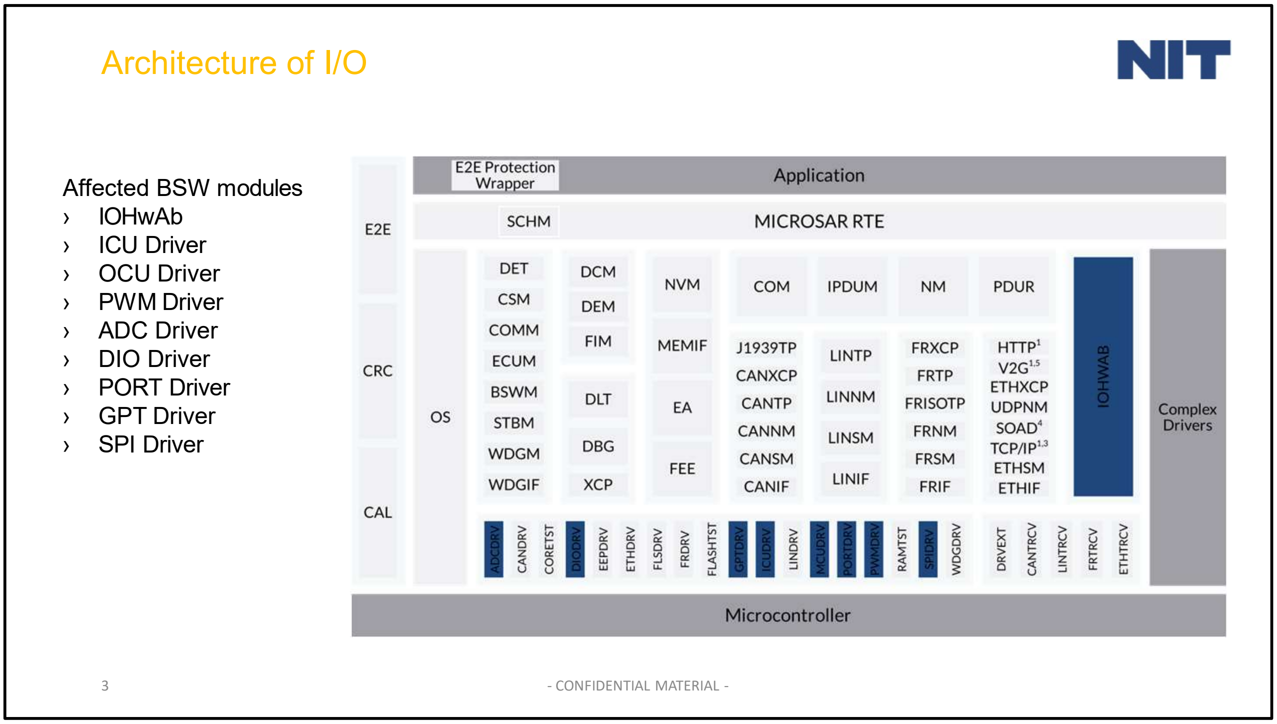
NIT

AUTOMOTIVE SOFTWARE WITH AUTOSAR

AUTOSAR

Agenda

- **Architecture of I/O**
- PORT Driver
- DIO Driver
- I/O Hardware Abstraction
- Client Server Concept



Architecture of I/O

Affected BSW modules

- IOHwAb
- ICU Driver
- OCU Driver
- PWM Driver
- ADC Driver
- DIO Driver
- PORT Driver
- GPT Driver
- SPI Driver

E2E Protection Wrapper

Application

E2E

SCHM

MICROSAR RTE

CRC

OS

DET
CSM
COMM
ECUM
BSWM
STBM
WDGM
WDGIF

DCM
DEM
FIM

DLT
DBG
XCP

NVM
MEMIF
EA
FEE

COM
IPDUM
NM
PDUR

J1939TP
CANXCP
CANTP
CANNM
CANSM
CANIF

LINTP
LINNM
LINSM
LINIF

FRXCP
FRTP
FRISOTP
FRNM
FRSM
FRIF

HTTP[1]
V2G[1,5]
ETHXCP
UDPNM
SOAD[4]
TCP/IP[1,3]
ETHSM
ETHIF

IOHWAB

Complex Drivers

CAL

ADCDRV CANDRV CORETST DIODRV EEPDRV ETHDRV FLSDRV FRDRV FLASHTST GPTDRV ICUDRV LINDRV MCUDRV PORTDRV PWMDRV RAMTST SPIDRV WDGDRV DRVEXT CANTRCV LINTRCV FRTRCV ETHTRCV

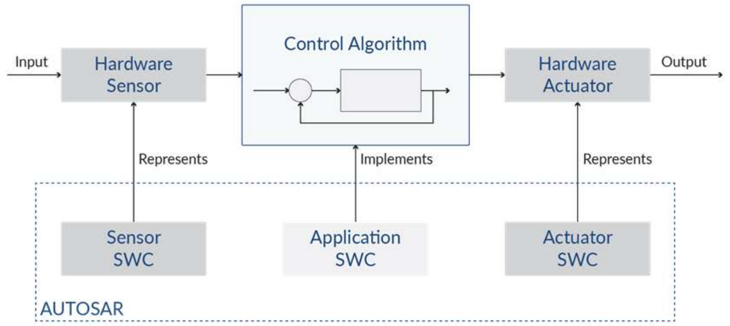
Microcontroller

Abstraction of location of I/O peripheral devices and layout.

