td>• 64-bit CPUs, integrated GPUs</td>
</tr>
<tr>
<td>• Mostly single core, single function</td>
<td>• Multicore, multiple tasks</td>
</tr>
</tbody>
</table>
# Hardware & OS Evolution

<table>
<thead>
<tr>
<th>AUTOMOTIVE DOMAIN</th>
<th>PC DOMAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 8→ 16→ 32 bit microcontrollers</td>
<td>• 64-bit CPUs, integrated GPUs</td>
</tr>
<tr>
<td>• Mostly single core, single function</td>
<td>• Multicore, multiple tasks</td>
</tr>
<tr>
<td>• Typically 10s-100s MHz</td>
<td>• GHz clock speed, hardware virtualization</td>
</tr>
</tbody>
</table>
## Hardware & OS Evolution

<table>
<thead>
<tr>
<th>AUTOMOTIVE DOMAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 8→ 16→ 32 bit microcontrollers</td>
</tr>
<tr>
<td>• Mostly single core, single function</td>
</tr>
<tr>
<td>• Typically 10s-100s MHz</td>
</tr>
<tr>
<td>• NXP/Freescale PowerPC, Infineon ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PC DOMAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 64-bit CPUs, integrated GPUs</td>
</tr>
<tr>
<td>• Multicore, multiple tasks</td>
</tr>
<tr>
<td>• GHz clock speed, hardware virtualization</td>
</tr>
<tr>
<td>• Intel &amp; AMD x86, ARM Cortex-A</td>
</tr>
</tbody>
</table>
Hardware & OS Evolution

**AUTOMOTIVE DOMAIN**
- 8→ 16→ 32 bit microcontrollers
- Mostly single core, single function
- Typically 10s-100s MHz
- NXP/Freescale PowerPC, Infineon ...
- Integrated CAN, GPIOs, ADCs

**PC DOMAIN**
- 64-bit CPUs, integrated GPUs
- Multicore, multiple tasks
- GHz clock speed, hardware virtualization
- Intel & AMD x86, ARM Cortex-A
- USB, PCIe, Ethernet, WiFi
# Hardware & OS Evolution

## AUTOMOTIVE DOMAIN
- 8 → 16 → 32 bit microcontrollers
- Mostly single core, single function
- Typically 10s-100s MHz
- NXP/Freescale PowerPC, Infineon ...
- Integrated CAN, GPIOs, ADCs

Simple RTOS
- OSEK, FreeRTOS, Tresos, ECOS ...

## PC DOMAIN
- 64-bit CPUs, integrated GPUs
- Multicore, multiple tasks
- GHz clock speed, hardware virtualization
- Intel & AMD x86, ARM Cortex-A
- USB, PCIe, Ethernet, WiFi

Complex General Purpose OS
- Windows, Mac OS, Linux
Automotive System Challenges

Reduce electronic costs

- Replace ECUs with fewer hardware components
  - e.g., multicore industrial PC
- Consolidate ECU functions as software tasks
  - Easier to update, reconfigure, extend
- => Need for functional consolidation
Automotive System Challenges

Reduce electronic costs

- Replace ECUs with fewer hardware components
  - e.g., multicore industrial PC
- Consolidate ECU functions as software tasks
  - Easier to update, reconfigure, extend

=> Need for **functional consolidation**

Address emerging real-time I/O needs

- Combined low-latency & high bandwidth data processing
- Google’s self-driving car (2013) ~ 1GB/s data
Automotive System Challenges

Reduce electronic costs

- Replace ECUs with fewer hardware components
  - e.g., multicore industrial PC
- Consolidate ECU functions as software tasks
  - Easier to update, reconfigure, extend

=> Need for functional consolidation

Address emerging real-time I/O needs

- Combined low-latency & high bandwidth data processing
- Google’s self-driving car (2013) ~ 1GB/s data
  A.D. Angelica: http://www.kurzweilai.net/googles-self-driving-car-gathers-nearly-1-gbsec

Functional safety and security (e.g., ISO26262, ISO21434)
Automotive System Challenges

Functional Consolidation => Need new vehicle OS

- Manage 100s of tasks on multiple cores
- Handle real-time low & high bandwidth I/O
- Provide safety, security and predictability
- Support mixed-criticality, fast boot, power management

