

Port Designs and Implementation Approaches


Content

	PORT
Content	1
1 Content.....	3
2 Port Design.....	4
2.1 Basic Concept	4
2.1.1 From Interface to Port	4
2.1.2 Facts	5
2.2 Unidirectional Single Port.....	6
2.2.1 Single Interface	6
2.2.2 Multiple Interfaces	7
2.3 Bidirectional Single Port.....	8
2.3.1 Single Interface per Direction	8
2.3.2 Multiple Interfaces per Direction	9
2.4 Multiple Ports.....	10
2.4.1 Single Ended.....	10
2.4.2 Multiple Ended	11
2.5 Split Ports	12
2.5.1 Single Ended.....	12
2.5.2 Multiple Ended	13
2.6 Nested Ports.....	14
2.6.1 Single Ended.....	14
2.6.2 Multiple Ended	15
3 Port Implementation.....	16
3.1 Approaches	16
3.2 Basic Concept	17

3.2.1	Overview	17
3.2.2	Accessor, Interface, Port and Realization: Path A.....	18
3.2.3	Accessor, Interface, Port and Realization: Path B.....	19
3.2.4	Interface Class: icX.hpp	20
3.2.5	Bound Class: cOutBound.hpp.....	21
3.2.6	Bound Class: cOutBound.cpp.....	22
3.2.7	Port Class: pcA.hpp	23
3.2.8	Port Class: pcA.cpp	24
3.2.9	Accessor Class: cA.hpp	25
3.2.10	Accessor Class: cA.cpp	26
3.2.11	Bound Class: cInBound.hpp	27
3.2.12	Bound Class: cInBound.cpp.....	28
3.2.13	Port Class: pcB.hpp	29
3.2.14	Port Class: pcB.cpp	30
3.2.15	Realization Class: cB.hpp	31
3.2.16	Realization Class: cB.cpp	32
3.2.17	Main_Application: Instantiation and Initialization	33
3.2.18	Main_Application: Execution.....	34
3.2.19	Main_Application: Execution.....	35
3.3	Controller and Counter	36
3.3.1	Overview	36
3.3.2	Controller Path.....	37
3.3.3	Counter Path	38
3.4	Additional Aspects.....	39
4	Summary and Outlook.....	40
5	MicroConsult Training and Coaching.....	41

1 Content

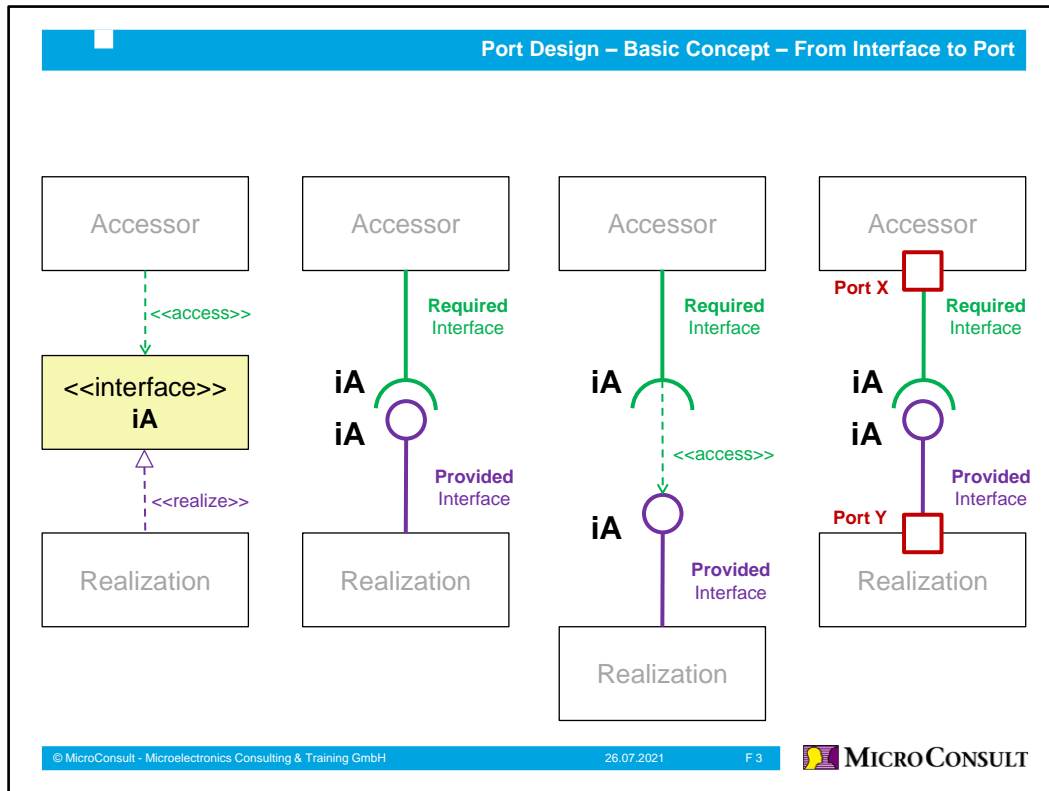
Content
▪ Port Concept Introduction
▪ Port Design: Unidirectional, Bidirectional
▪ Port Design: Multiple, Split and Nested
▪ Port Implementation: Approaches
▪ Port Implementation: Examples
▪ Port Implementation: Additional Aspects
▪ Summary and Outlook

© MicroConsult - Microelectronics Consulting & Training GmbH 26.07.2021 F 2 

2 Port Design

2.1 Basic Concept

2.1.1 From Interface to Port



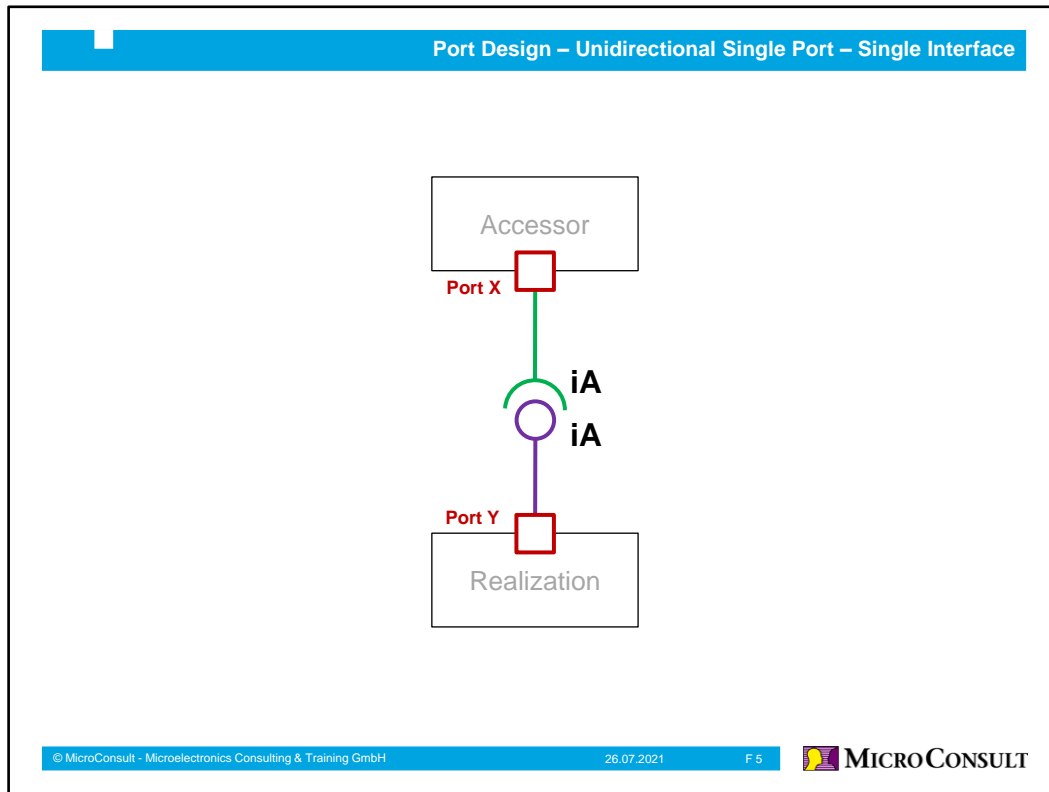
2.1.2 Facts

Port Design – Basic Concept – Facts

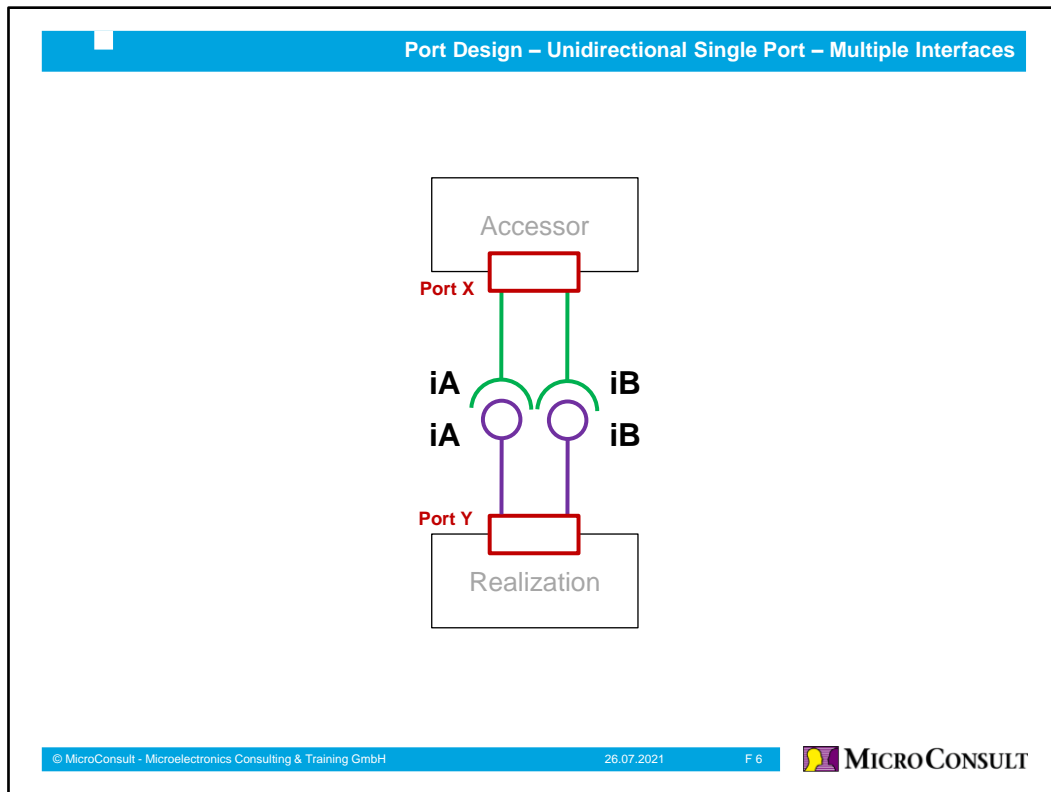
- Functions are logically grouped to **interfaces**.
- Interfaces are **logically grouped** to **ports**.
- A port consists of
 - zero to infinite **provided** and / or
 - zero to infinite **required** interfaces
- UML Unified Modeling Language based; one or more ports can be added to a **class** or **component** but not to a package.
- Two ports are **interconnectable** if all provided and required interfaces are **compatible**.