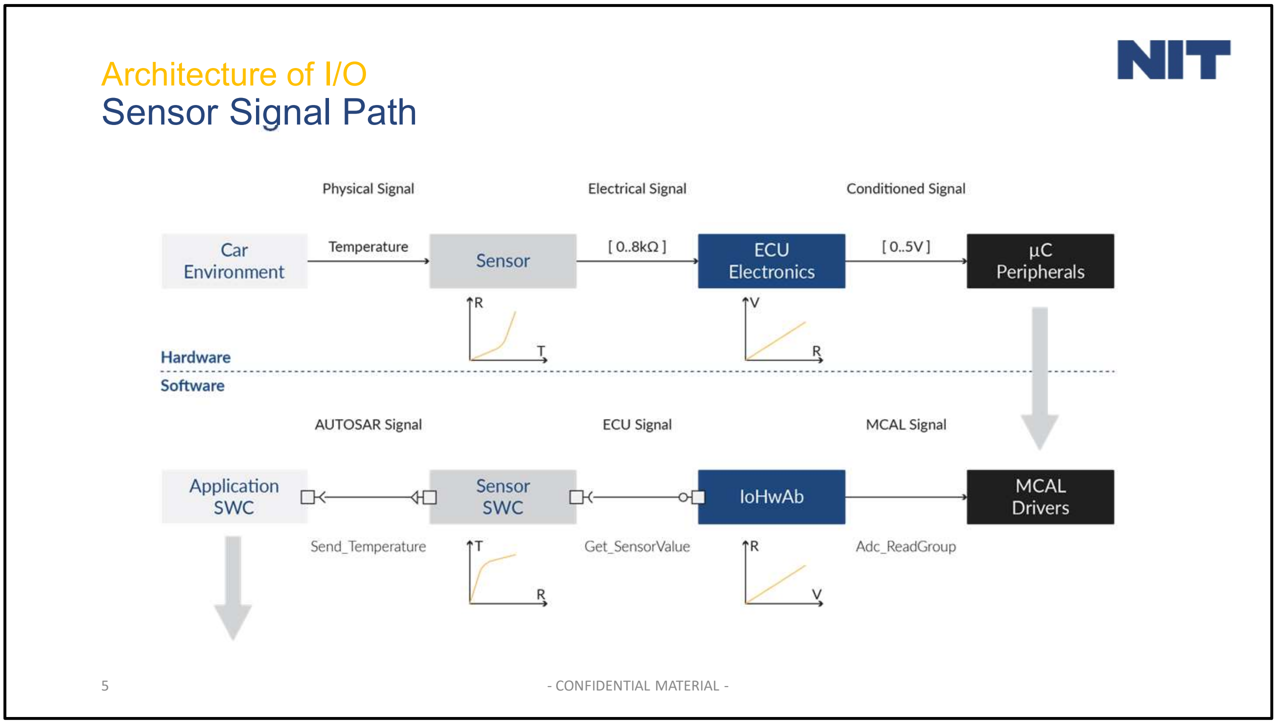
Inside Hardware I/O MCAL is implemented.
MCAL provides a standardized access to the HW of your microcontroller to the upper layers.
No signal state or pin state conversions are done by the MCAL.

Architecture of I/O

Application Level



A main element of automotive applications are control algorithms which perform a functions like fuel injection, temperature or light control. Control algorithms calculate output values based on input values. If the inputs and outputs are located on the same ECU as the control algorithm, then input/output peripherals of the ECU are used. Usually, sensor values are read and based on those values actuator is controlled. The following slides show how input/output control can be designed in AUTOSAR.



Architecture of I/O

Sensor Signal Path

Physical Signal

Electrical Signal

Conditioned Signal

Car Environment

Temperature

Sensor

[0..8kΩ]

ECU Electronics

[0..5V]