Prohibitive complexity to write new OS from scratch

- Combine real-time with legacy code
- e.g. small RTOS + Linux
- Symbiotic solution
Vehicle Vulnerabilities

Functional Safety (e.g., ISO26262) + Cybersecurity (e.g., ISO21434)

- ASIL classification based on Hazard Analysis and Risk Assessment
- ASIL = Exposure [E0-4] x Controllability [C0-3] x Severity [S0-3]

Example:
Vehicle Vulnerabilities

Functional Safety (e.g., ISO26262) + Cybersecurity (e.g., ISO21434)

- ASIL classification based on Hazard Analysis and Risk Assessment
- ASIL = Exposure [E0-4] x Controllability [C0-3] x Severity [S0-3]

Example:

- Brake lights (ASIL B)
- Powertrain: Unwanted acceleration (ASIL D)
- Active Cruise Control (ASIL C)
- Rear lights (ASIL A)
- Airbags: Failed or inadvertent deployment (ASIL D)
- Power Steering (ASIL D)
- Headlights (ASIL B)
- Instrument Cluster (ASIL B)
- Antilock Brakes: Failed or delayed braking (ASIL D)

Remote Surface Attacks
- Wi-Fi, Cellular, FM/AM radio, TPMS, Remote Keyless Entry, Bluetooth

ADAS Failures
- Lane Keep Assist, Lane Departure Warning, Collision Avoidance

CAN Attacks
- e.g. Miller & Valasek, 2014 Jeep Cherokee CAN attack via Uconnect IVI Head Unit
Moving Forward: DriveOS
DRAKO DriveOS

DriveOS supports traditional hardware functions as software tasks running on a multicore virtualized platform.
DriveOS supports traditional hardware functions as software tasks running on a multicore virtualized platform.
DRAKO DriveOS I/O

**USB-centric solution**: works with legacy devices + supports higher bandwidth future needs

*Secure access to USB + CAN mediated by trusted I/O sandbox in DriveOS*
Reference Design: DRAKO GTE DriveOS

- Thermal Control
- Powertrain Control
- Infotainment
- Battery Management
- Cloud Connectivity
- Drako DriveOS Intel PC

- ADAS
- Suspension Control
- Instrument Cluster
- Body Control
- Fast Charging
DRAKO DriveOS Reference Stack

Cloud Services Layer

Secure V2X Communication Layer

<table>
<thead>
<tr>
<th>Powertrain &amp; I/O Services</th>
<th>Chassis</th>
<th>Functional Safety</th>
<th>Battery &amp; Thermal</th>
<th>ADAS</th>
<th>Instrument Cluster</th>
<th>Infotainment</th>
<th>Connected Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCPU + USB Scheduler</td>
<td>RTOS (Quest)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real Time Device I/O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure Separation Kernel (Quest-V)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real Time Secure Shared Memory Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Legacy Device I/O</td>
<td>Linux/Android GPOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Layer: CPUs, Memory, I/O (Direct Access)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Layer (Direct Access)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DRAKO DriveOS Functional Overview

USB 2/3.x Host Controller (CAN, LIN, GPIO, ADC, Cameras, LiDAR, IMU, …), UART

USB2 eHCI Host Controller (Touchscreen), Graphics, Bluetooth, WiFi, Cellular, Audio, Storage, UART

Secure Shared Memory Communication

Connected services
- Mender server
- Telemetry
- Remote access
- MapBox (maps/navigation)
- HERE (locations)

Secure Shared Memory Communication

Apps (navigation, Youtube Music, etc.)
Infotainment
Qt 5

Message server
- Poky
- Yocto Linux

Fault Mgmt
- HVAC
- VCU
- Body Control
- BMS
- Power Mgmt

OVE DO-178

CAN-IO Gateway
- VCPU API
- Pthreads
- CAN-IOlib
- Shmcomm

Quest RTOS
- Quest-V

CAN-IOlib
Pthreads
VCPU API

VCU API
Pthreads
CAN-IOlib
Shmcomm

Secure Shared Memory Communication

USB 2/3.x Host Controller (CAN, LIN, GPIO, ADC, Cameras, LiDAR, IMU, …), UART

USB2 eHCI Host Controller (Touchscreen), Graphics, Bluetooth, WiFi, Cellular, Audio, Storage, UART