2.2 Unidirectional Single Port

2.2.1 Single Interface

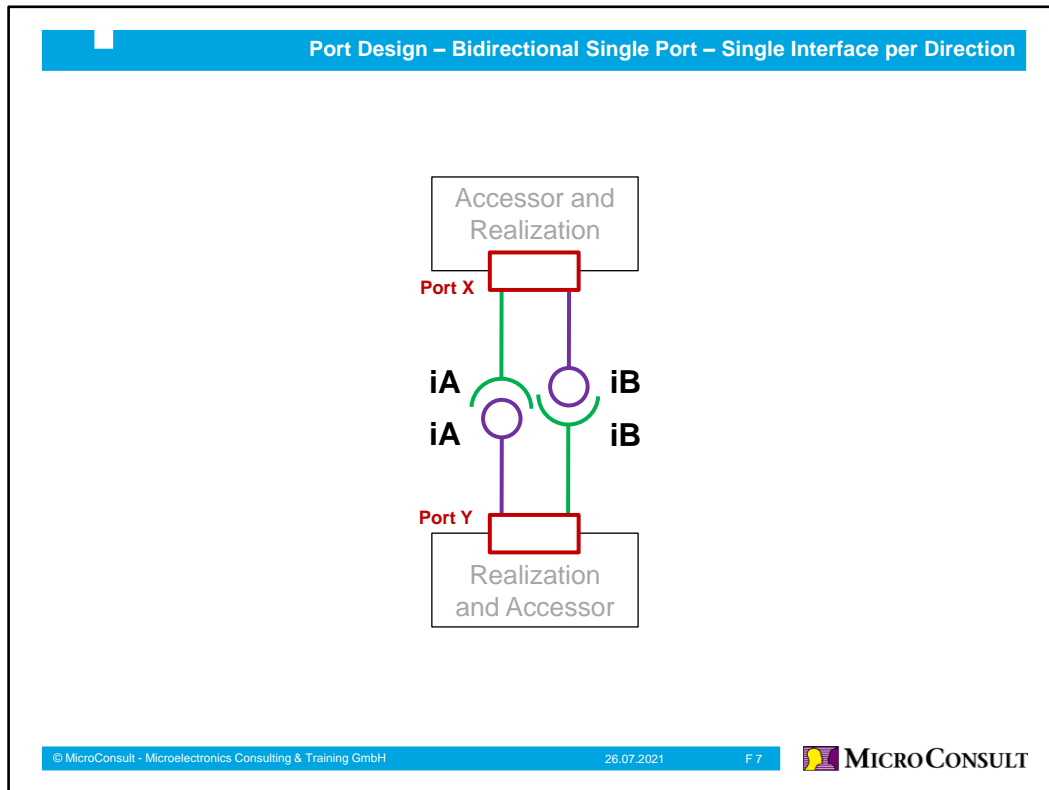


2.2.2 Multiple Interfaces

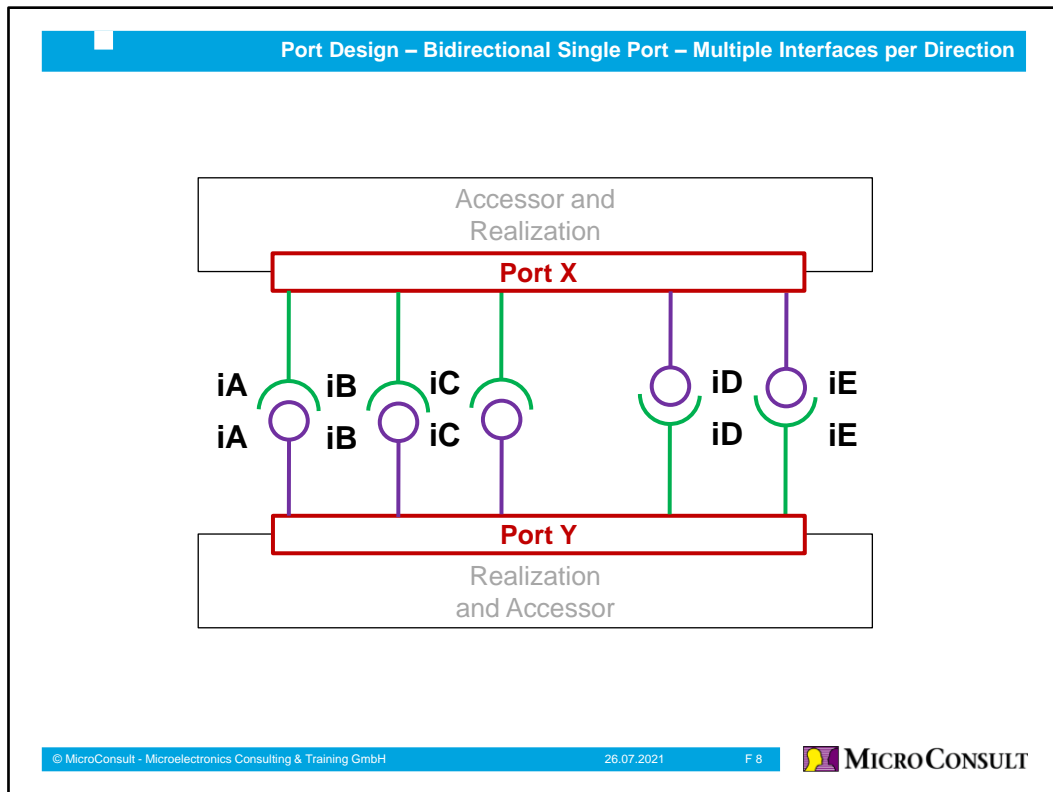


2.3 Bidirectional Single Port

2.3.1 Single Interface per Direction

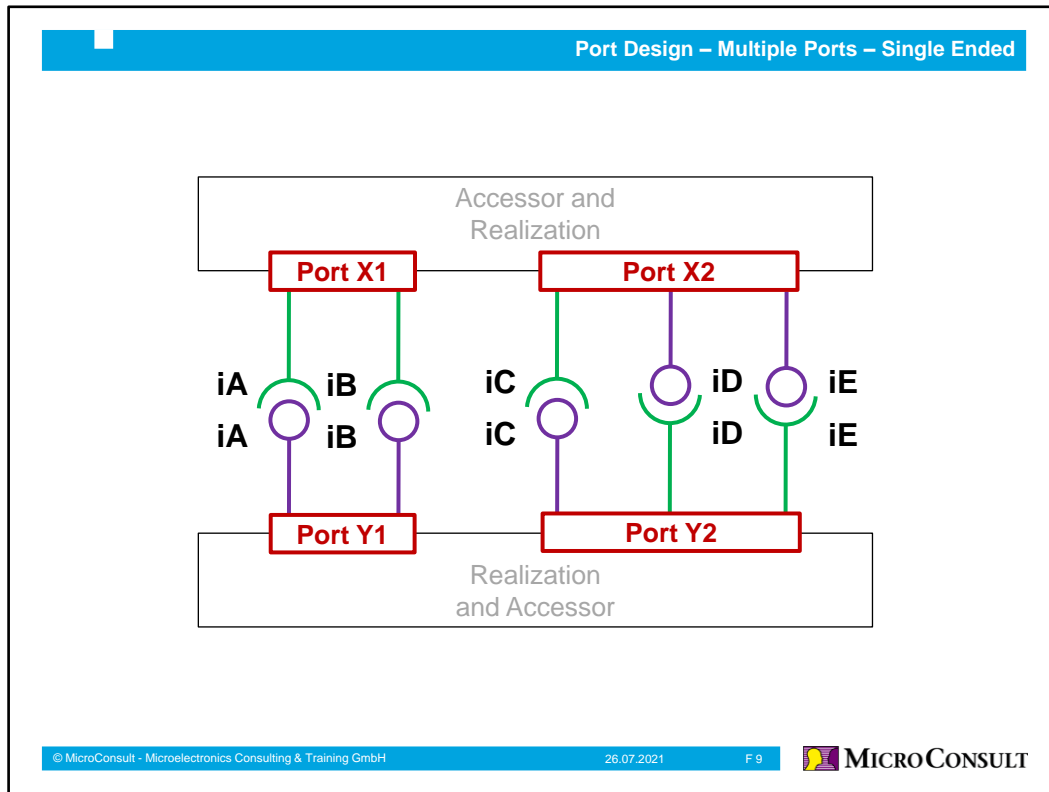


2.3.2 Multiple Interfaces per Direction

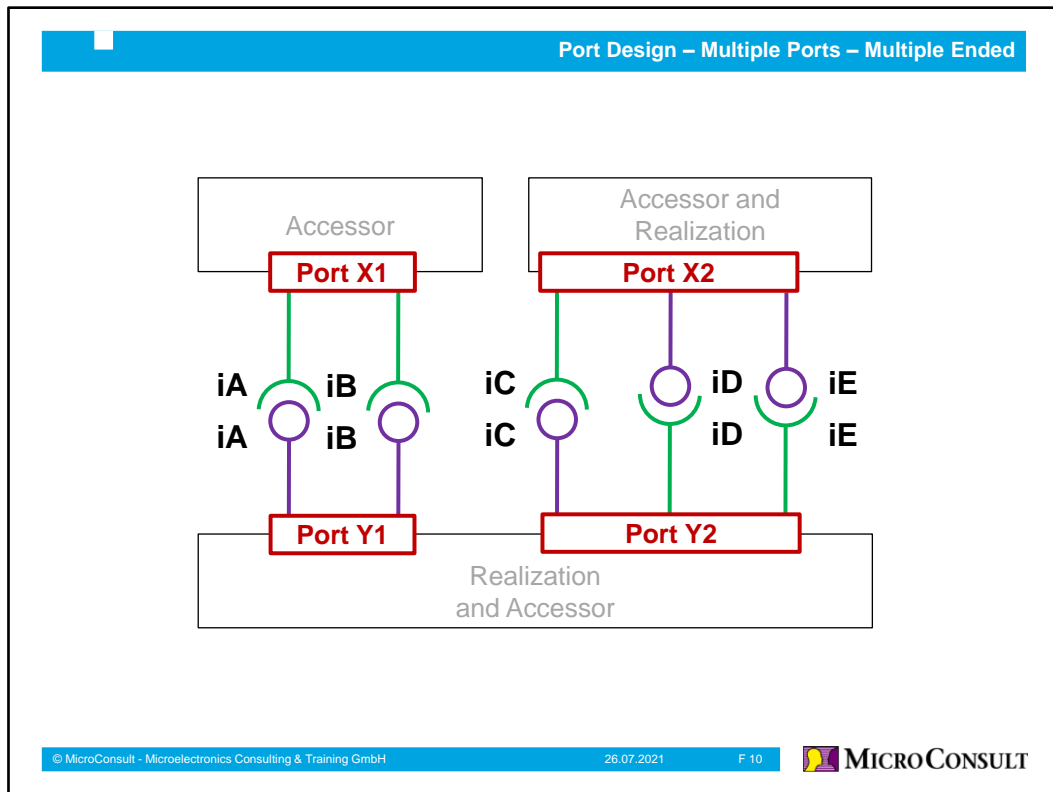


2.4 Multiple Ports

2.4.1 Single Ended

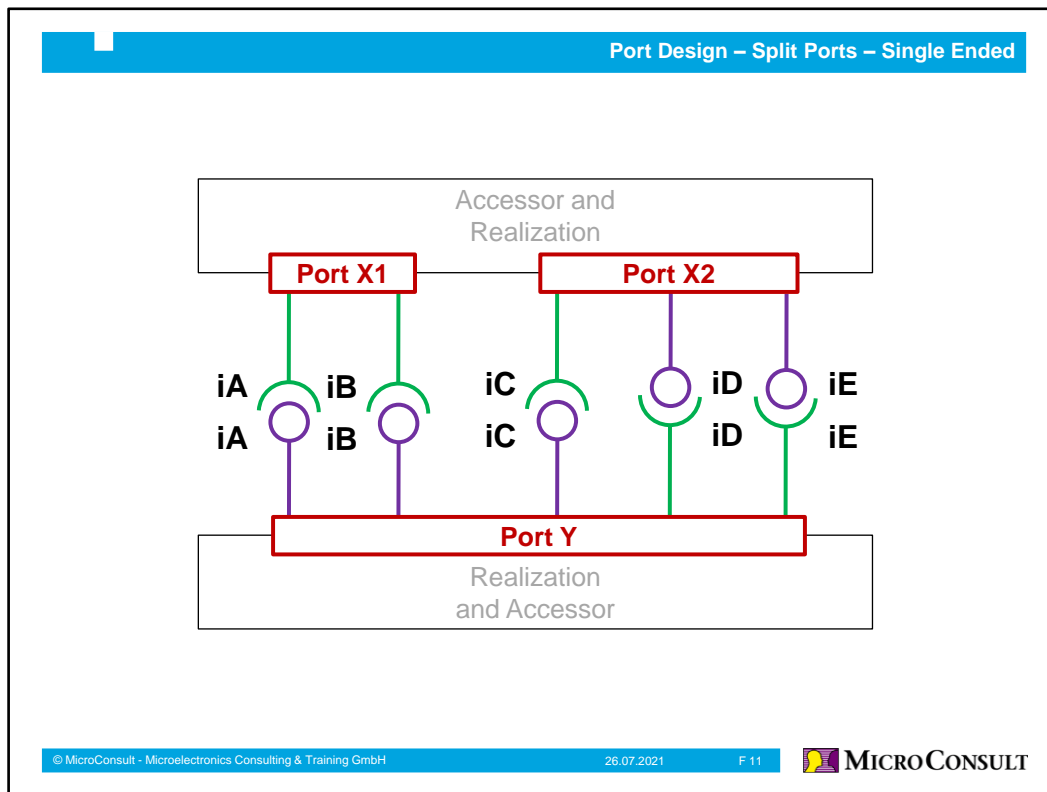


2.4.2 Multiple Ended

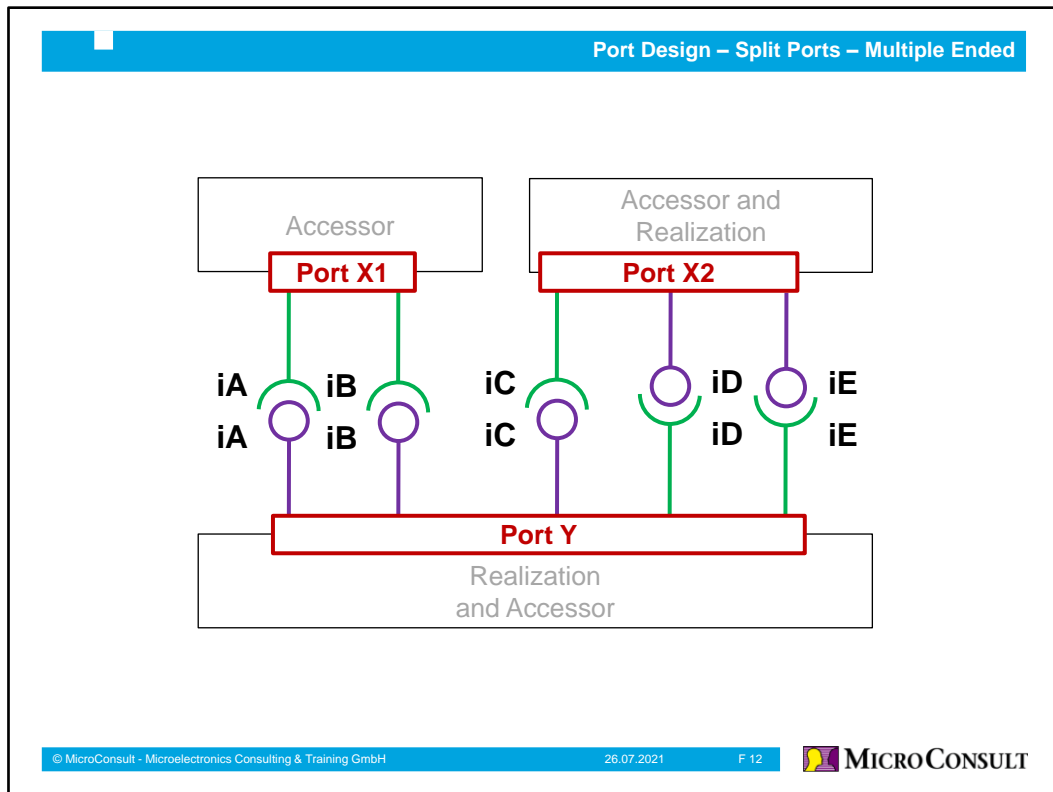


2.5 Split Ports

2.5.1 Single Ended

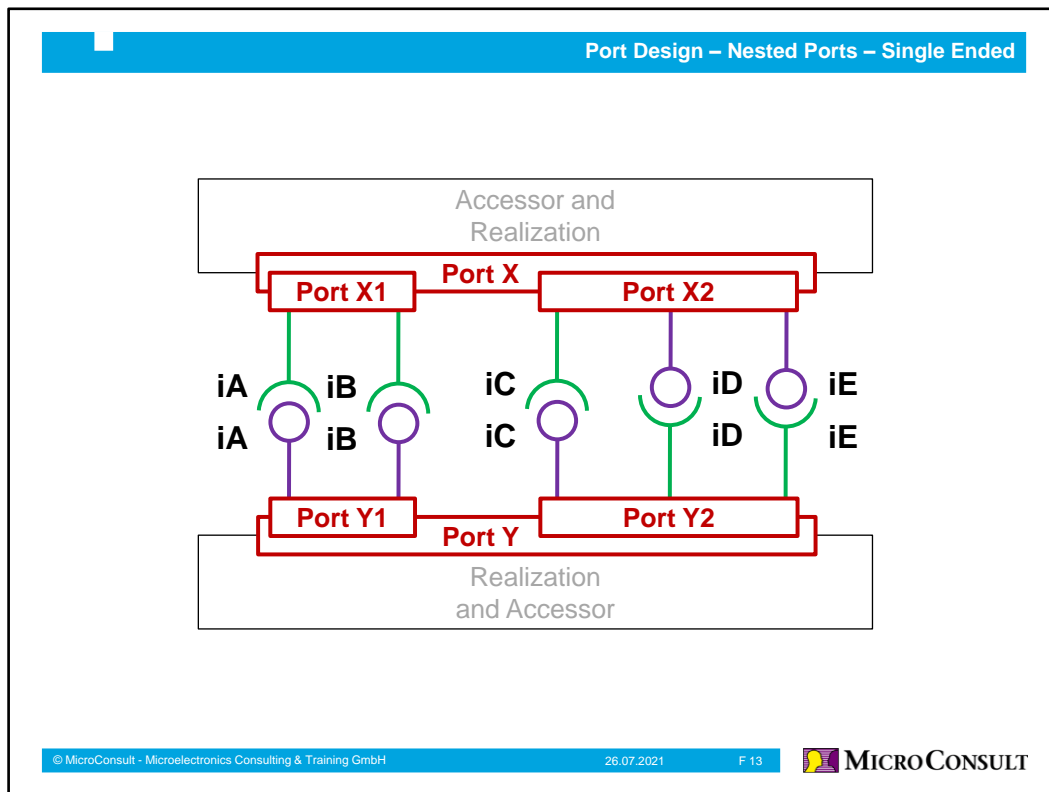


2.5.2 Multiple Ended

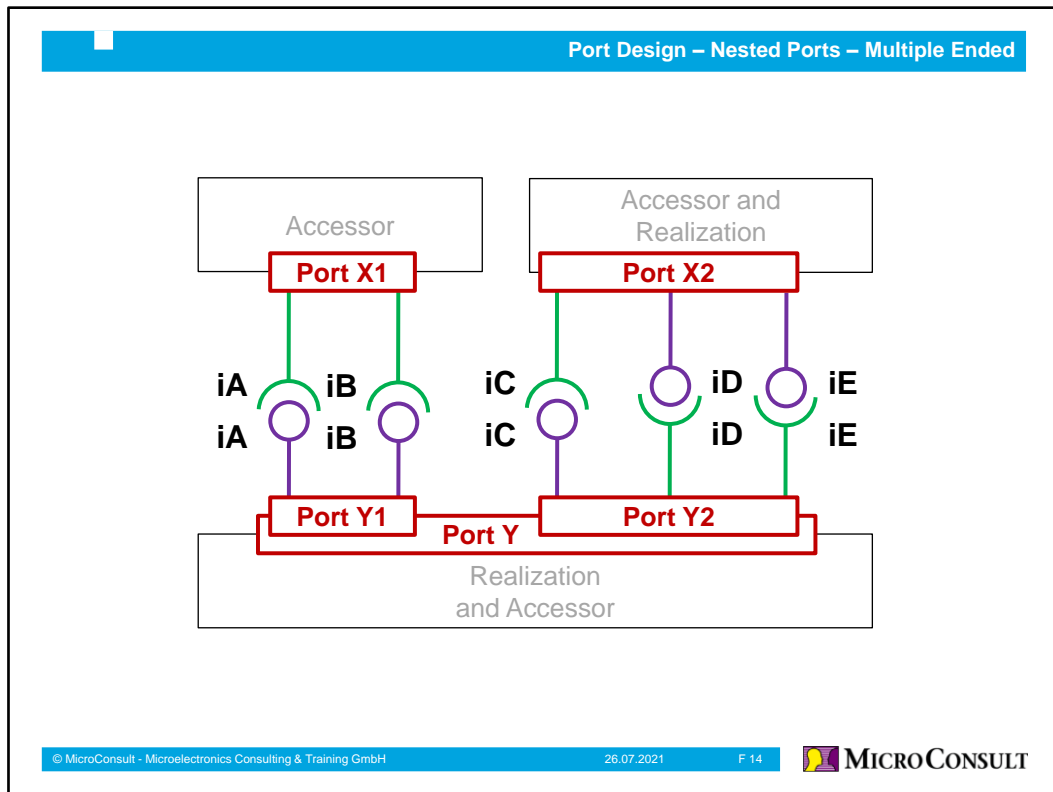


2.6 Nested Ports

2.6.1 Single Ended



2.6.2 Multiple Ended



3 Port Implementation

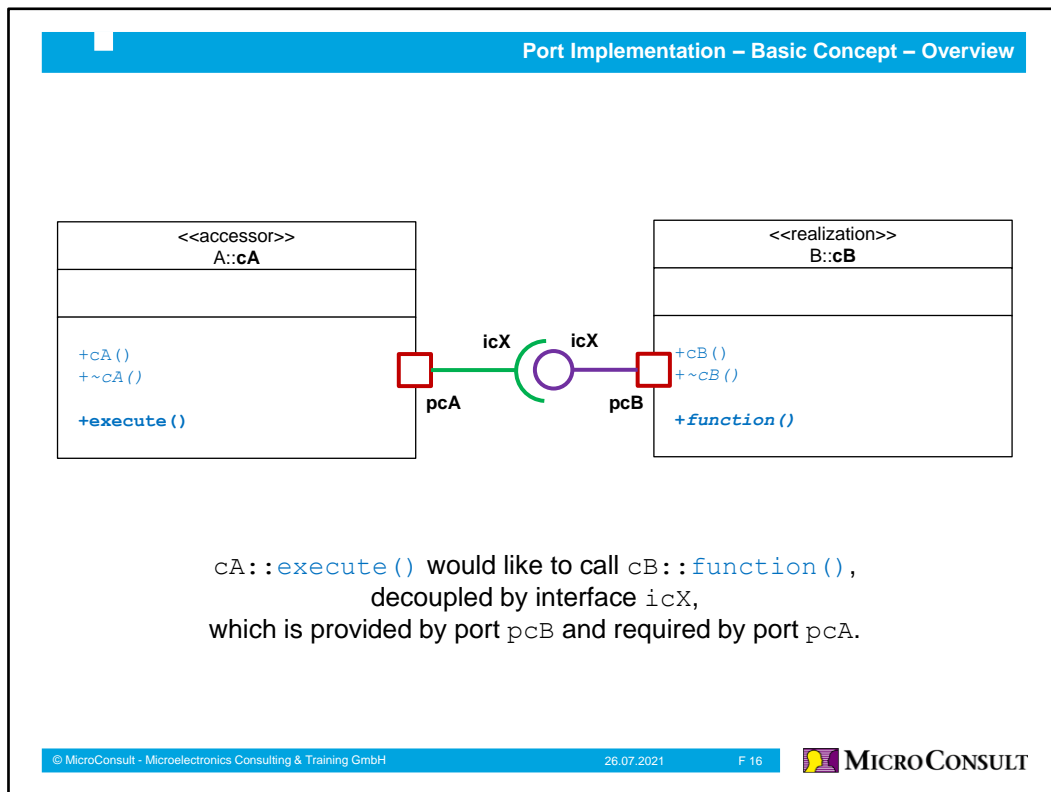
3.1 Approaches

Port Implementation – Approaches

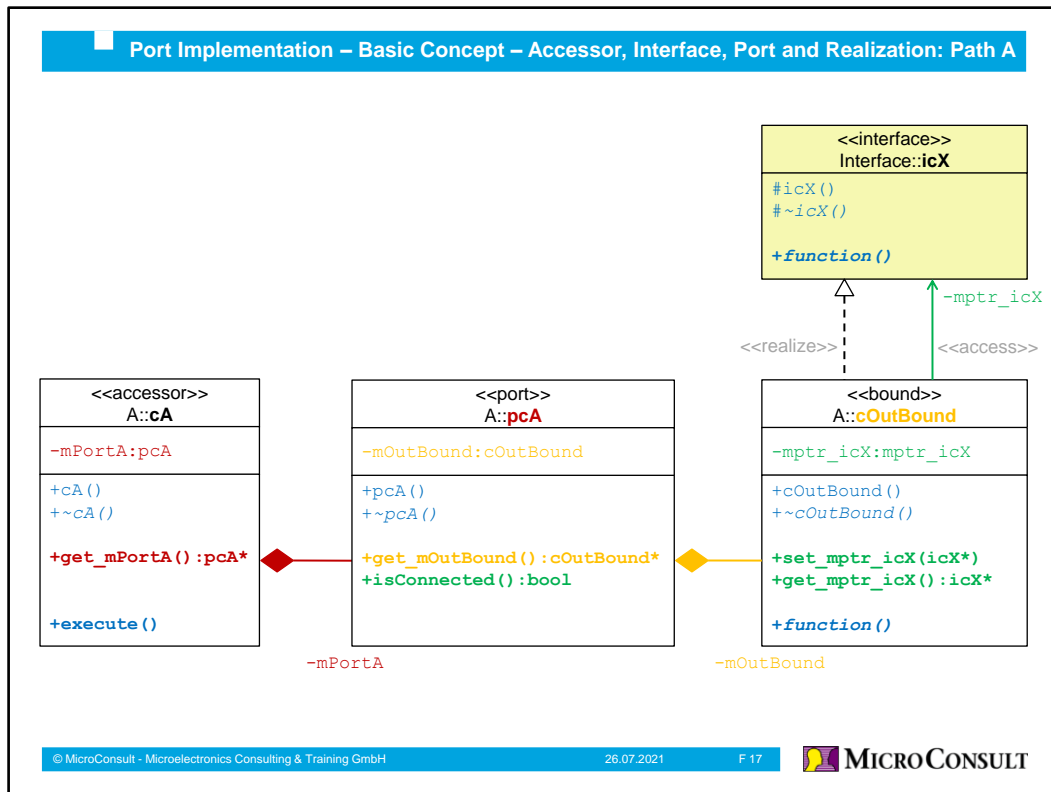
- Provided and required **interfaces** are **classes** with **pure virtual functions**.