µC Peripherals

Hardware

Software

AUTOSAR Signal

ECU Signal

MCAL Signal

Application SWC

Sensor SWC

IoHwAb

MCAL Drivers

Send_Temperature

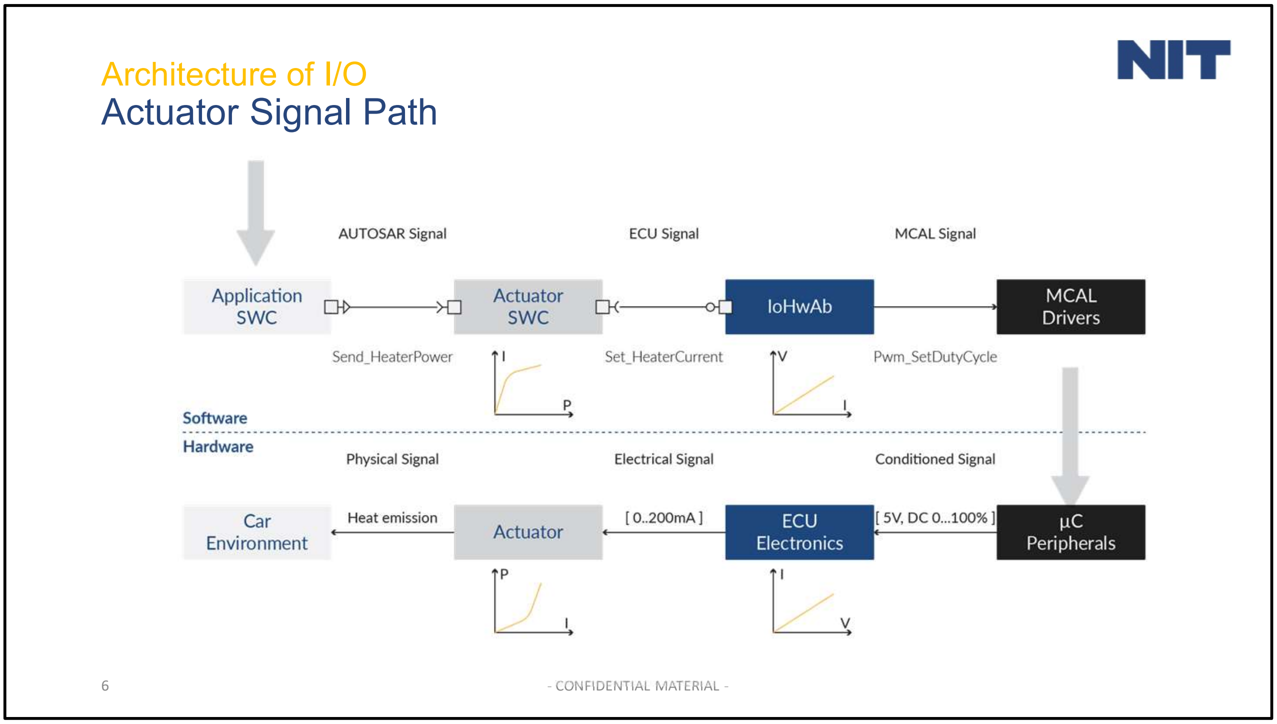
Get_SensorValue

Adc_ReadGroup

Sensor converts a physical or environmental signal into an electrical signal (e.g., temperature to resistance).
ECU Electronics converts sensor's electrical signals into an input signal suitable for the microcontroller (e.g., resistance to voltage).

uC Peripherals: Analog to Digital Converter (ADC), Digital Input Output (Dio), SPI…
MCAL Drivers implement peripheral drivers according to AUTOSAR (Dio, ADC, PWM, etc.).
I/O Hardware Abstraction converts from hardware-specific raw values into physical values (suitable for sensor SWC).

Sensor SWC is SW representation of hardware sensor. It converts electrical input values into application signals. Sensor SWC knows sensor hardware characteristics.
Application SWC implements control algorithm.



NIT

Architecture of I/O
Actuator Signal Path

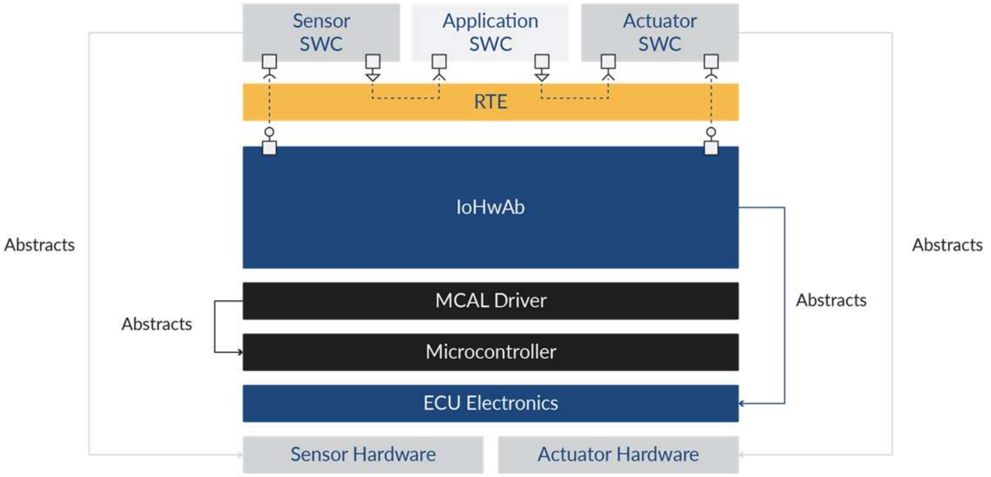
6

- CONFIDENTIAL MATERIAL -

Actuator SWC is software representation of hardware actuator. It converts application signals into electrical output values (suitable for physical actuator). Actuator SWC knows actuator HW characteristics.

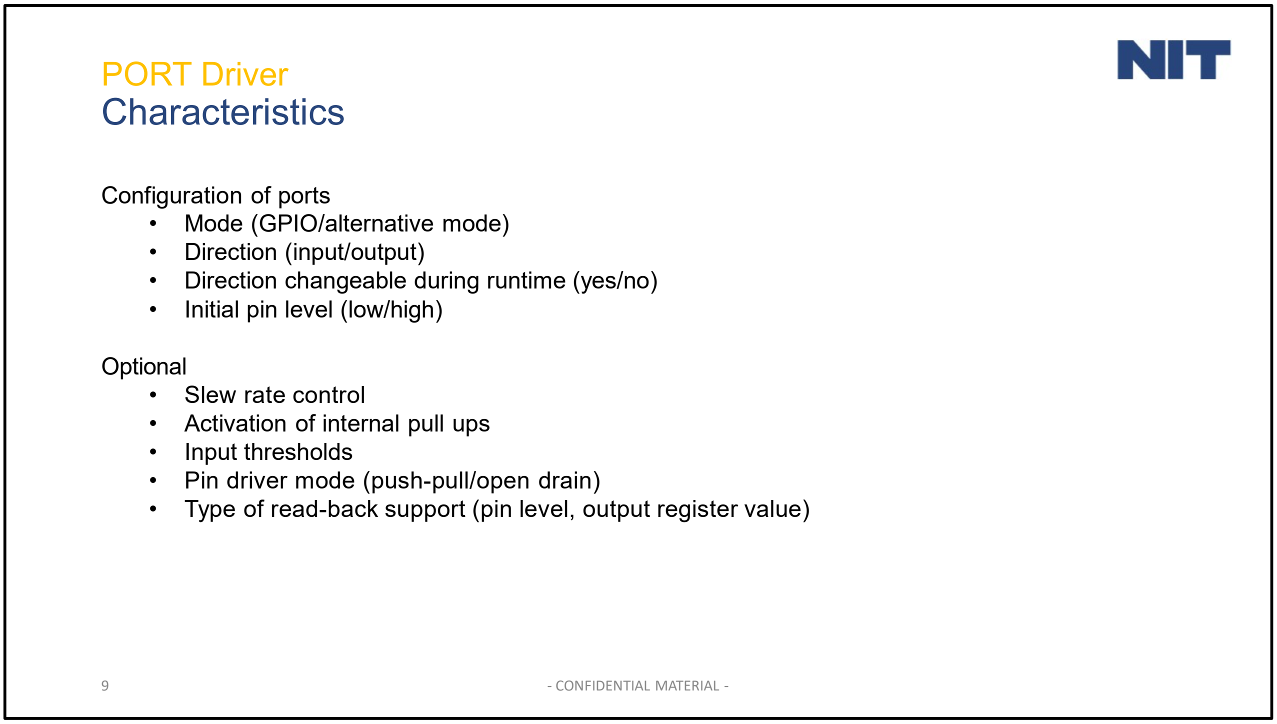
Actuator Converts an electrical signal into a physical or environmental signal.