USB 2/3.x Host Controller (CAN, LIN, GPIO, ADC, Cameras, LiDAR, IMU, …), UART

USB2 eHCI Host Controller (Touchscreen), Graphics, Bluetooth, WiFi, Cellular, Audio, Storage, UART
**Quest-V Separation Kernel** *(VEE’14, ACM TOCS’16)*

- Monitors partition CPU cores, RAM, I/O devices among sandboxed guests
- Monitors have small trusted compute base – no runtime resource management

![Diagram showing sandboxed guests with monitors partitioning PCPUs, Memory, and IO Devices.](image)
Quest-V Separation Kernel (VEE’14, ACM TOCS’16)

- Partitioning hypervisor – statically partitions resources
- Separation kernel – distributed collection of sandboxed components, indistinguishable from separate private machines for each component
Quest RTOS (RTAS’11)

VCPUs are first-class entities within the RTOS
- Budgeted real-time execution of threads and interrupts
- Tasks → Main VCPUs (Sporadic servers: budget & period)
- Interrupts → IO VCPUs (PIBS: derive budget & period from Main VCPU)

Real-time IO
- IOVCPU_CLASS_USB
- IOVCPU_CLASS_NET
- IOVCPU_CLASS_GPIO
- ...
VCPU Control Flow

1. Blocking I/O System Call
2. Program Device
3. Device Interrupt
4. Top Half wakes up IO VCPU
5. IO VCPU wakes up Main VCPU
VCPU Scheduling (RTAS’11)

Sandbox with 1 PCPU, n Main VCPUs (SS), and m IO VCPUs (PIBS)

- $C_i = \text{Budget Capacity of Main VCPU, } V_i$
- $T_i = \text{Replenishment Period of } V_i$
- $U_j = \text{Utilization factor for I/O VCPU, } V_j$

- Utilization bound feasibility test (with rate-monotonic scheduling of VCPUs):

$$\sum_{i=0}^{n-1} \frac{C_i}{T_i} + \sum_{j=0}^{m-1} (2 - U_j) \cdot U_j \leq n \cdot (\sqrt{2} - 1)$$
Single x86 Multicore PC Solution

Map all services to a single industrial automotive PC

Cincoze DX1100

Real-time I/O via Tuned Pipes

Yocto Linux

DriveOS

Communication

Kernel Plugin

Quest RTOS

CAN-IO Gateway

USB-CAN-IO Stack

USB-CAN-IO I/F

Instrument Cluster

In-vehicle Infotainment

ADAS
Tuned Pipes (RTSS’18)

Like POSIX pipes but guarantee throughput and delay on communication
- Simpson’s 4-slot (asynchronous) & FIFO (synchronous) buffering

**Boomerang** I/O subsystem in Quest-V supports real-time pipelines across Quest RTOS and legacy OSes
- Rate match tasks in pipeline to avoid blocking or missed data
- Quest appears as a **real-time virtual device interface** to Linux/Android
DriveOS: Example OpenPilot ADAS + IC + IVI (EMSOFT’21)

Latency path 1 to 6?

*Compare with Linux-only system with PREEMPT_RT & SCHED_DEADLINE tasks
DriveOS: OpenPilot Control Loop Latency (EMSOFT’21)

- ADAS Control Loop End-to-end Latency in presence of background Linux tasks
  
  *Target bound = 10ms*

*Both Linux cases use PREEMPT_RT. Optimized Linux maps USB interrupts to separate core*
Conclusions

Now is the time to look to alternative hardware + OS automotive solutions

DriveOS uses hardware virtualization for real time temporal and spatial isolation of software functions

+ Multicore PC-class platform replaces ECUs with software tasks
+ USB-centric I/O control
+ Symbiosis between RTOS & legacy OS
+ Real-time I/O & task pipeline processing

Fast startup of critical services on PC-class hardware (RTAS’22)
Key Points

- **Functional consolidation** to drive down costs of electronics
- **Centralized software stack** to reduce hardware + code complexity
- **Must consider OS challenges**
  - More than just supporting driverless & connected cars
  - ML is great for object detection but an RTOS is needed to avoid objects!
- **Real-time I/O is critical**
## Related Work