- All provided interfaces are realized and accessed in an **input bound class**.
- All required interfaces are realized and accessed in an **output bound class**.
- The corresponding **input** and **output bound classes** are **instantiated** in the **port class**.
- The **port class** is **instantiated** in the class to which the port belongs.
- In order to operate, the **ports** need to be **connected** by **pointer**.

3.2 Basic Concept

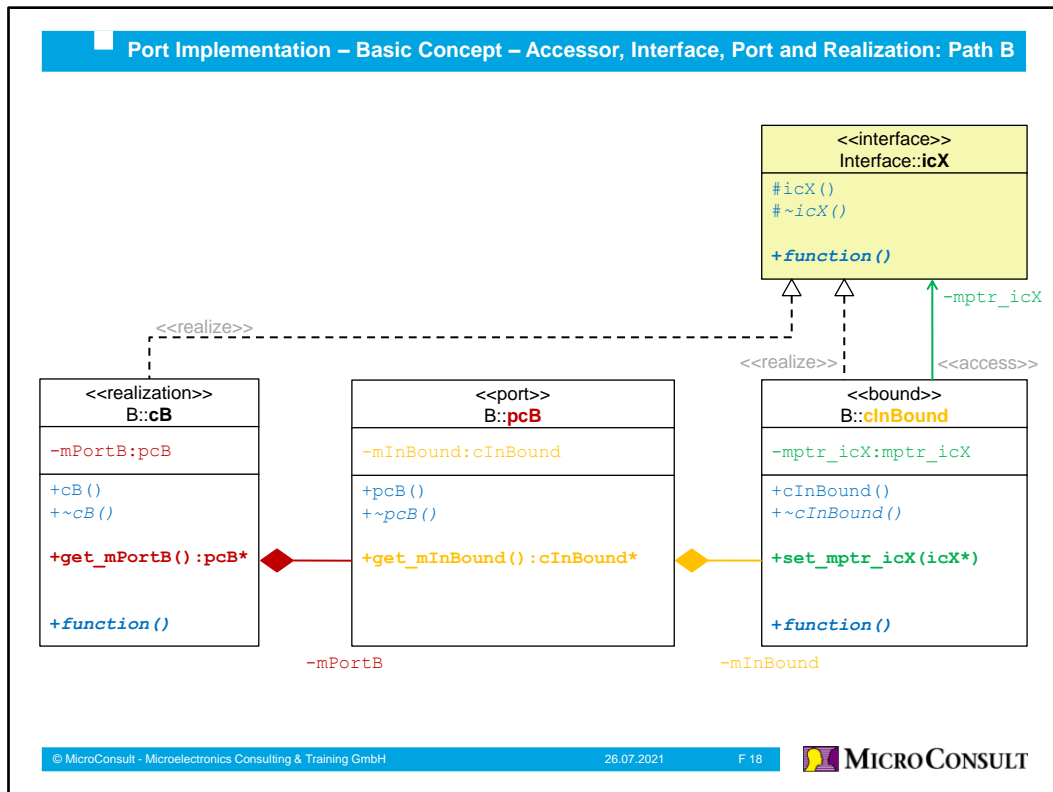
3.2.1 Overview



3.2.2 Accessor, Interface, Port and Realization: Path A



3.2.3 Accessor, Interface, Port and Realization: Path B



3.2.4 Interface Class: icX.hpp

Port Implementation – Basic Concept – Interface Class: icX.hpp

```
#ifndef __icX_HPP__
#define __icX_HPP__

namespace Interface
{
    class icX
    {
    public:
        virtual void function(void) = 0;

    protected:
        icX() = default;
        virtual ~icX() = default;
    };
}

#endif // __icX_HPP__
```

<<interface>> Interface::icX
#icX() #~icX()
+function()

Interface function to be
accessed by class cA and
implemented by class cB

© MicroConsult - Microelectronics Consulting & Training GmbH 26.07.2021 F 19 MICROCONSULT

3.2.5 Bound Class: cOutBound.hpp

Port Implementation – Basic Concept – Bound Class: cOutBound.hpp

```

#ifndef __cOutBound_HPP__
#define __cOutBound_HPP__

#include "../Interface/icX.hpp"
using Interface::icX;

namespace A
{
    class cOutBound : public icX
    {
    public:
        cOutBound();
        virtual ~cOutBound() = default;

        void function(void) override;
        void set_mptr_icX(icX* const ptr_icX);
        icX* get_mptr_icX(void) const;

    private:
        icX* mptr_icX;
    };
}

#endif // __cOutBound_HPP__

```

icX is a required interface for cA, so it is implemented in class cOutBound.

cOutBound contains a pointer of the required interface type icX. This pointer has to be connected to port cB, which provides the interface icX.