Architecture of I/O
Abstraction Layers



Agenda

- Architecture of I/O
- **PORT Driver**
- DIO Driver
- I/O Hardware Abstraction
- Client Server Concept



PORT Driver
Characteristics

Configuration of ports

- Mode (GPIO/alternative mode)
- Direction (input/output)
- Direction changeable during runtime (yes/no)
- Initial pin level (low/high)

Optional

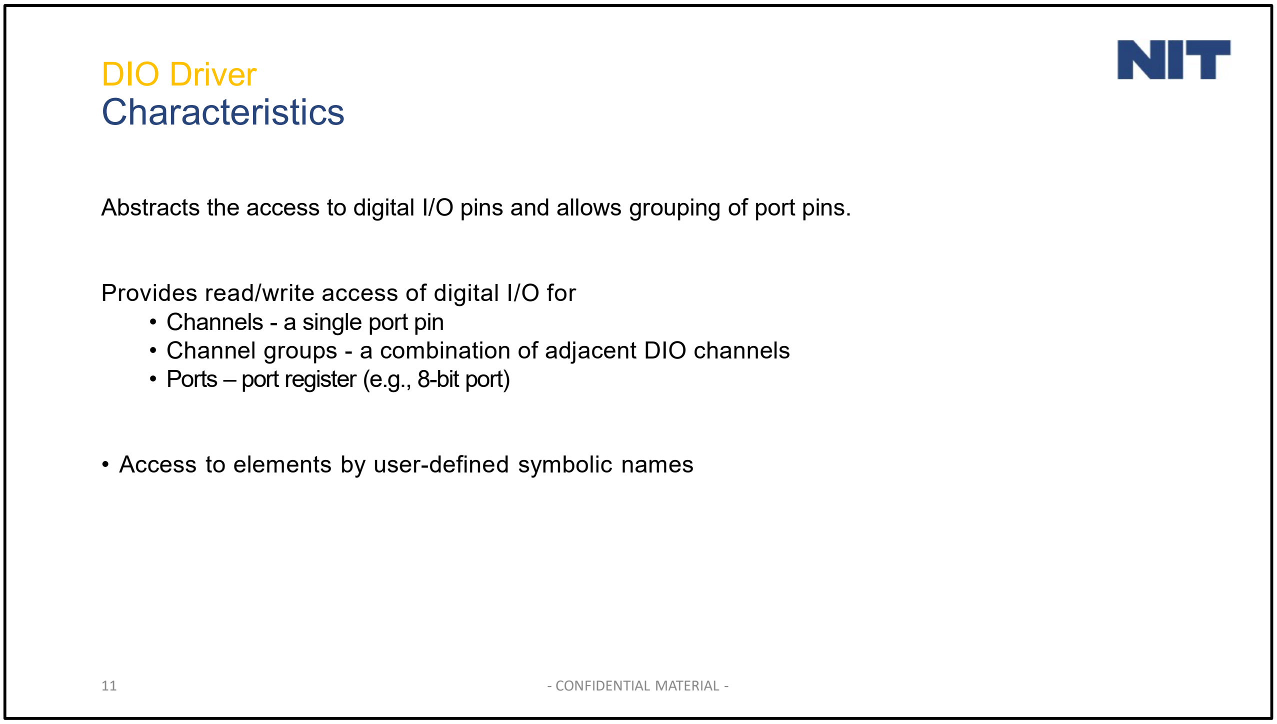
- Slew rate control
- Activation of internal pull ups
- Input thresholds
- Pin driver mode (push-pull/open drain)
- Type of read-back support (pin level, output register value)

PORT Driver APIs:

- `PortInit`: Initialize port data register of a specific port pin
- `PortSetPinDirection`: Enables setting of the port pin direction during runtime
- `PortRefreshPortDirection`: Refresh direction of non-dynamically configurable port pins
- `PortSetPinMode`: Change the pin mode (GPIO or alternative function) during runtime

Agenda

- Architecture of I/O
- PORT Driver
- **DIO Driver**
- I/O Hardware Abstraction
- Client Server Concept