<table>
<thead>
<tr>
<th>Automotive Company / System</th>
<th>Operating System</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DRAKO DriveOS™</strong></td>
<td>Quest RTOS, Quest-V Separation Kernel + Yocto Linux / Android</td>
<td>Centralized; Quest/Linux/Android sandboxes IC, IVI, HVAC, Powertrain, ADAS, etc. Simulink Multi-OS Support</td>
</tr>
<tr>
<td><strong>Toyota Entune 3.0 (Future: Arene)</strong></td>
<td>Automotive Grade Linux (Arene: Apex.OS)</td>
<td>Infotainment (Arene will support autonomy)</td>
</tr>
<tr>
<td><strong>BMW OS7 and OS8 (iX)</strong></td>
<td>Greenhills Integrity RTOS + Linux</td>
<td>[Linux] Infotainment, IC [RTOS] RT vehicle control functions</td>
</tr>
<tr>
<td><strong>Polestar + Google</strong></td>
<td>Automotive Android</td>
<td>Infotainment</td>
</tr>
<tr>
<td><strong>Nvidia Drive OS</strong></td>
<td>Nvidia Hypervisor, QNX Neutrino RTOS, Linux</td>
<td>ADAS, Linux + QNX SDK</td>
</tr>
<tr>
<td><strong>Ford Sync 3</strong></td>
<td>QNX (current); Android (future)</td>
<td>Microkernel, RTOS, Infotainment</td>
</tr>
<tr>
<td><strong>TTTech</strong></td>
<td>Car.OS</td>
<td>Supports AUTOSAR, Linux, QNX + others <strong>Centralized</strong>: IC, IVI, ADAS, HVAC, Powertrain</td>
</tr>
<tr>
<td><strong>Mercedes Benz</strong></td>
<td>MB.OS</td>
<td><strong>Centralized</strong>: RTOS + Linux support IC, IVI, ADAS, Body Control, HVAC</td>
</tr>
<tr>
<td><strong>Tesla</strong></td>
<td>Linux + FSD (Full Self Driving)</td>
<td>Infotainment (AMD for Model 3 &amp; Y), ADAS</td>
</tr>
</tbody>
</table>
Questions?
Extra Details
System Software Safety

Temporal and Spatial Isolation
  • Ensure critical tasks are free from interference from less critical tasks

Timing and Functional Safety
  • Ensure timing-critical tasks meet deadlines
  • Functionally correct output values for given inputs

Correct Information Exchange
  • No loss, duplication or corruption of data

Memory Safety
  • No buffer overruns, stack under/overflow, invalid memory addressing

IO Safety
  • Controlled access to IO devices
System Security

Integrity
- Avoid attacker compromising critical functionality
  - e.g., Miller & Valasek, 2014 Jeep Cherokee CAN attack via remote access to IVI
- Resource partitioning and access only via secure interfaces
- Validate arguments to functional interfaces

Confidentiality
- Avoid leaking sensitive data (CAN packets, personal information, app data,...)
- Encrypt data or enforce information flow policies
- Eliminate side channels (e.g., via caches – possibly use cache/page coloring)
- Use containerization for critical components

Access Rights
- Avoid user gaining elevated accesses to resources beyond allowed rights
  - e.g., CVE-2019-5736 Breaking out of Docker via RunC
- Enforce a capability mechanism on access to resources
- Digitally sign software images
Today’s ECU Vehicle Network

- **Diagnostics**
  - **Powertrain Gateway**
    - CAN / Flexray
      - Transmission Management
      - Engine Management
      - Battery Monitoring
      - Alternator Regulator
  - **Body & Comfort Gateway**
    - CAN / LIN
      - Window Lift
      - HVAC & Comfort
      - Interior/Exterior Lighting
      - Door & Seat Modules
  - **Chassis Gateway**
    - CAN / LIN / Flexray
      - Steer by Wire
      - Brake by Wire
      - Power Steering
      - Tire Pressure Monitoring
  - **Infotainment Gateway**
    - Ethernet / MOST / CAN
      - Head Unit
      - Head Up Display
      - Navigation
      - Instrument Cluster (IC)
**DRAKO DriveOS**

Leverage the Quest-V separation kernel
- Open Source
- Partitions CPU cores, RAM, I/O devices among guests

Co-locate Quest RTOS with Linux and Android guests on same hardware

Real-time interface for device I/O
+ Processing moved to PC
+ I/O via e.g. USB-CAN or custom control-class interface
Example: Quest-V for DriveOS

