2017

Team JPL: Source Language Reference Document

SWATI DHAWAN
KUNAL YADAV
JIANAN ZHENG
YIFEI LIU

Table of Contents

INTRODUCTION **3**

SYNTAX OVERVIEW **3**

KEYWORDS 3

GENERAL TYPES 4

Element *4*

Comments *5*

Property *5*

Import *6*

Constants/Symbols *6*

Identifier Visibility *6*

DIFFERENT ELEMENTS 7

Namespace *7*

PortType *7*

Port *8*

SubPortType and SubPortInstance *9*

Component *13*

Instance *14*

Connections *14*

Topology *15*

Mapping *16*

Include *17*

Argument *18*

DataType *18*

Return *19*

Interface Dictionary *19*

Subsystem *19*

Examples of Subsystem Scenarios *22*

System *23*

FULL SYSTEM EXAMPLE 24

SOME OTHER SCENARIOS OF SUBSYSTEM PORTS 29

Example 1: Different subport structure in either side of the topology *29*

Example 2: Subport connecting directly with port *32*

Example 3: Subsystemport connecting with subsystemport of child type *34*

PROJECT HIERARCHY 37

PRIMITIVE SEMANTIC RULES FOR SOURCE LANGUAGE 37

PRIMITIVE SYNTAX RULES FOR SOURCE LANGUAGE 39

REFERENCES 43

Introduction

This document is a reference manual for the source modeling language which will be used in the text-based editor of the modeling tool. The language that has been defined is customized to JPL domain. This source language will be parsed by the customized parser and will be converted into representation language.

Syntax Overview

The syntax is represented using declarative language form. We have included keywords which will help the parser to interpret the code properly. Also, the curly braces with each element is used so that parser would know the scope of the element. The sub-elements/attributes (key-value pairs) are delimited through newline character and are mentioned in form of <key-name> = < value>.

Keywords

The following keywords are reserved and may not be used as identifiers. Please note that these keywords are *case-insensitive.*

component

port

namespace

instance

subsystem

topology

enum

portType

arg

return

import

from

include

constant

subPortType

subPortInstance

mapping

system

datatype

dict

General Types

Element

The elements are the basic building block of the system design. There can exists two kinds of elements - Block element or single element.

Each block element name has a different way of declaration which are described below:

If the element has a type attribute (e.g. instance, port, argument, return, ...)

Block element with type

```
<element type> <element name>:<type> { <attributes> }

or

<element type> <element name>:<namespace>.<type> { <attributes> }
```

If the element does not have a type attribute (e.g. component, port_type, ...), we ignore type attribute and represent as below:

Block Element without type

```
<element type> <element name> { <attributes> or <subelements> }
```

For normal element, the element is described without curly braces:

Normal element

```
<element type/keyword> <element name>
```

Here are some examples:

Element definition example

```
Component SignalGen{ //block element without type
        .....
}


Port p2_out:Fw.CmdReg{ //block element with type and namespace
        ....
}


Instance SG1:SignalGen{ //block element with type but without namespace
   .....
}


datatype ParamBuffer //normal element
```

Comments

Comments refer to the textual part in the source language which will not be parsed by the parser. Comments are detected by "//"

Please note that "comment"as an property will be parsed by the parser as an attribute(example below)

Comment Declaration

```
//Commented part of the block which will not be parsed


//Comment as property
Comment= Comment to be parsed
```

Property

Also referred as "attribute" in this document. Property inside any element are defined as key-value pair separated by an assignment operator(=). Like below are properties of component.

Property Declaration

```
Component SignalGen {
        Kind=queued
        Namespace=ref
```

```
}
```

In case, any attribute has space inside it, then it should be enclosed inside double quotes. However, key in the property should not have spaces.

Property Declaration

```
Component SignalGen {
        someproperty= "some value"
}
```

Import

This keyword enables the import of other namespaces/subsystems through namespaces so that instances from those namespace scan be referred in current project.

Import declaration

```
//importing projects
import Pr1
import FwPr


//importing port type or subsystem from project
import TimePortAi from Pr1
import events from FwPr
```

Constants/Symbols

This is to define the constants in the namespace to access the integer values directly. These can be declared inside a namespace.

constant declaration

```
Constant SIGNAL_OUTPUT = 1
```

Identifier Visibility

The visibility of the component instances are identified using keys (Private or public). By default it is public. Like, for below instances,

```
Instance SG1:SignalGen{
    BASE_ID=1
}

Instance SG2:SignalGen{
        BASE_ID=2
}
```

Above instances example can be used in below manner. When Topology2 extends Topology1, then connections related to SG1 will not be imported.

Topology Declaration with Contains

```
topology Topology1{
        private SG1

        SG1.port1 ->  Comp2.port1
    SG2.port2  -> Comp1.port2
}

//Topology2 will not contain SG1.port1 ->  Comp2.port1 because SG1 is private
instance
topology Topology2 contains Topology1{
        Comp2.port3 -> Comp1.port3
    Comp3.port1 -> Comp4.port1
}
```

Different Elements

Namespace

This is unique name identifier to distinguish different namespaces and should be specified at the beginning of the file(top element in the file). There also can be the concept of nested namespaces.

namespace Declaration

```
namespace SignalGenerator
```

The namespace in the language defines the scope of the element. Either the namespace could be defined as an attribute in the element or it could also be included in the element type.

Namespace declaration

```
...
        Namespace=Fw
...
...
        Type= Fw.CmdReg
...
```

PortType

This is basically the definition of the port as how the port instance should be declared and what properties/attributes it should have.

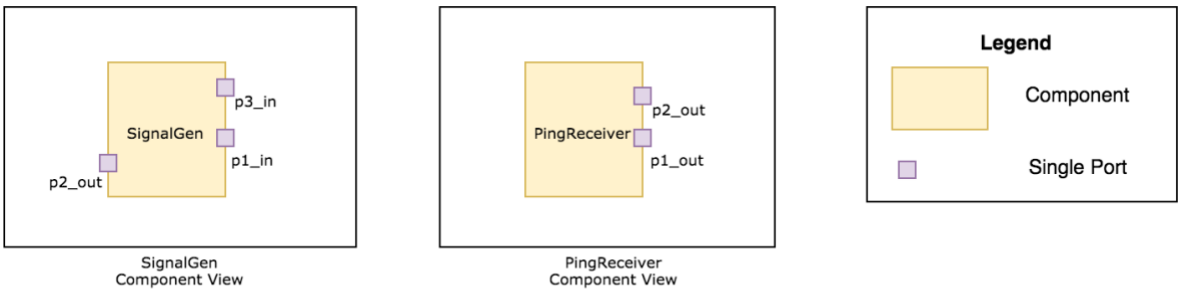
Port Type Declaration

```
PortType PrmGet {
         = Fw
        arg id:FwPrmIdType {
                pass_by = value
```

```
        }
        arg val:Fw.ParamBuffer {
                pass_by = reference
        }
        return {
                type = ParamValid
        }
}namespace
```