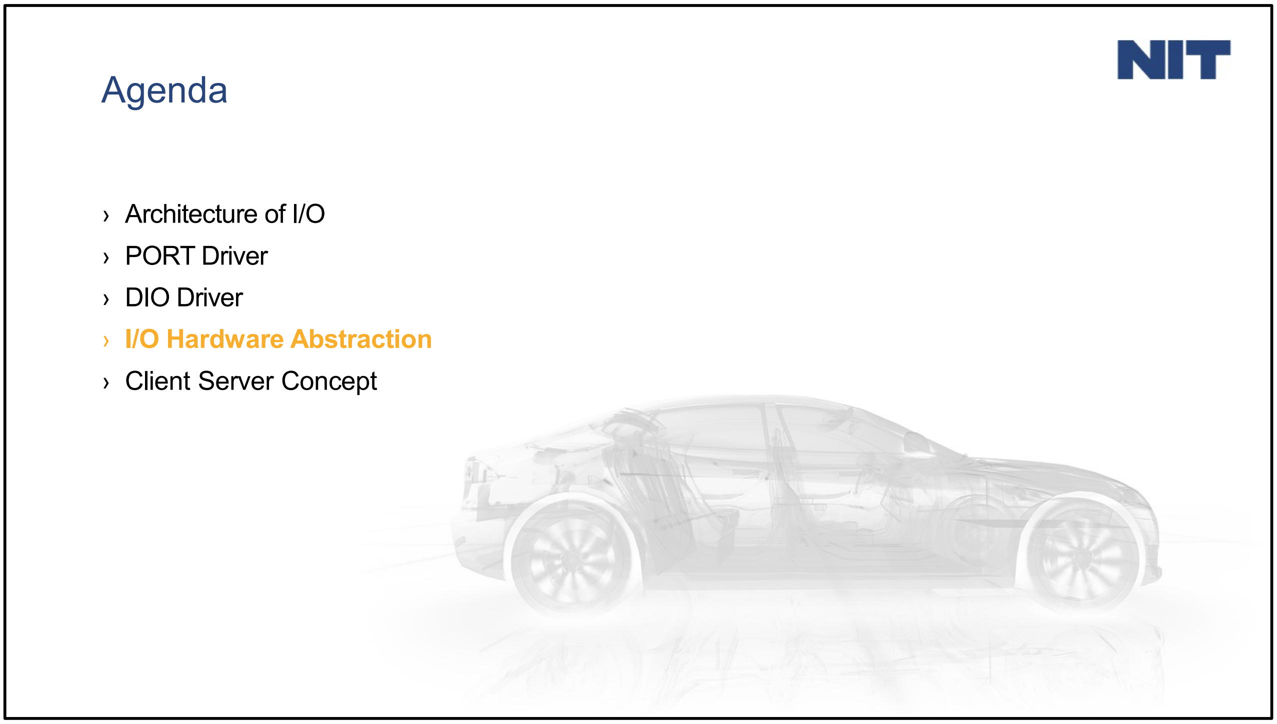
DIO Driver
Characteristics

Abstracts the access to digital I/O pins and allows grouping of port pins.

Provides read/write access of digital I/O for
- Channels - a single port pin
- Channel groups - a combination of adjacent DIO channels
- Ports – port register (e.g., 8-bit port)

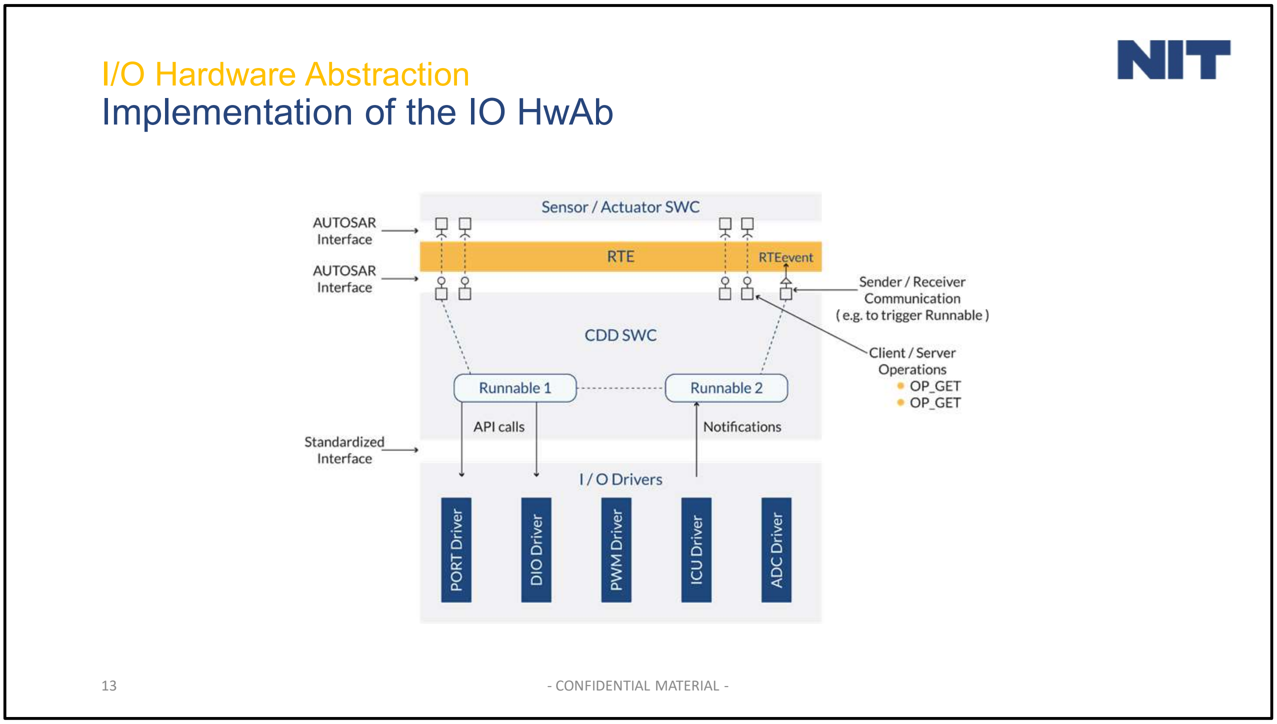
- Access to elements by user-defined symbolic names