- Real-time Control, Sensor Data Processing and Actuation
- Linux Applications
- Libraries and Runtime
- Linux Kernel
- Shared Memory
- Core 1 ... i
- Core (i+1) ... m
- RAM
- RAM
- USB, CAN (LIDAR, Cameras, IMU, ...)
- Non-RT Devices

Cincoze DX1100 Industrial PC
Cache Partitioning (Spatial and Temporal Isolation)

- Shared caches controlled using color-aware memory allocator [COLORIS – PACT’14]
- Quest-V uses EPTs to map guest physical to machine physical addresses
- Last-level cache occupancy prediction based on h/w performance counters
  - local (core) + global (all core) cache hits and misses between scheduling points [Book Chapter, OSR’11, PACT’10]
Quest RTOS – USB Scheduling (RTAS’13)

USB 2/3.x Bus scheduler

Each periodic request represented as a tuple \((w_i, t_i)\)

- \(w_i\) – time to send transaction \(i\)
- \(t_i\) – time interval of transaction \(i\)

Given set of \(n\) tuples \(\{(w_1, t_1), (w_2, t_2), \ldots, (w_n, t_n)\}\), is there an assignment of USB transactions to 125uS microframes, such that no frame is over-committed?

A request assigned to microframe \(f\) is also assigned to microframe \(f+n*t_i\), \(n \in \mathbb{N}\)

Using variant of first-fit decreasing packing algorithm, shown to outperform Linux

- Sort by decreasing \(w_i\) (largest first)
- First pick request based on smallest \(t_i\), breaking ties with largest \(w_i\)
Quest RTOS – USB Scheduling (RTAS’13)

- Consider all permutations of 1 to 5 requests
- Intervals: 2, 4, 8, 16 microframes
- Packet Sizes: 32, 64,…,1024 bytes
- Quest ≈ 150 thousand failed schedules
- Linux ≈ 95 million failed schedules
Boomerang Inter-OS Task Pipeline Example (RTAS’20)

Boomerang tuned pipe path (1) spans Quest + Linux + USB-CAN

Boomerang tuned pipe path (2) spans Quest + USB-CAN
Boomerang sub-system in DriveOS meets communication timing guarantees

A Linux SMP (multicore) OS with real-time extensions cannot perform I/O predictably
Jumpstart Power Management (RTAS’22)

- PC hardware requires Firmware POST, bootloader, device & service initialization to boot OS
- DriveOS uses Jumpstart ACPI S3 suspend-to-RAM & resume-from-RAM for low latency restart of critical tasks (e.g., CAN gateway services)
Jumpstart Power Management (RTAS’22)

- Jumpstart services span all guests
  - RTOS coordinates suspension but enables parallel reboot
- Potential for ACPI S4 suspend-to-disk using non-volatile memory (e.g., Intel Optane)
  - Eliminates system power usage during suspension
DriveOS: Screenshot 3/4
Simulink Multi-OS Modeling and Code Generation

- Model-based design for Multi-OS target
- Automatic support for nested ELF binaries with inter-sandbox RPC bindings
Shared memory inter-task/sandbox communication

```
channel_no

data

ShmcommAsyncWriteQuest success
```

```
channel_no

input_data

ShmcommAsyncReadQuest data
```

```
channel_no

data

ShmcommSyncWriteQuest success
```

```
channel_no

input_data

ShmcommSyncReadQuest data
```

Time management

```
time_since_last_called

ShmcommAsyncWriteQuest time_since_lastCalled
```

```
time_from_start

ShmcommAsyncReadQuest time_from_start
```

CAN-bus Management

```
channel_no

message

length

can_flag

timestamp

can_status
```

```
channel_no

id

message

can_status

length

canWriteMatlab

timestamp
```

```
channel_no

id

message

can_status

length

canWriteMatlab

timestamp
```

```
channel_no

can_handle

CANChannelSetup
```

```
channel_no

cANbusoff

CANBusoff
```
Example: Quest HVAC CAN ↔ Shared Memory Logic
Mapping a Function to a Quest VCPU

Configurable Parameters:
1. Target Sandbox:
2. Task Budget (C)
3. Execution Period (T)

A VCPU is bound to an automotive function via the output signal link of the vcpusetup block.

Set up new channel or connect to an existing one.