Port

The port instance declaration is included in the component (explained later below) and the port's attributes are represented in the form of key-value pair. The namespace is also included in the variable definition i.e. **<port name>:<namespace>.<port type>**



Port Declaration

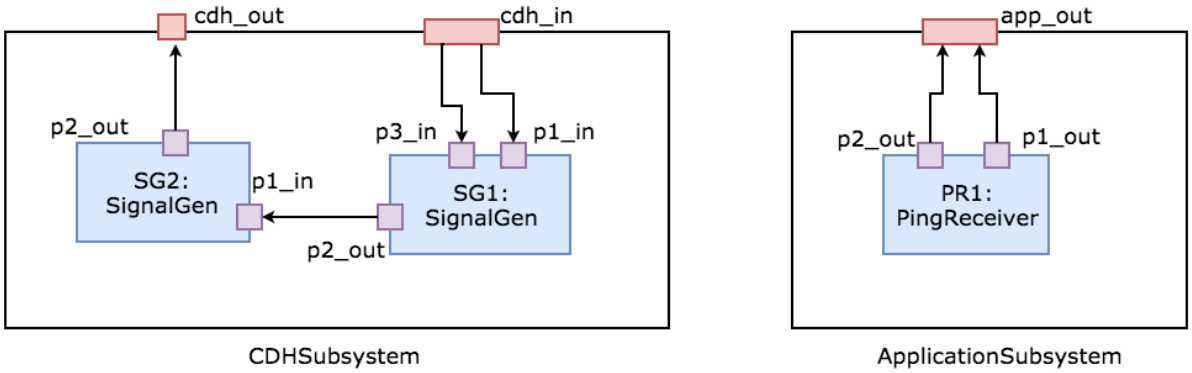
```
port p2_out:Svc.Sched{
        Direction= Out
        ...
}

port p1_in:Svc.Sched{
        Direction= In
        ....
}
port p3_in:Fw.CmdReg {
        direction=in
        kind=guarded
        ...
}
port p1_out:Svc.Sched{
        direction=out
        kind=sync
        ...
}
port p2_out:Fw.CmdReg {
        direction=out
        kind=async
        ...
}
```

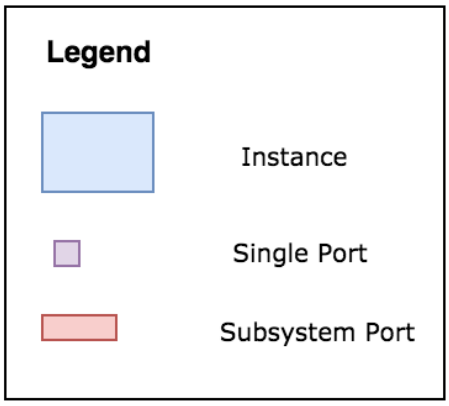
In case, the port capable of having multiple connection, the multiplicity can be defined by below definition. If "number" is not defined, it will be 0..1 by default(i.e. port can hold one connection). In a similar manner, max number of connections can also be defined with the same attribute. Please note that this holds only for output ports.

multiple connection port

```
port p1_in:Svc.Sched{
        Direction= In
        number= 0..5
}


//or just define max number of connections
port p1_in:Svc.Sched{
        Direction= In
        number= 5
}
```

SubPortType and SubPortInstance





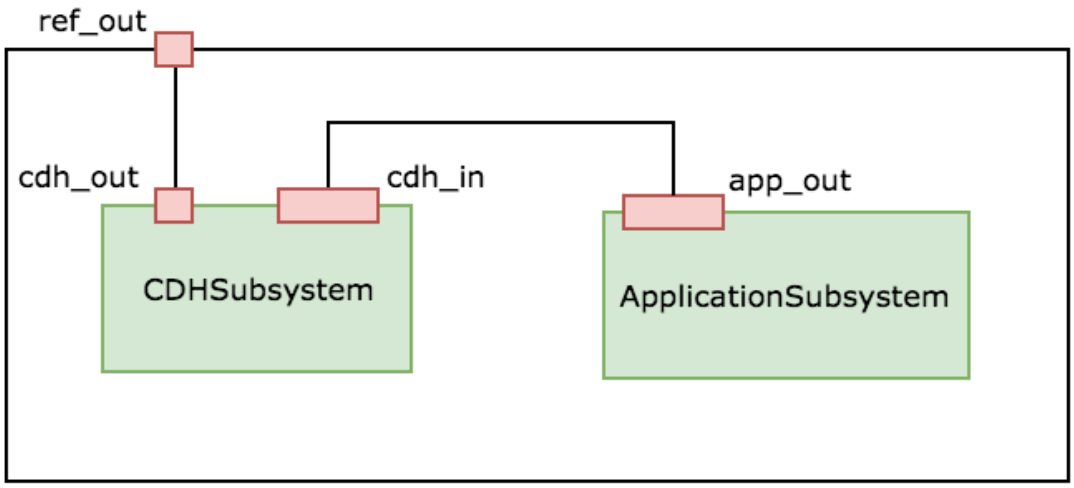
SubPortType

These represent the external port of the subsystem. These will be the heterogeneous array of all types of port where each element is uniquely identified by **<distinctportname>:<porttype>**.

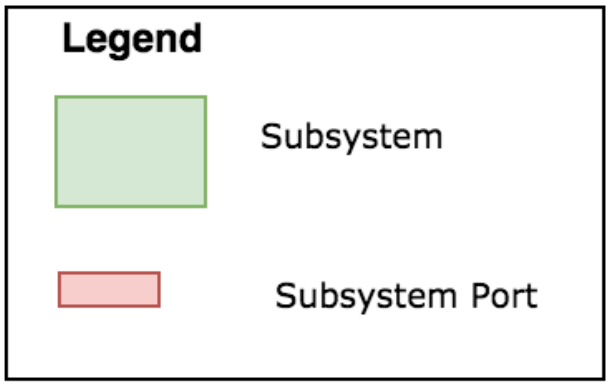
External Port Declaration

```
SubPortType extports_in{
        c1:Fw.CmdReg
        c2:Svc.Sched
}


SubPortType extports_out{
        c3:Svc.Sched
}
```

Even when the internal port is of "subporttype" then identifier is needed.