DIO Driver APIs:
- `Dio_ReadPort`: Reading of a complete port register
- `Dio_WritePort`: Writing of a complete port register
- `Dio_ReadChannelGroup`: Reading of a channel group
- `Dio_WriteChannelGroup`: Writing of a channel group
- `Dio_ReadChannel:` Reading of a single channel (Port-Pin)
- `Dio_WriteChannel`: Writing of a single channel (Port-Pin)
- `Dio_FlipChannel:` Reading of an output channel, inversion and writing of this value



Agenda

- Architecture of I/O
- PORT Driver
- DIO Driver
- **I/O Hardware Abstraction**
- Client Server Concept



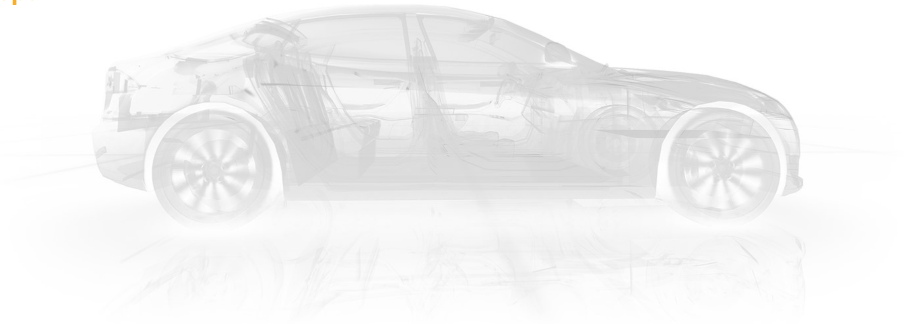
I/O Hardware Abstraction
Implementation of the IO HwAb

Sensor/Actuator SWCs communicate with IOHwAb via Client/Server ports.
Server Runnables of IOHwAb are calling APIs of correct drivers.

Example:
Sensor SWC Door wants to read value from door sensor.
Door SWC will call `Rte_Call_DoorStatus_ReadChannel` which will trigger `DoorStatus_ReadChannel`, server runnable of IOHwAb.
`DoorStatus_ReadChannel` will call correct driver API to get required value (i.e., `Dio_ReadChannel`) and return it to Door SWC.

Agenda

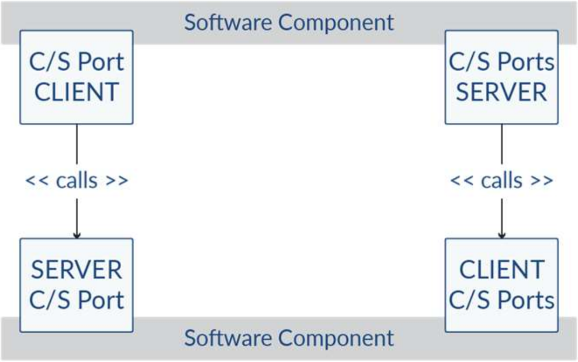
- Architecture of I/O
- PORT Driver
- DIO Driver
- I/O Hardware Abstraction
- **Client Server Concept**



Client Server Concept
Client and Server

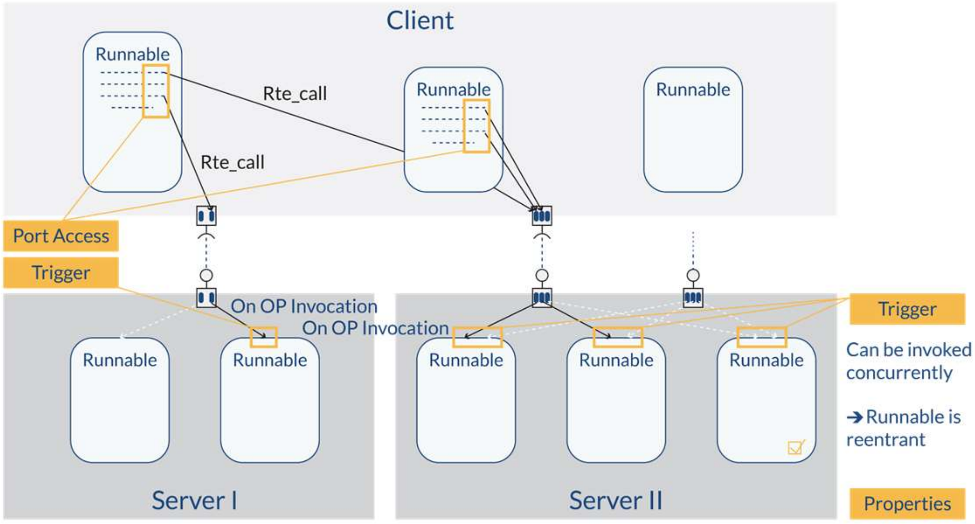
Client
Uses operations of the server.