© MicroConsult - Microelectronics Consulting & Training GmbH 26.07.2021 F 20 MICROCONSULT

Port Implementation – Basic Concept – Bound Class: cOutBound.cpp

```
#include "cOutBound.hpp,,

#include "..\..\Plattform\HAL\BasicIO.hpp"
using namespace Plattform::HAL;

namespace A
{
    cOutBound::cOutBound() : mpPtr_icX(nullptr)
    { /* no implementation */ }

    void cOutBound::function(void)
    {
        showValue("\ncOutBound::function() is called");
        if (mpPtr_icX != nullptr)
        {
            mpPtr_icX->function();
        }
    }

    void cOutBound::set_mpPtr_icX(icX* const ptr_icX)
    {
        mpPtr_icX = ptr_icX;
    }

    icX* cOutBound::get_mpPtr_icX(void) const
    {
        return const_cast<icX*>(mpPtr_icX);
    }
}
```

The UML class diagram illustrates the following components and relationships:

- Interface ICX**: Defines `+ICX()`, `+I/O()`, and `+Function()`.
- A:CA** (Base Class): Implements `+ICX()` as `-return CA::pCA()`. It has a `+get_mpPtr_icX() const` method.
- A:pCA** (Derived Class): Inherits from **A:CA**. It implements `+ICX()` as `-return CA::cOutBound()` and `+I/O()` as `-return CA::cOutBound().I/O()`. It also has a `+set_mpPtr_icX(const icX*)` method.
- A:cOutBound** (Bound Class): Inherits from **A:pCA**. It implements `+Function()` as `-if (mpPtr_icX != nullptr) mpPtr_icX->Function();`. It has a `+set_mpPtr_icX(icX* const ptr_icX)` method.

Relationships are shown with solid arrows for inheritance and dashed arrows for associations. The **A:cOutBound** class is highlighted with a red box.

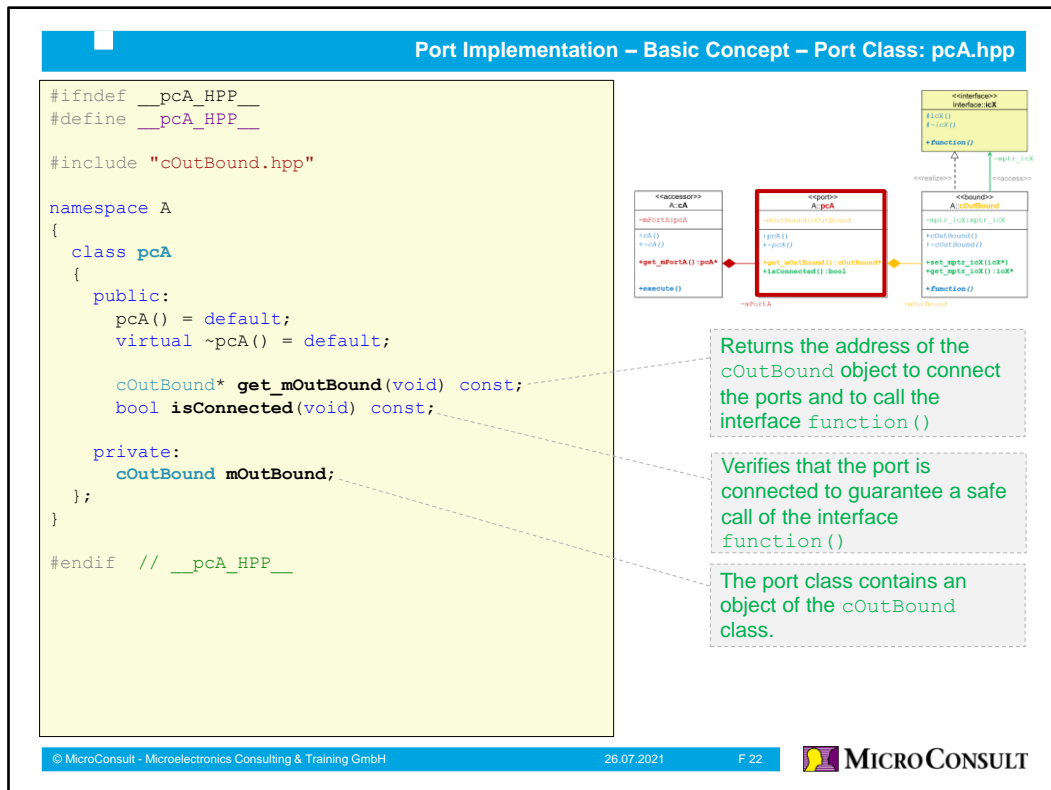
The interface pointer is initialized by a default value.

In the implementation of the interface function, the function itself calls the realization of path B.

The interface pointer has to be connected to the real function() implementation in class cB.

Required to verify successful port connection

3.2.7 Port Class: pcA.hpp



3.2.8 Port Class: pcA.cpp

Port Implementation – Basic Concept – Port Class: pcA.cpp

```
#include "pcA.hpp"

namespace A
{
    cOutBound* pcA::get_mOutBound(void) const
    {
        return const_cast<cOutBound*>(&mOutBound);
    }

    bool pcA::isConnected(void) const
    {
        bool locStatus = false;

        if (mOutBound.get_mptr_icX() != nullptr)
        {
            locStatus = true;
        }

        return locStatus;
    }
}
```

The UML class diagram illustrates the following components and relationships:

- Interface ICX**: Defines methods `get_icX()` and `set_icX()`. It is implemented by `cOutBound`.
- Class cEA**: A concrete class that inherits from `pcA` (indicated by a solid line with an open arrow). It has a private attribute `mOutBound` of type `cOutBound`.
- Class pcA**: An abstract class (indicated by a hollow rectangle) that defines the `get_mOutBound()` and `isConnected()` methods. It has a private attribute `mOutBound` of type `cOutBound`.
- Class cOutBound**: A concrete class that inherits from `ICX` (indicated by a solid line with an open arrow). It implements the `get_icX()` and `set_icX()` methods. It has a private attribute `mptr_icX` of type `ICX`.

Annotations in the diagram:

- A red box highlights the `mOutBound` attribute in the `pcA` class, with a callout stating: "Returns the address of the mOutBound object to connect the port via class cA".
- A dashed box highlights the `isConnected()` method in the `pcA` class, with a callout stating: "Checks whether the interface pointer is set and thus whether the port is connected".

© MicroConsult - Microelectronics Consulting & Training GmbH 26.07.2021 F 23 MICROCONSULT

3.2.9 Accessor Class: cA.hpp

Port Implementation – Basic Concept – Accessor Class: cA.hpp

```
#ifndef __cA_HPP__
#define __cA_HPP__

#include "pcA.hpp"

namespace A
{
    class cA
    {
    public:
        cA(void) = default;
        virtual ~cA() = default;

        pcA* get_mPortA(void) const;
        void execute(void);

    private:
        pcA mPortA;
    };
}

#endif // __cA_HPP__
```

```
classDiagram
    class cA {
        <<abstract>>
        ~mPortA:pcA
        +cA()
        ~cA()
        +get_mPortA() pcA*
        +execute()
    }
    class pcA {
        <<abstract>>
        ~mOutBound:mOutBound
        +pcA()
        ~pcA()
        +get_mOutBound() mOutBound*
        +isConnected() bool
    }
    class cCoutBound {
        <<concrete>>
        ~mPortA:cA
        ~mOutBound:pcA
        +cCoutBound()
        ~cCoutBound()
        +get_mPortA() cA*
        +get_mOutBound() pcA*
        +execute()
    }
    cA <|-- cCoutBound
    pcA <|-- cCoutBound
    cA --> pcA : mPortA
    pcA o-- cCoutBound : mOutBound
```

The diagram illustrates the class hierarchy and associations for the port implementation. It features three classes: **cA** (an abstract base class), **pcA** (an abstract base class), and **cCoutBound** (a concrete class). **cCoutBound** inherits from both **cA** and **pcA**. **cA** has a pointer attribute `mPortA:pcA` and methods `cA()`, `~cA()`, `get_mPortA() pcA*`, and `execute()`. **pcA** has a pointer attribute `mOutBound:mOutBound` and methods `pcA()`, `~pcA()`, `get_mOutBound() mOutBound*`, and `isConnected() bool`. **cCoutBound** has two pointer attributes, `mPortA:cA` and `mOutBound:pcA`, and methods `cCoutBound()`, `~cCoutBound()`, `get_mPortA() cA*`, `get_mOutBound() pcA*`, and `execute()`. Dashed arrows indicate inheritance from **cCoutBound** to both **cA** and **pcA**. Solid arrows show associations: from **cA** to **pcA** (labeled `mPortA`) and from **pcA** to **cCoutBound** (labeled `mOutBound`).

Returns the address of the `mPortA` object to access the `mOutBound` element


Accesses the interface function `()` via the port object

The accessor class contains an object of port class `pcA`.