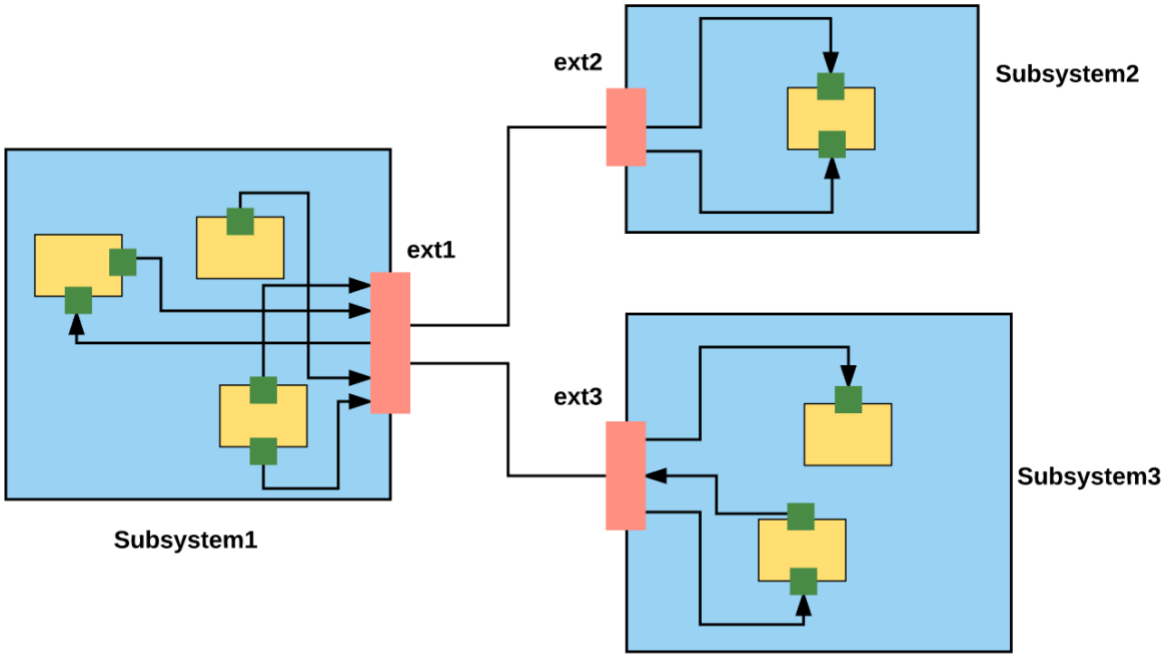
Ref



Subsystem port within Subsystem Port

```
SubPortType extPorts_parent_out{
        c6:extPorts_out
}
```

Also, two subsystem ports may not contain same number of port types. This means that there can exists different combination of subsystem ports with each other.




```
SubPortType ext1{
        c1:Fw.CmdReg
        c2:Svc.Sched
        c3:Svc.Tlm
        c4:Svc.Sched
        c5:Fw.Reg
}

SubPortType ext2{
        c2:Svc.Sched
        c5:Fw.Reg
}

SubPortType ext3{
        c1:Fw.CmdReg
        c3:Svc.Tlm
        c4:Svc.Sched
}
```

SubPortInstance

This represent the instance of the external port on the subsystem. The definition includes **subportinstance <identifier>: <subport type>**

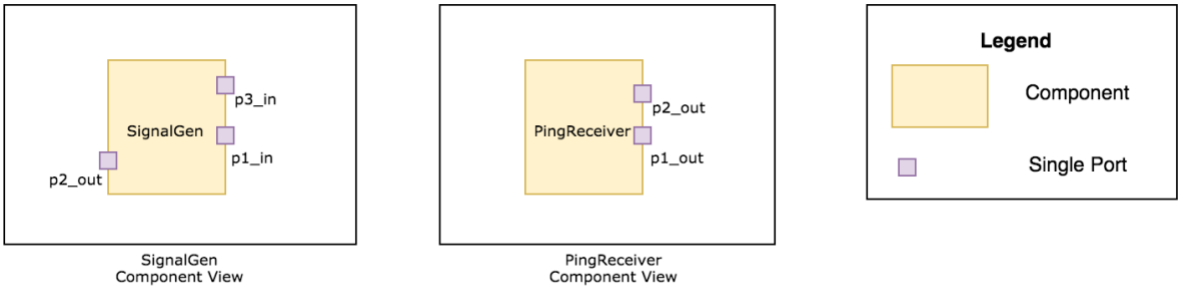
External Port Declaration

```
subportinstance cdh_in:extPorts_in
subportinstance app_out:extPorts_in
```

```
subportinstance cdh_out:extPorts_out
subportinstance ref_out:extPorts_parent_out
```

Component

The component is a block element not only includes the component attributes which is basically key-value pair, but also the port instances which are related to that component. The namespace of the component is taken from the nearest enclosed namespace.

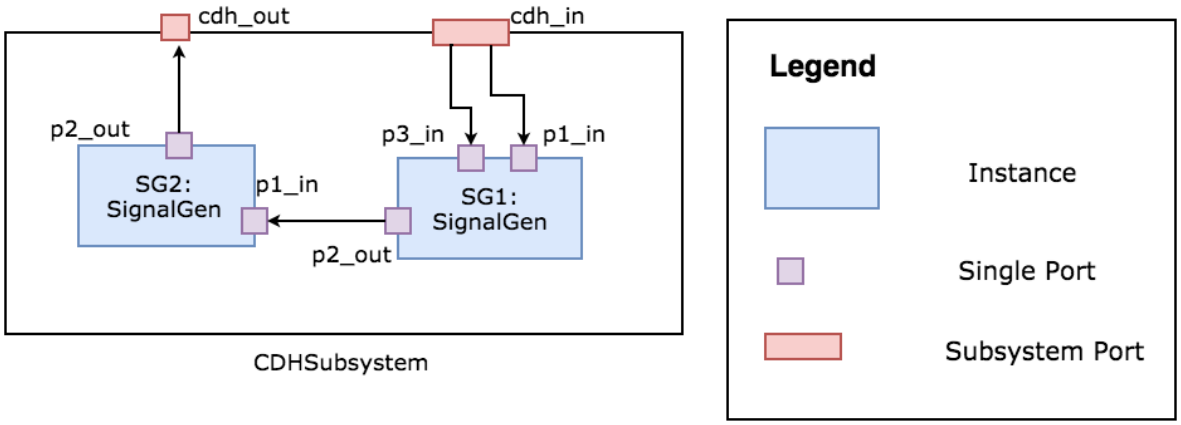


Component Declaration

```
component SignalGen {
        port p2_out:Svc.Sched{
         Direction= Out
         max_number=1
    }

        port p1_in:Svc.Sched{
         Direction= In
         max_number=1
    }
        port p3_in:Fw.CmdReg {
                direction=in
                kind=guarded
                ...
        }
}


component PingReceiver{
        port p1_out:Svc.Sched{
                direction=out
                kind=sync
                ...
        }
        port p2_out:Fw.CmdReg {
                direction=out
                kind=async
                ...
        }
}
```

Instance

Instances are defined inside the subsystem or the topology. The definition of the instance should include BASE_ID which is supposed to be unique in the system space.