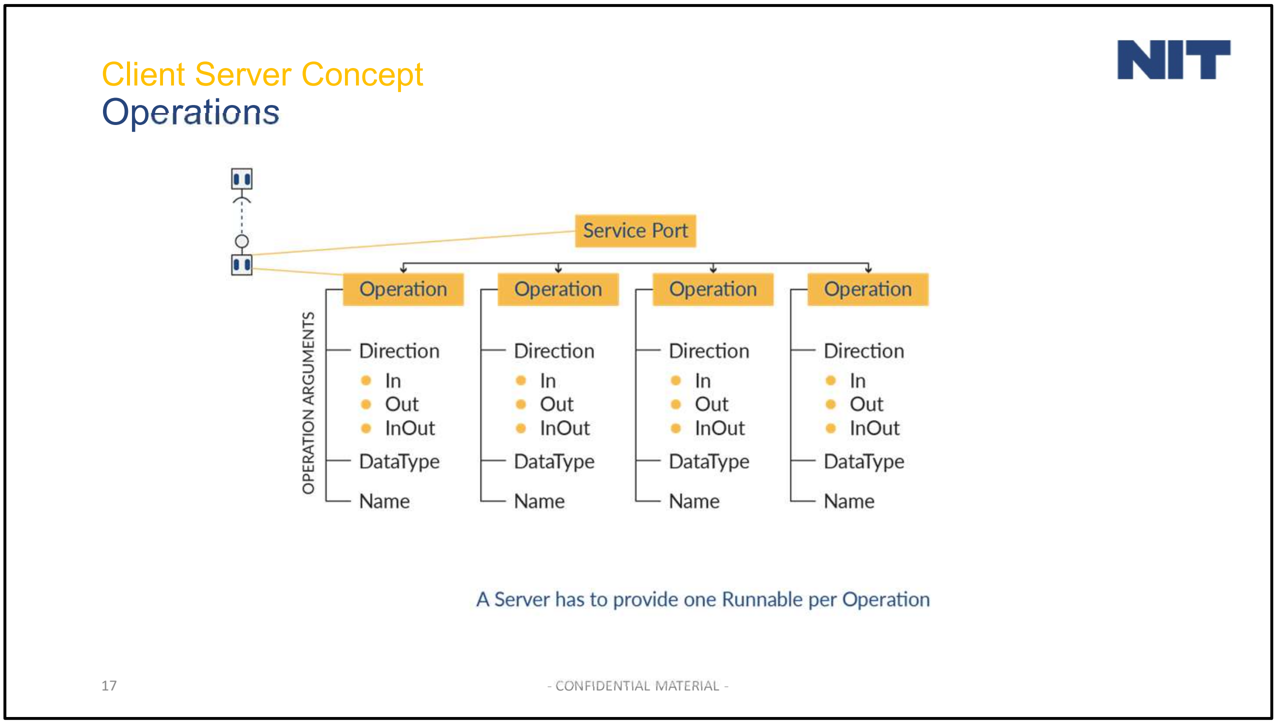
Server
When a Software Component is a server it has to provide all operations (Runnables) the client requests.



Client Server Concept

Call of Server Ports





Client Server Concept

Operations

Service Port

OPERATION ARGUMENTS

Operation
- Direction
  - In
  - Out
  - InOut
- DataType
- Name

Operation
- Direction
  - In
  - Out
  - InOut
- DataType
- Name

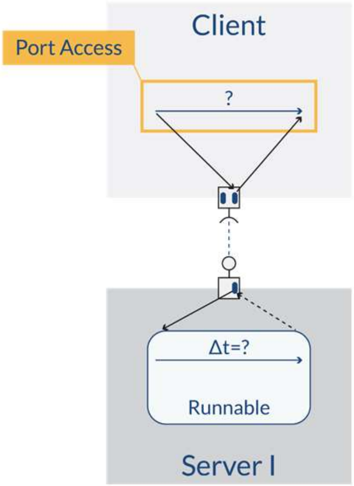
Operation
- Direction
  - In
  - Out
  - InOut
- DataType
- Name

Operation
- Direction
  - In
  - Out
  - InOut
- DataType
- Name

A Server has to provide one Runnable per Operation

Client Server Concept

Overview: Runtime of Servers



Server runnables can be triggered:

- Synchronous with/without Timeout
- Asynchronous and Waiting
- Asynchronous and Polling
- Asynchronous and trigger on Runnable

Client Server Concept

Synchronous with/without Timeout

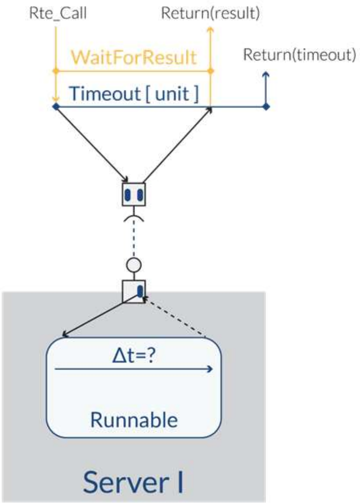
Synchronous → Timeout [unit]

The Rte Call

- Triggers the timeout monitoring
- Waits for the result

The return value contains:

- The result of the server when return occurs before the timeout expires
- A timeout error code as the timeout expires before the result of the server is ready