© MicroConsult - Microelectronics Consulting & Training GmbH

26.07.2021

F 24

 MICROCONSULT

3.2.10 Accessor Class: cA.cpp

Port Implementation – Basic Concept – Accessor Class: cA.cpp

```

#include "cA.hpp"

namespace A
{
    pcA* cA::get_mPortA(void) const
    {
        return const_cast<pcA*>(&mPortA);
    }

    void cA::execute(void)
    {
        // first check port connection
        // second call port function
        if (mPortA.isConnected())
        {
            mPortA.get_mOutBound() -> function();
        }
    }
}

```

```

classDiagram
    class Interface {
        <<interface>>
        +lock()
        +unlock()
        +function()
    }
    class pcA {
        <<abstract>>
        +mPortA: pcA*
        +get_mPortA(): pcA*
        +execute()
    }
    class OutBound {
        <<abstract>>
        +get_mPortA(): pcA*
        +get_mOutBound(): OutBound*
        +isConnected(): bool
        +function()
    }
    Interface <|-- pcA
    Interface <|-- OutBound
    pcA <|-- OutBound
    pcA --> OutBound : mPortA
    OutBound --> OutBound : mOutBound

```

UML Class Diagram illustrating the structure:

- Interface:** `Interface: ICK` with methods `lock()`, `unlock()`, and `function()`.
- Abstract Class:** `pcA` (marked as `<<abstract>>`) inherits from `Interface`. It contains a member `mPortA: pcA*` and methods `get_mPortA(): pcA*` and `execute()`.
- Abstract Class:** `OutBound` (marked as `<<abstract>>`) inherits from `Interface`. It contains methods `get_mPortA(): pcA*`, `get_mOutBound(): OutBound*`, `isConnected(): bool`, and `function()`.
- Relationships:**
 - `pcA` and `OutBound` both inherit from `Interface`.
 - `OutBound` inherits from `pcA` (indicated by a solid line with an open arrow head).
 - `pcA` has a reference to `OutBound` (indicated by a solid line with an open arrow head labeled `mPortA`).
 - `OutBound` has a reference to `OutBound` (indicated by a solid line with an open arrow head labeled `mOutBound`).

Annotations:

- Returns the address of the port object `mPortA`
- Final interface function() call via port and outbound object.
- Knowledge about the real interface class to which the function() belongs is not required.

© MicroConsult - Microelectronics Consulting & Training GmbH 26.07.2021 F 25 MICROCONSULT

3.2.11 Bound Class: cInBound.hpp

Port Implementation – Basic Concept – Bound Class: cInBound.hpp

```

#ifndef __cInBound_HPP__
#define __cInBound_HPP__

#include "..\Interface\icX.hpp"
using Interface::icX;

namespace B
{
    class cInBound : public icX
    {
    public:
        cInBound();
        virtual ~cInBound() = default;

        void function(void) override;
        void set_mptr_icX(icX* const ptr_icX);

    private:
        icX* mptr_icX;
    };
}

#endif // __cInBound_HPP__

```

icX is a provided interface from cB, so it is implemented in class cInBound.

cInBound contains a pointer of the provided interface type icX. This pointer has to be connected to the interface implementing object of cB.

© MicroConsult - Microelectronics Consulting & Training GmbH 26.07.2021 F 26 MICROCONSULT

3.2.12 Bound Class: cInBound.cpp

Port Implementation – Basic Concept – Bound Class: cInBound.cpp

```

#include "cInBound.hpp"

#include "..\..\Plattform\HAL\BasicIO.hpp"
using namespace Plattform::HAL;

namespace B
{
    cInBound::cInBound():mptr_icX(nullptr)
    {
        // no implementation
    }

    void cInBound::function(void)
    {
        showValue("\ncInBound::function() is called");
        if (mptr_icX != nullptr)
        {
            mptr_icX->function();
        }
    }

    void cInBound::set_mptr_icX(icX* const ptr_icX)
    {
        mptr_icX = ptr_icX;
    }
}

```

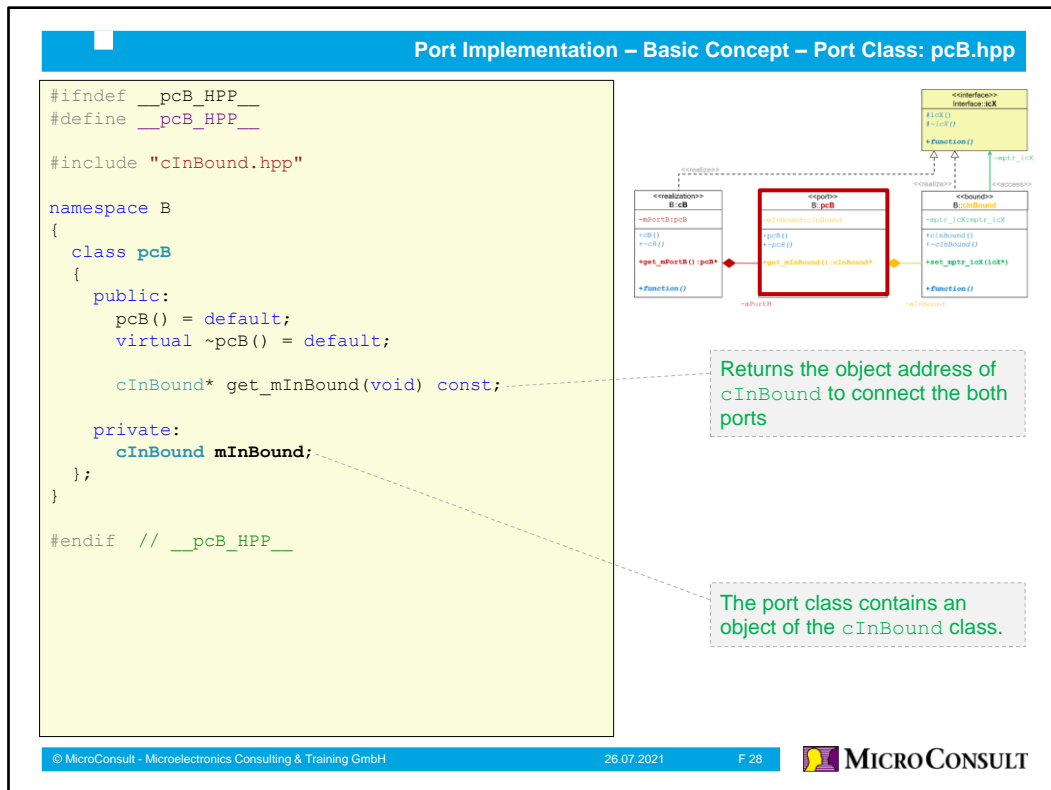
The interface pointer is initialized by a default value.

In the implementation of the interface function, cB::function() is called, which is behind the interface pointer after initialization.

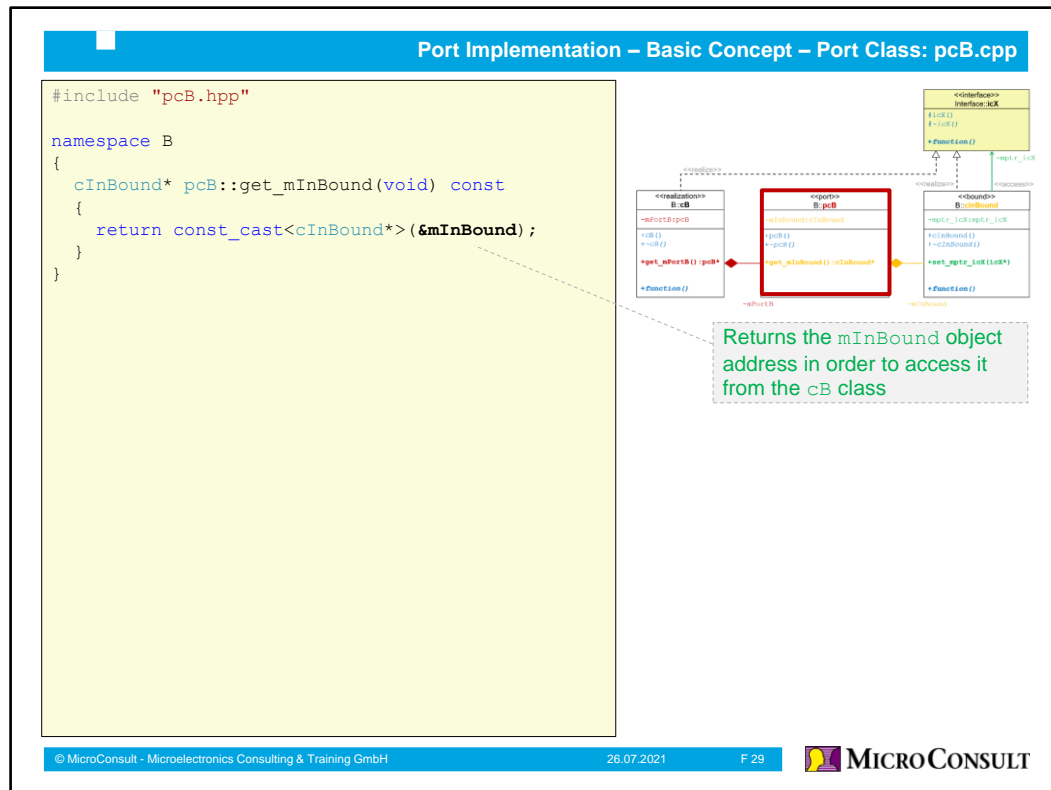
The interface pointer has to be connected to the real function() implementation in class cB.