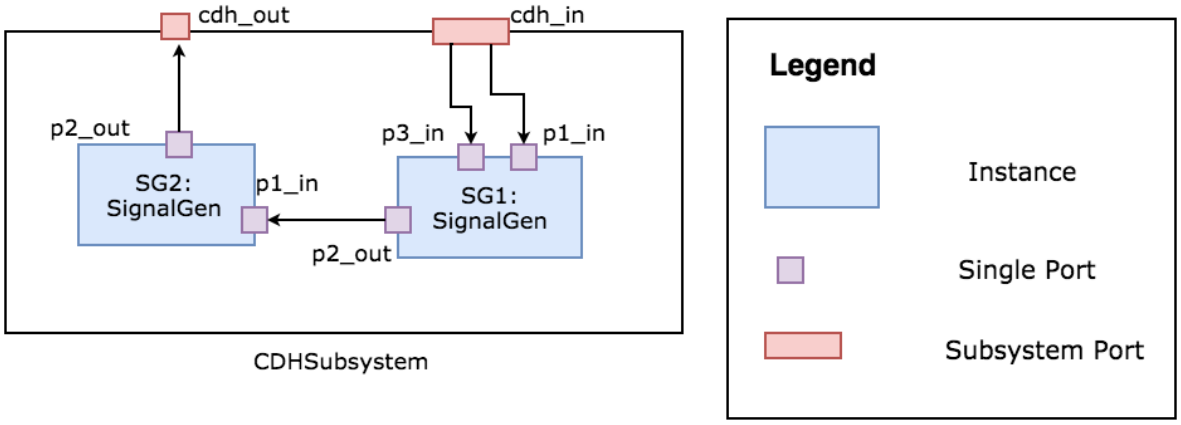


Instance Declaration

```
Instance SG1:SignalGen{
    BASE_ID=1
}
Instance SG2:SignalGen{
    BASE_ID=2
}
```

Connections

The connections basically defines the connection between two components, two subsystem or a component and subsystem. The direction of the connection is specified by an arrow(->) which basically indicates that connection is going from source to target i.e. **<source> → <target>**

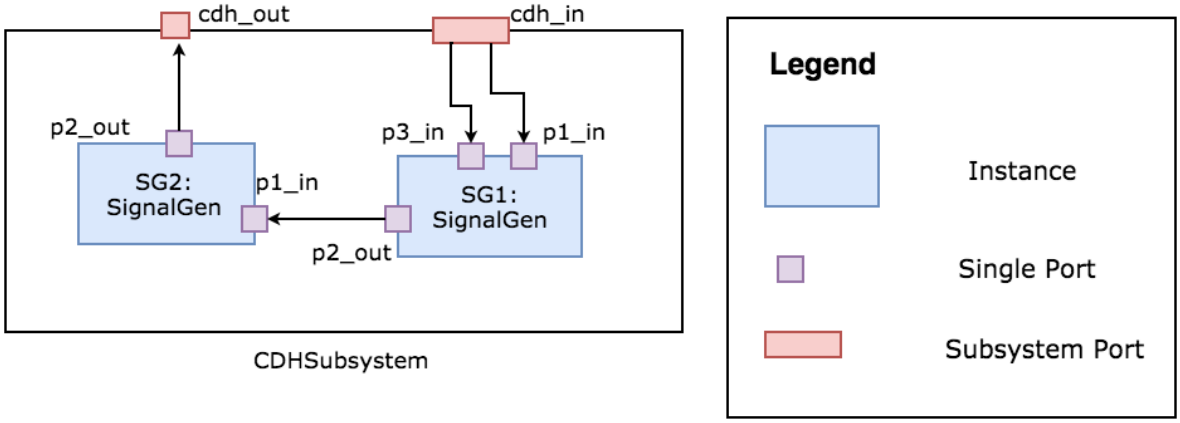


Connection Declaration

```
SG1.p2_out -> SG2.p1_in
```

Topology

The topology will contain only connections related to the topology graph. The connection will be defined between source and target element, so it is mainly defined as **<source element>.<output port>** →**<target element>.<input port>**. These source/target elements can be component instance or subsystem.



Topology Declaration

```
topology topo1{
        SG1.p2_out -> SG2.p1_in
}
```

Topology inheritance

In case where one topology is included another topology, below is the way to represent it. The connections that are present in Topology 1 will be inherited by the Topology 2. Various topologies can be defined inside a subsystem or system.

Topology Declaration with Contains

```
topology Topology1{
        Comp1.port1 ->  Comp2.port1
    Comp2.port2  -> Comp1.port2
}

topology Topology2 contains Topology1{ // contains is like import. Topology2
imports Topology1
        Comp2.port3 -> Comp1.port3
    Comp3.port1 -> Comp4.port1
}
```

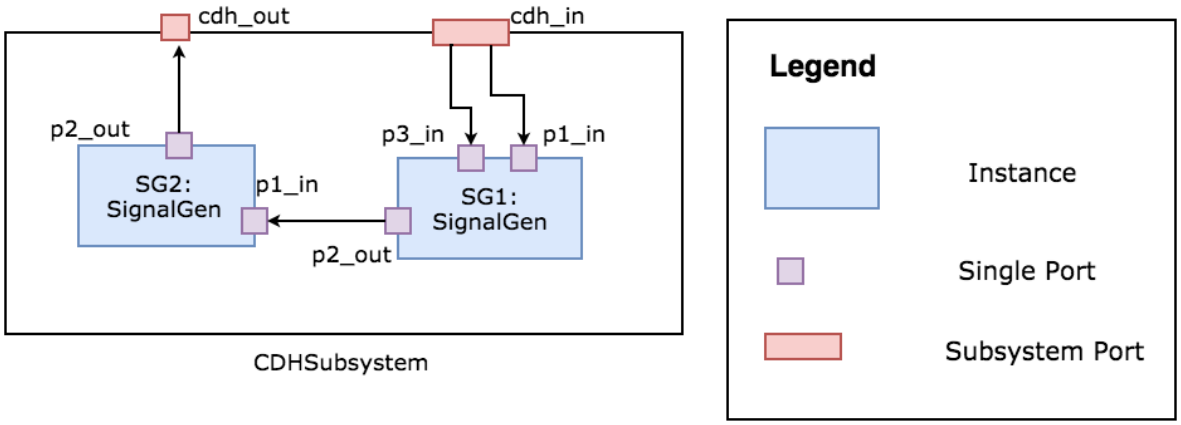
Also, in case of multi-connection port array(lets assume p1_in is an array port with size 2), the topology can be defined as