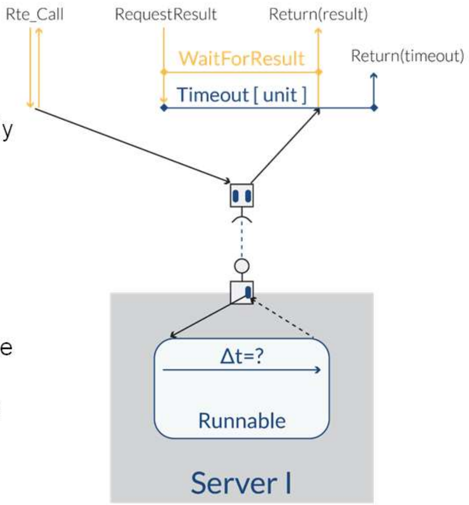
Client Server Concept
Asynchronous and Waiting

Asynchronous → Waiting → Timeout [unit]

The Rte_Call
- Triggers the server Runnable and returns immediately

The Rte_Result
- Triggers the timeout monitoring
- Waits for the result

The return value contains:
- The result of the server when return occurs before the timeout expires
- A timeout error code if the timeout expires before the result of the server is ready



Client Server Concept
Asynchronous and Polling

Asynchronous → Polling

The Rte_Call

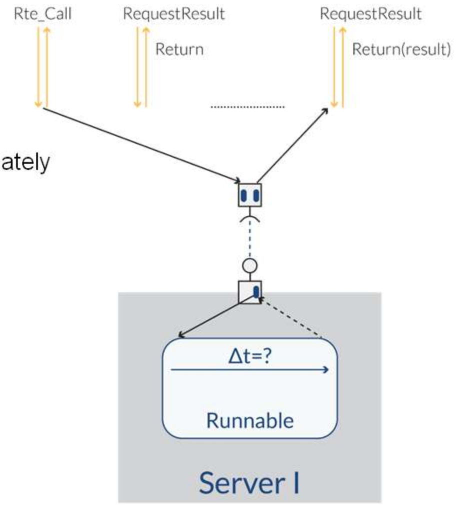
- Triggers the server Runnable and returns immediately

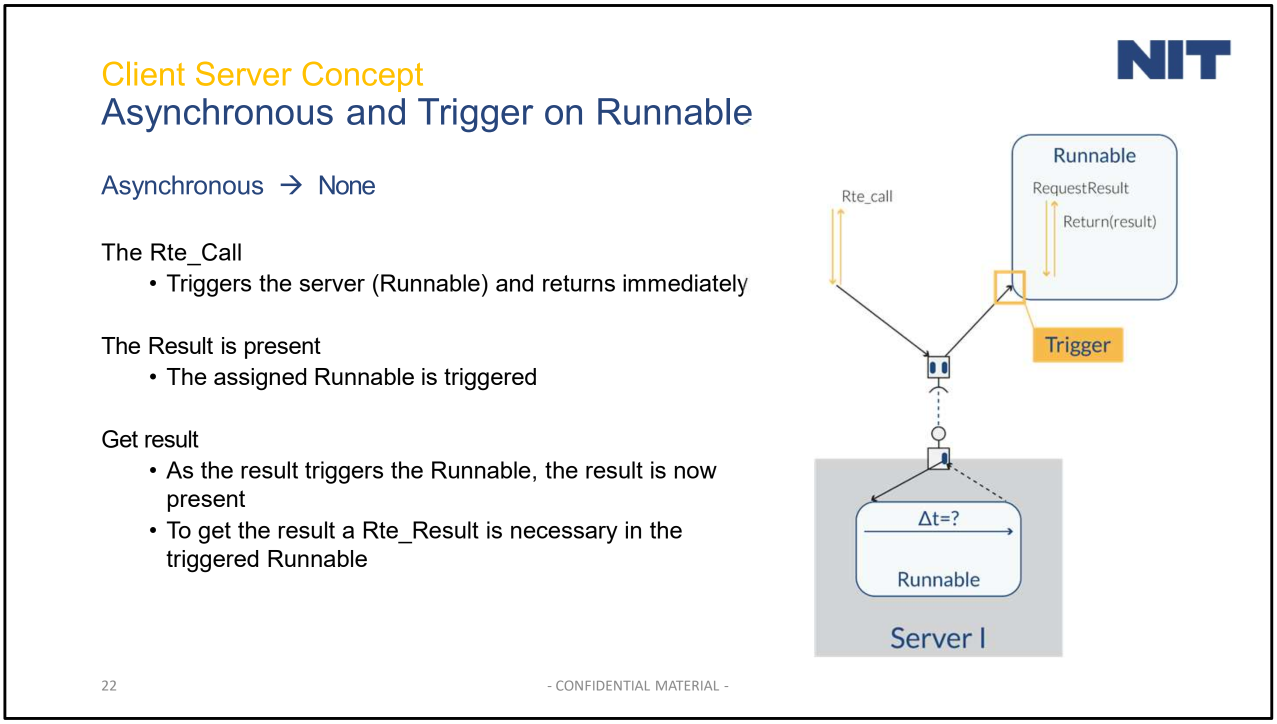
The Rte_Result

- Sees if the result is already there

The return value contains:

- The result of the server if present
- Information that result is not ready yet





Client Server Concept
Asynchronous and Trigger on Runnable

Asynchronous → None

The Rte_Call

- Triggers the server (Runnable) and returns immediately

The Result is present

- The assigned Runnable is triggered

Get result

- As the result triggers the Runnable, the result is now present
- To get the result a Rte_Result is necessary in the triggered Runnable

Runnable
RequestResult
Return(result)
Rte_call
Trigger
Δt=?
Runnable
Server I

Rte_Call sets an Event for the Server Runnable.

When the Server Runnable has finished, another Event will be set to notify the configured client Runnable.

Everything should be configured already through the RTE Runnable Triggers