© MicroConsult - Microelectronics Consulting & Training GmbH
26.07.2021
F 27
 MICROCONSULT

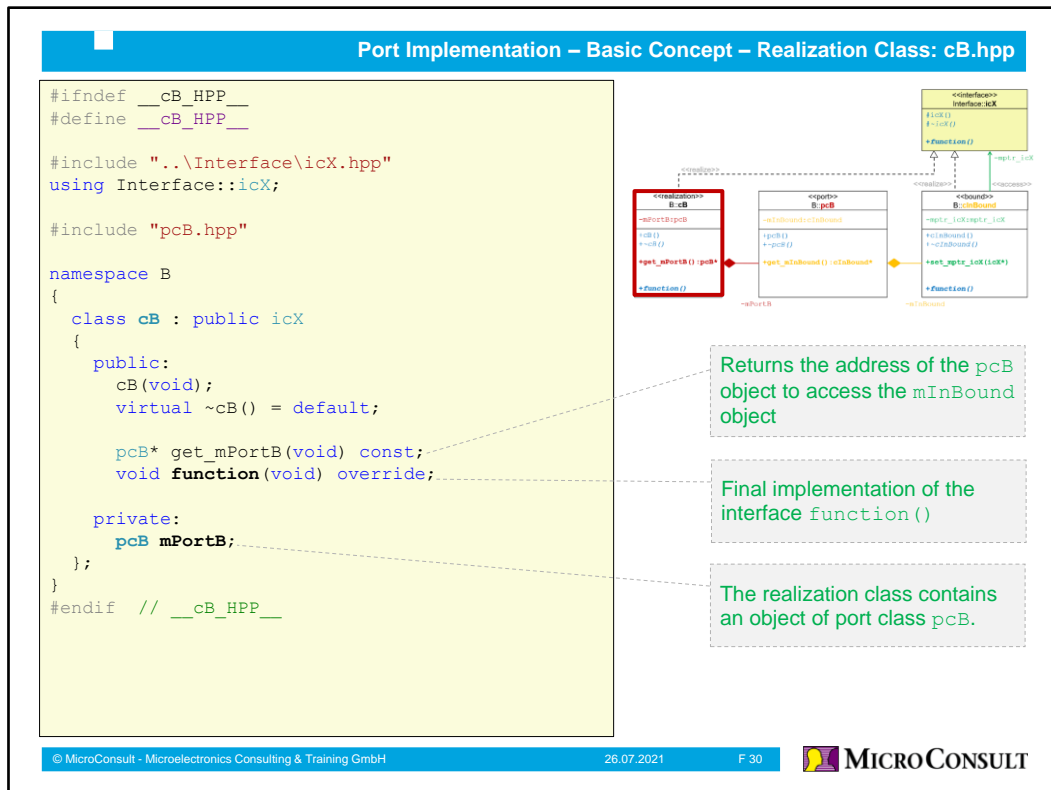
3.2.13 Port Class: pcB.hpp



3.2.14 Port Class: pcB.cpp



3.2.15 Realization Class: cB.hpp



3.2.16 Realization Class: cB.cpp

Port Implementation – Basic Concept – Realization Class: cB.cpp

```

#include "cB.hpp"

#include "..\..\Plattform\HAL\BasicIO.hpp"
using namespace Plattform::HAL;

namespace B
{
    cB::cB(void)
    {
        // class cB implements icX
        mPortB.get_mInBound()->set_mptr_icX(this);
    }

    pcB* cB::get_mPortB(void) const
    {
        return const_cast<pcB*>(&mPortB);
    }

    void cB::function(void)
    {
        showValue("\ncB::function() is call");
    }
}

```

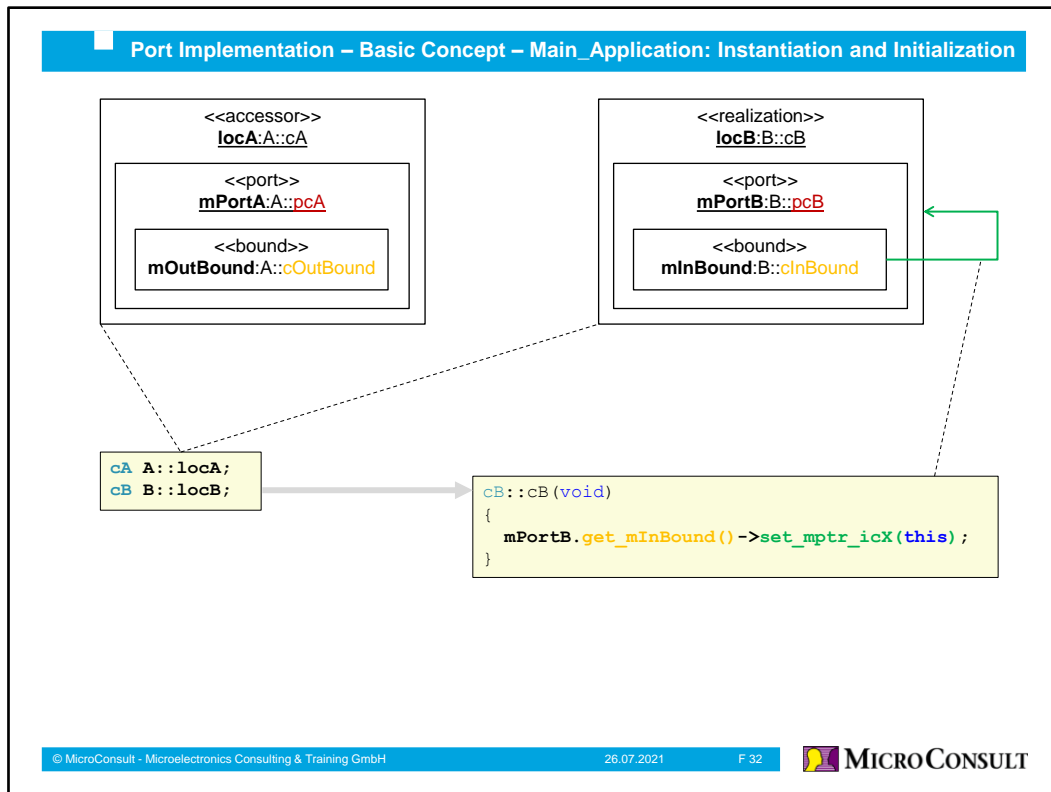
Initializes the interface pointer to the implementing object of class cB
This is done once by default in the constructor, so it is not verified anymore.

Returns the address of port object mPortB

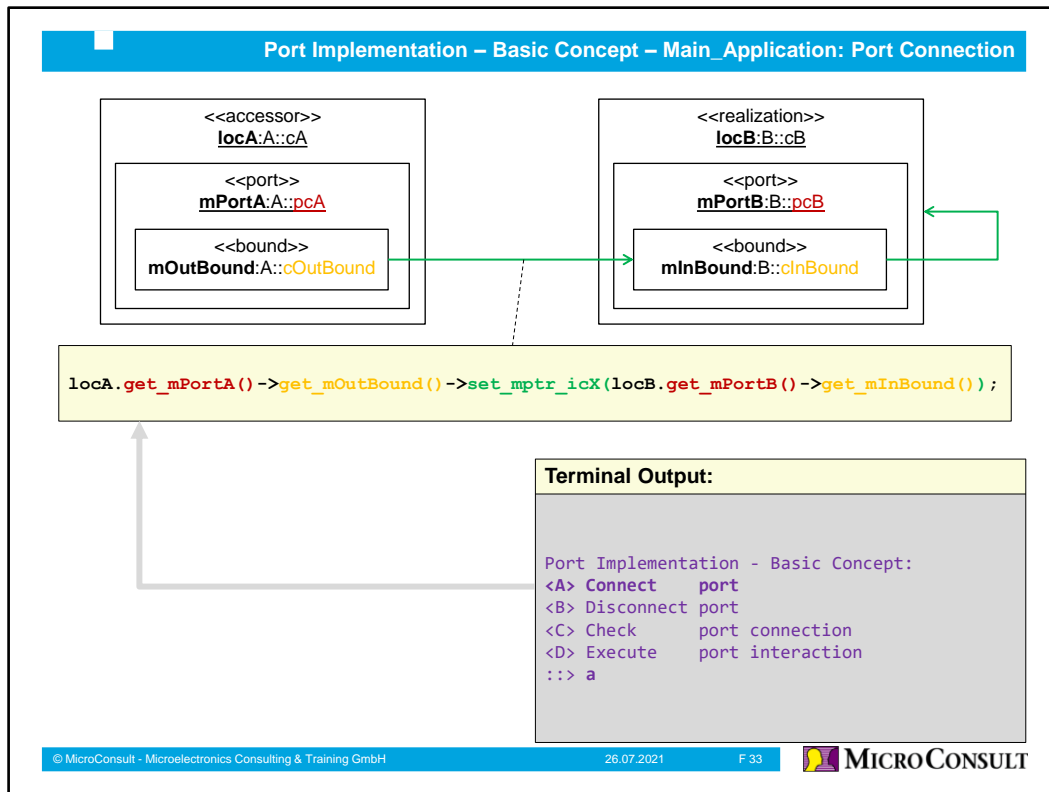
Final implementation of the interface function()

© MicroConsult - Microelectronics Consulting & Training GmbH 26.07.2021 F 31 MICROCONSULT

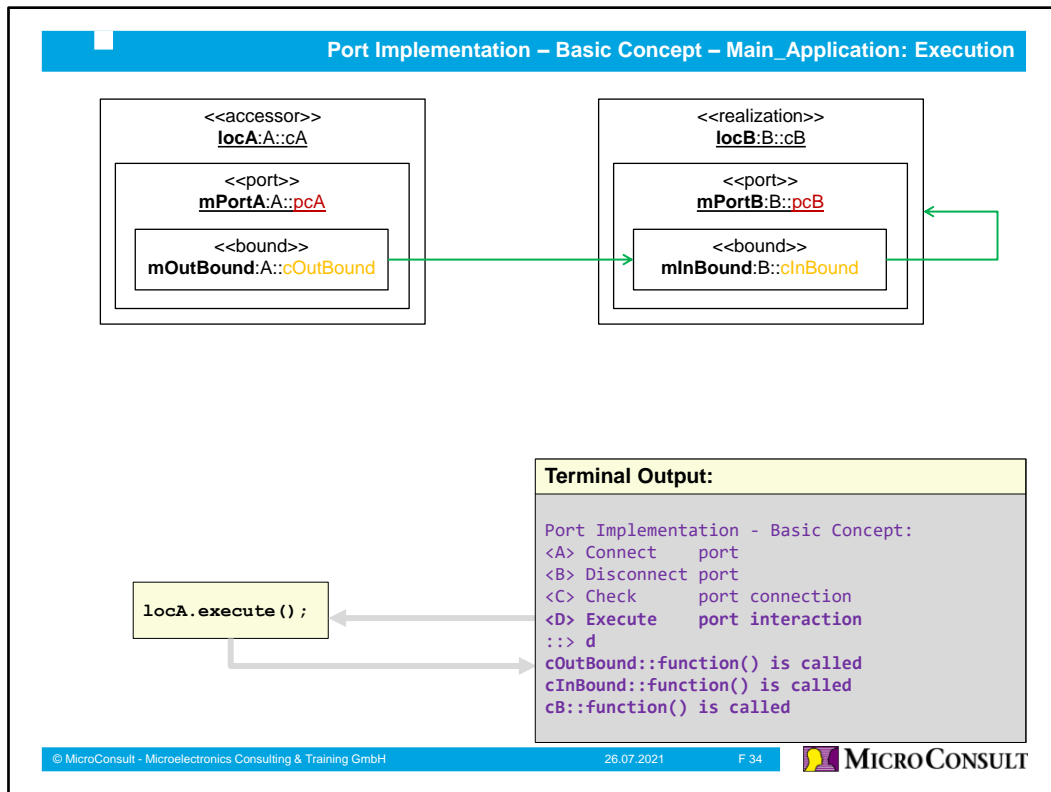
3.2.17 Main_Application: Instantiation and Initialization