Topology Declaration

```
//Constant can be declared as constant EXT_OUT_B = 0
topology topo1{
        SG1.p2_out -> SG2.p1_in [EXT_OUT_B] //this is similar to SG1.p2_out ->
SG2.p1_in[0]
}
```

Mapping

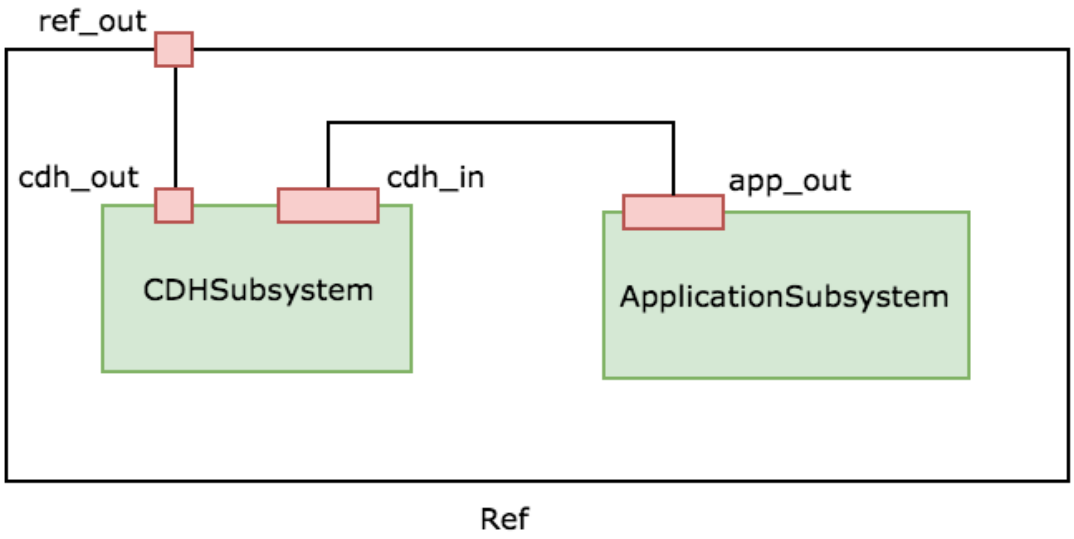
This is the mapping defined between internal and external ports in the subsystem. The direction of ports specified by the direction of the connector word(to) i.e. **source to target**

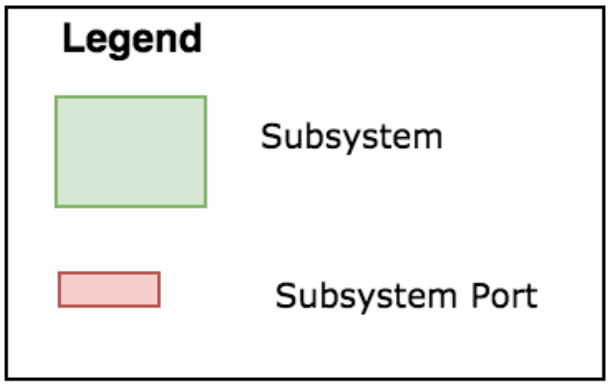


Mapping Declaration

```
Mapping  CDHmap{
        cdh_in.c1 to SG1.p1_in
        cdh_in.c2 to SG1.p3_in
        SG2.p2_out to cdh_out.c3 //order matters. source to target
}
```

However in case of mapping of subsystem porttype to parent subsystemport does not need to be in order(as there is no direction attached with subsystem port type)





mapping declaration with subsystem port type

```
Mapping map1{
        CDHSubsystem.cdh_out to Ref.ref_out //or Ref.ref_out to
CDHSubsystem.cdh_out
}
```

Include

This keyword used in port-type to include header files. The path should be enclosed within double quotes

Include Declaration

```
Include "Fw/Prm/PrmBuffer.hpp"
```

Also, to import other information like where to find the C++ implementation of each instance and how to initialize each instance. We could specify the file through "include" keyword.

Including Other information

```
Include "code.cpp"
```

Argument

These are basically argument definition for ports and thus, these are enclosed within port definition.

Argument Declaration

```
arg response:CommandResponseEnum{
    Pass_by = Value
    comment = Some comment here
}
```

DataType

There can be various kinds of data types. Some known ones are array, struct and enum. JPL developer can even design customized data types as well.So, for definition of struct or array, below definition should be used.

Data type definition

```
datatype <datatype name>
```

Enum

This is special kind of datatype. "Enum" like any other language is the enumerator type of the language.

Enum Declaration

```
enum CommandResponseEnum{
        COMMAND_OK = 0
    COMMAND_INVALID_OPCODE = 1
    COMMAND_VALIDATION_ERROR = 2
    COMMAND_FORMAT_ERROR = 3
    COMMAND_EXECUTION_ERROR = 4
}
```

In another way , we can directly inline the enum definition. Like below:

Argument with Enum Def

```
arg response:CommandResponseEnum{
    Pass_by= Value
    comment= Some comment here
```

```
        enum CommandResponseEnum{
                COMMAND_OK = 0
        COMMAND_INVALID_OPCODE = 1
        COMMAND_VALIDATION_ERROR = 2
        COMMAND_FORMAT_ERROR = 3
        COMMAND_EXECUTION_ERROR = 4
        }
}
```

Return

These are basically return type definition for ports and thus, these are enclosed within port definition.

Return Declaration

```
return {
        pass_by= value
        type= CommandResponseEnum
}
```

Interface Dictionary

These are the definitions of the interfaces that are included in the project. As of now, there are three types; telemetry, command and event.

Interface dictionary definition

```
dict <top element>{
        <child element>{
        }
}


//Example:Telemetry definition
dict telemetry{
        channel{
        }
}


//Example:Events definition
dict events{
        event{
                id=0
                ...
        }

        event{
                id=1
                ...
        }
}
```

Subsystem

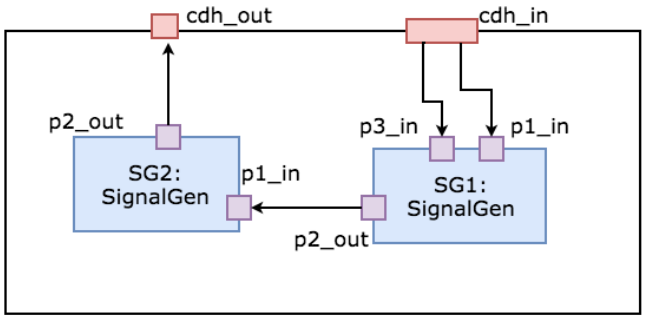
The Subsystem will contain the subsystem port, mapping(connection between external port and internal port) and topology(which consists of topologies). The subsystem can also contain multiple subsystem inside it.

Subsystem Declaration

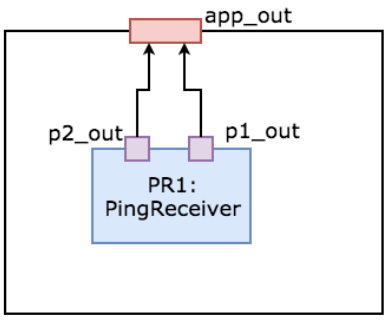
```
//Subsystem declaration
Subsystem CDHSubSystem{
        SubPortInstance cdh_in:extPorts_in
        SubPortInstance cdh_out:extPorts_out

    Mapping     CDHmap{
                cdh_in.c1 to SG1.p1_in
                cdh_in.c2 to SG1.p3_in
                SG2.p2_out to cdh_out.c3
        }
    topology topo1{
        SG1.p2_out -> SG2.p1_in
    }
}
```

Subsystem Declaration

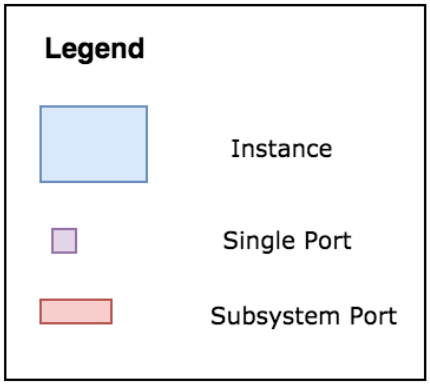
```
//Subsystem declaration
Subsystem ApplicationSubsystem{
        SubPortInstance app_out:extPorts_in

        mapping AppMap{
                PR1.p1_out to app_out.c1
                PR1.p2_out to app_out.c2
        }
}
```



CDHSubsystem



ApplicationSubsystem



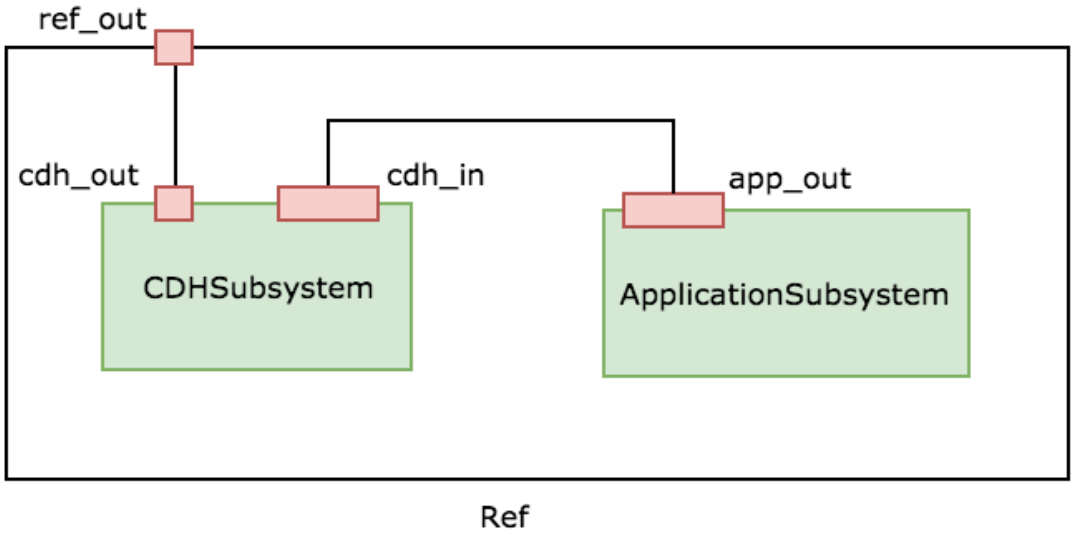
Below code describes about the parent level of subsystems(mentioned above).

Subsystem Port Declaration

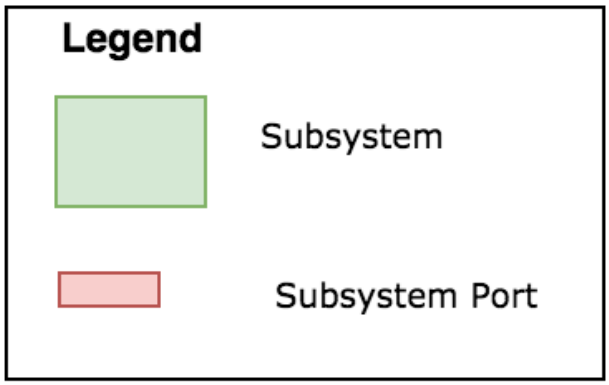
```
SubPortType extPorts_parent_out{
        c6:extPorts_out
}
```

Subsystem Declaration

```
//Subsystem Declaration
subsystem Ref{
        SubPortInstance ref_out:extPorts_parent_out

        mapping refMap{
                CDHSubsystem.cdh_out to ref_out.c6
        }

        topology sub{
                ApplicationSubsystem.app_out -> CDHSubSystem.cdh_in
        }
}
```



Ref



Examples of Subsystem Scenarios

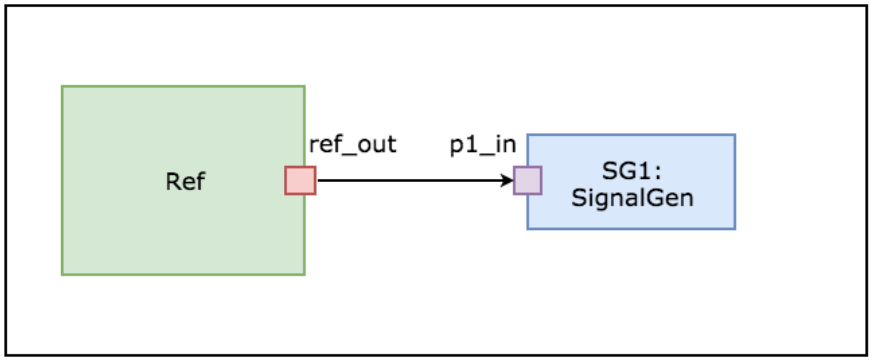
Below are different scenarios for a subsystem with examples:

System

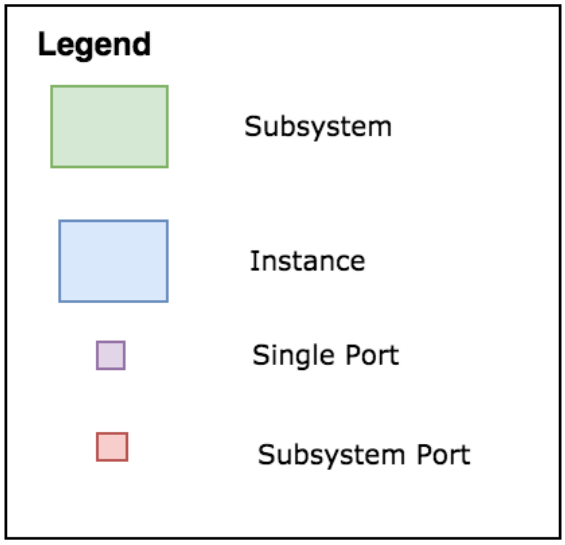
This is the top level module of the system definition. This contains all instances declaration and other topologies(which is not covered in internal subsystems). *There can only be one system in the namespace*.

System Declaration

```
System sys1{

        //Instance declaration
        Instance SG1:SignalGen{
                BASE_ID=1
        }
        Instance SG2:SignalGen{
        BASE_ID=2
        }
        Instance PR1:PingReceiver{
        BASE_ID=3
        }

        // Using the subsystems
        topology demo_subsystem{
        Ref.ref_out -> SG1.p1_in
        }
}
```



sys1

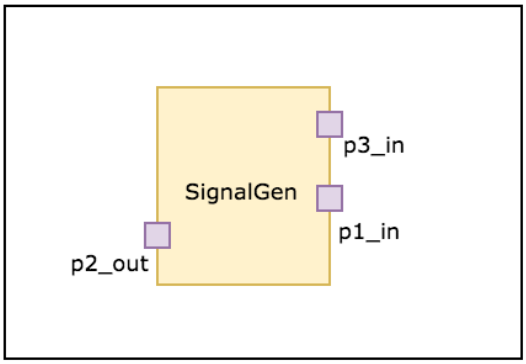


Full System Example

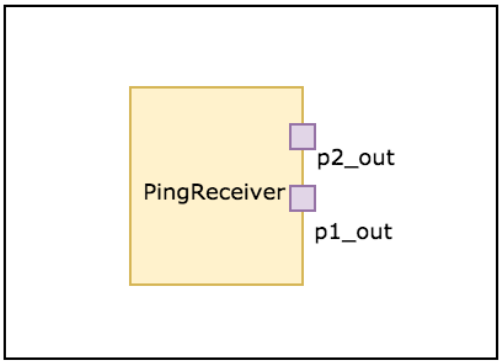
Component Declaration

```
namespace Svc


//Component Definitions
component SignalGen {
        port p2_out:Svc.Sched{
         Direction= Out
         max_number=1
    }

        port p1_in:Svc.Sched{
         Direction= In
         max_number=1
    }
        port p3_in:Fw.CmdReg {
```

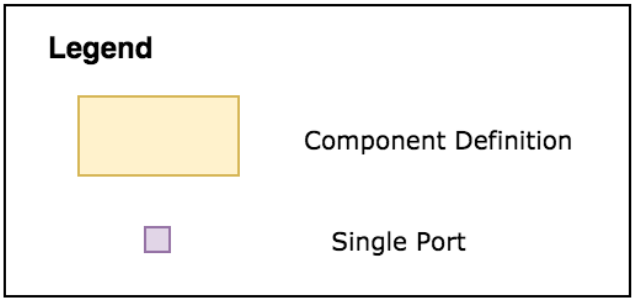
```
                direction=in
                kind=guarded
                ...
        }
}


component PingReceiver{
        port p1_out:Svc.Sched{
                direction=out
                kind=sync
                ...
        }
        port p2_out:Fw.CmdReg {
                direction=out
                kind=async
                ...
        }
}
```



SignalGen
Component View



PingReceiver
Component View



Subsystem Port Declaration

```
//SubportType Definitions
SubPortType extports_in{
        c1:Fw.CmdReg
```

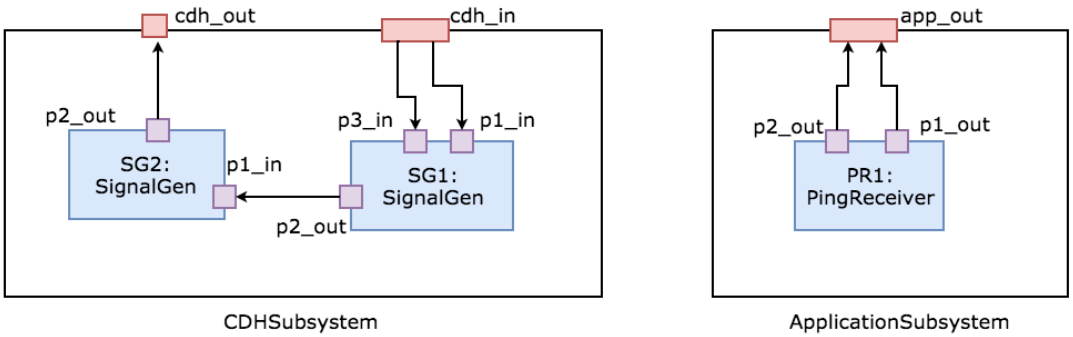
```
        c2:Svc.Sched
}

SubPortType extports_out{
        c3:Svc.Sched
}

//Subporttype consists of subsystem port(no identifier used)
SubPortType extPorts_parent_out{
        c6:extPorts_out
}
```

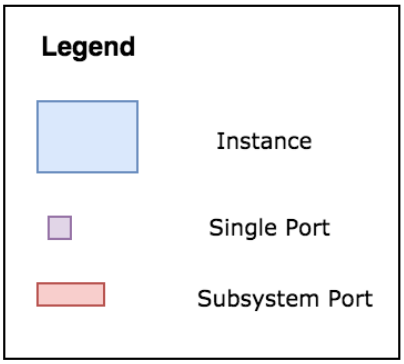
CDH Subsystem(Level 3)

```
//Subsystem declaration
Subsystem CDHSubSystem{
        SubPortInstance cdh_in:extPorts_in
        SubPortInstance cdh_out:extPorts_out

    Mapping     CDHmap{
                cdh_in.c1 to SG1.p1_in
                cdh_in.c2 to SG1.p3_in
                SG2.p2_out to cdh_out.c3
        }
    topology topo1{
        SG1.p2_out -> SG2.p1_in
    }
}
```

Application Subsystem(Level 3)