3.2.18 Main_Application: Execution

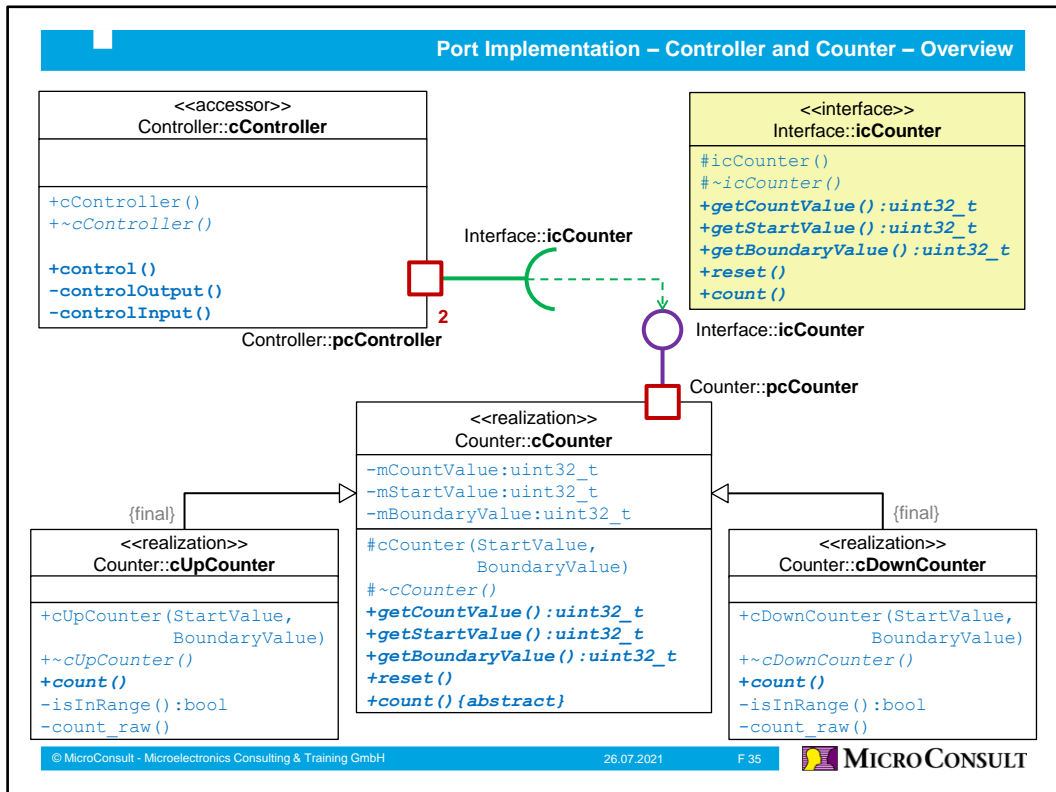


3.2.19 Main_Application: Execution

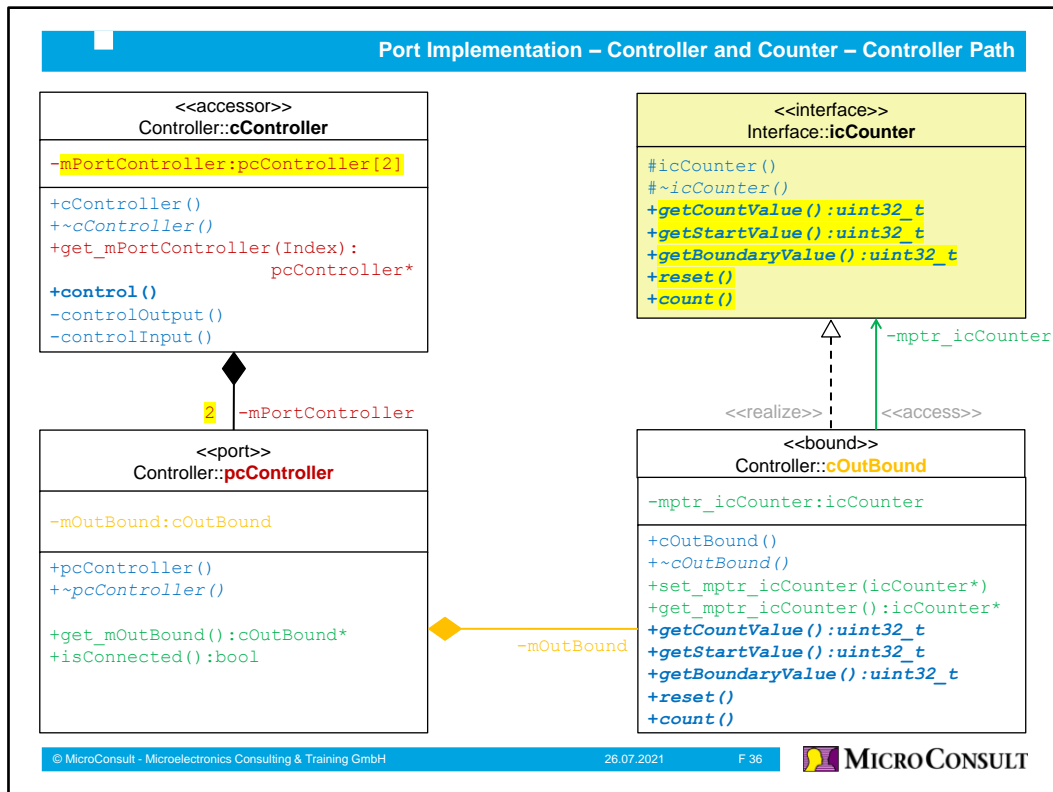


3.3 Controller and Counter

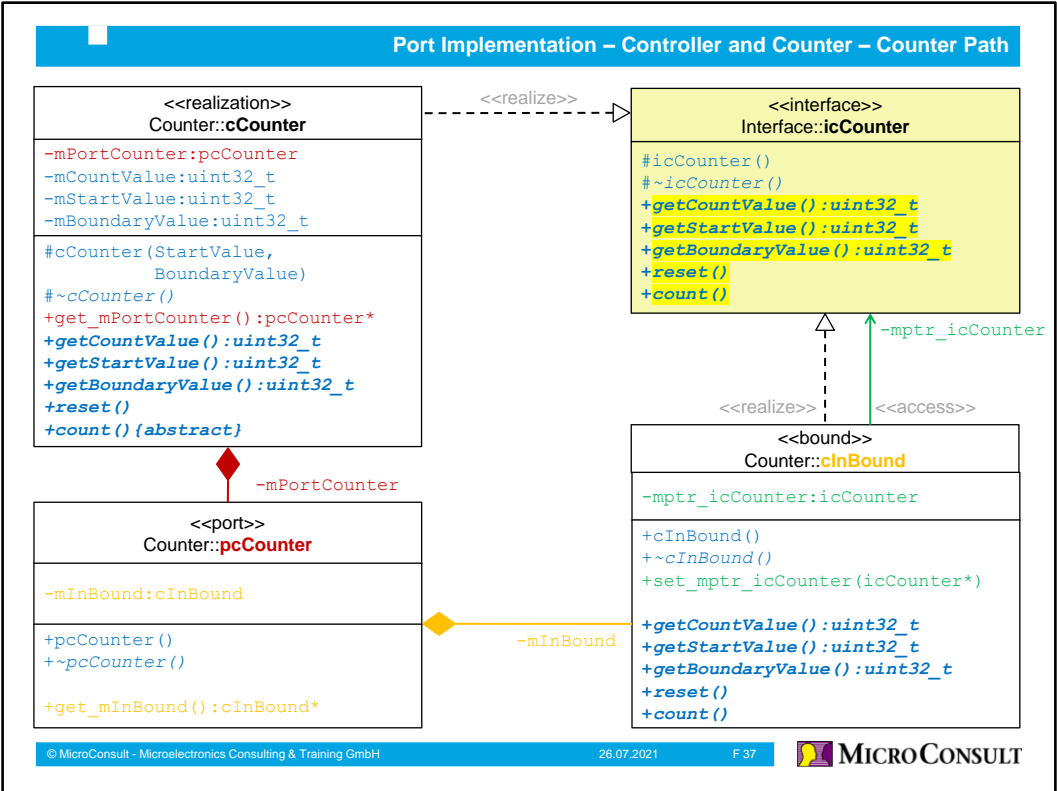
3.3.1 Overview



3.3.2 Controller Path



3.3.3 Counter Path



3.4 Additional Aspects


Port Implementation – Additional Aspects

- **Embedded classes** are used instead of embedded objects.
- Provided and required interfaces can be directly implemented in ports **without** using **bound** classes.
- **Port** connections can be **initialized** in the constructor of another (**builder**) class.
- With a **static port connection**, the successful connection only has to be verified once and not before each call via port.
→ increased performance

© MicroConsult - Microelectronics Consulting & Training GmbH

26.07.2021

F 38


 MICROCONSULT

4 Summary and Outlook

Summary and Outlook

- The port concept
 - ✓ is an **advanced interface concept**
 - ✓ allows **logical grouping** of provided and required **interfaces**
 - ✓ is applicable for **larger** embedded software **architectures**
- Define a suitable port and interface **implementation** according to the software **requirements**.
- Port and interface concepts support software **quality properties**:
 - ✓ Reusability, portability
 - ✓ Exchangeability, expandability, ...
- Port and interface concepts support software **principles**:
 - ✓ Loose coupling, externalization of dependencies
 - ✓ Modularization, high degree of cohesion, ...
- Download: <http://download.microconsult.net/ese2020/interface-designs.zip>
- Download: <http://download.microconsult.net/ese2021/port-designs.zip>

© MicroConsult - Microelectronics Consulting & Training GmbH26.07.2021F 39

 **MICROCONSULT**

5 MicroConsult Training and Coaching

MicroConsult Training and Coaching

Software Architectures for Embedded and Real-Time Systems
[English](#) [German](#)

Embedded C++ Advanced: Object-Oriented Programming for μ C with C++/EC++
[English](#) [German](#)

Embedded Software Design and Patterns with C
[English](#) [German](#)

Coaching [English and German]:
Please [contact us](#) with your topic.

MicroConsult GmbH
Dipl.-Ing. (FH) **Thomas Batt**
Senior Manager Training & Coaching


t.batt@microconsult.com
+49 (0)89 450617-35
www.microconsult.de



© MicroConsult - Microelectronics Consulting & Training GmbH

26.07.2021

F 40

 **MICROCONSULT**