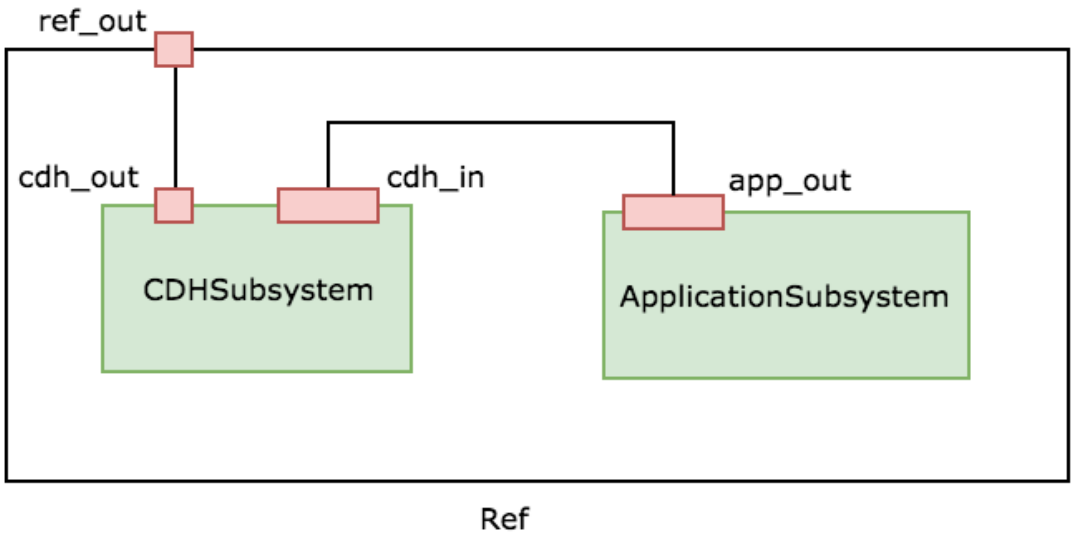
```
//Subsystem declaration
Subsystem ApplicationSubsystem{
        SubPortInstance app_out:extPorts_in

        mapping AppMap{
                PR1.p1_out to app_out.c1
                PR1.p2_out to app_out.c2
        }
}
```



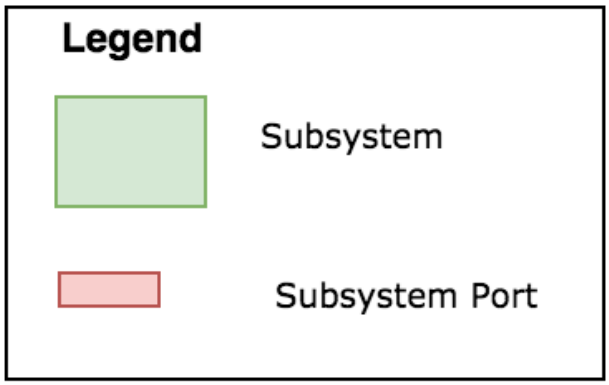


Ref Subsystem(Level 2)

```
//Subsystem Declaration
subsystem Ref{
        SubPortInstance ref_out:extPorts_parent_out

        mapping refMap{
                CDHSubsystem.cdh_out to ref_out.c6
        }

        topology sub{
                ApplicationSubsystem.app_out -> CDHSubSystem.cdh_in
        }
}
```

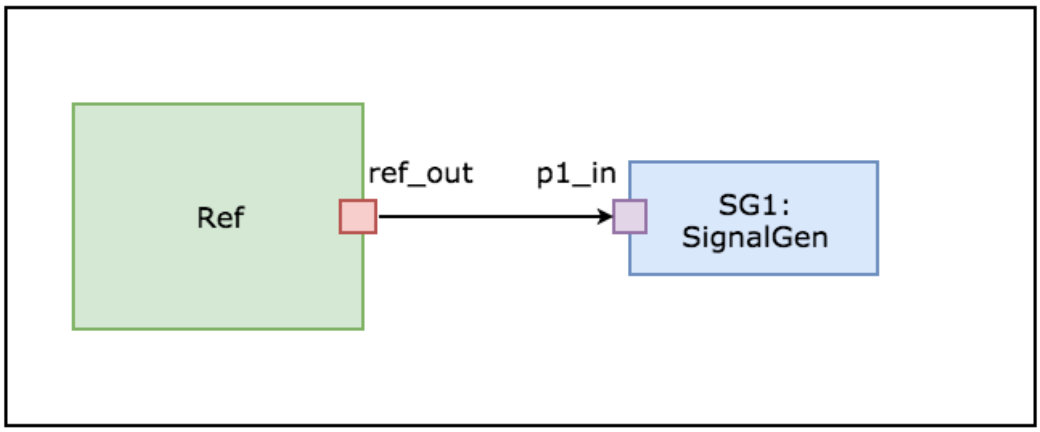


Ref

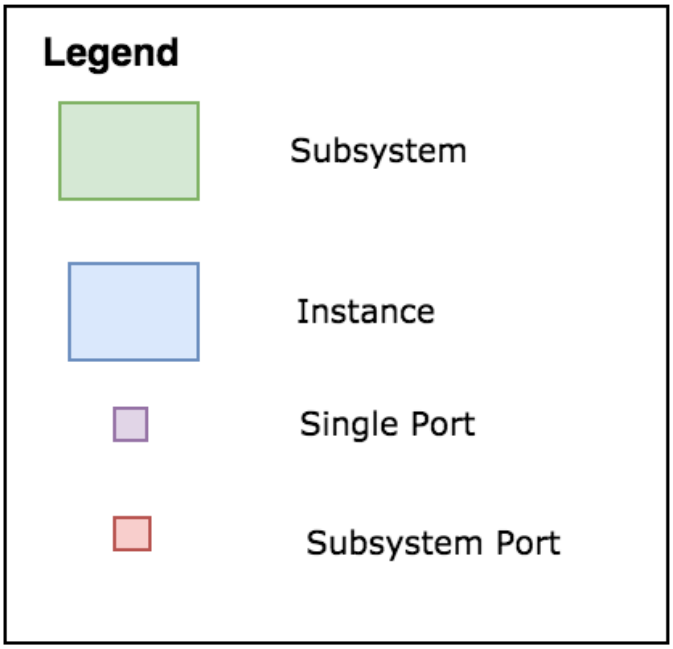


System(Level 1)

```
System sys1{

        //Instance declaration
        Instance SG1:SignalGen{
                BASE_ID=1
        }
        Instance SG2:SignalGen{
        BASE_ID=2
        }
        Instance PR1:PingReceiver{
        BASE_ID=3
        }

        // Using the subsystems
        topology demo_subsystem{
        Ref.ref_out -> SG1.p1_in
        }
}
```

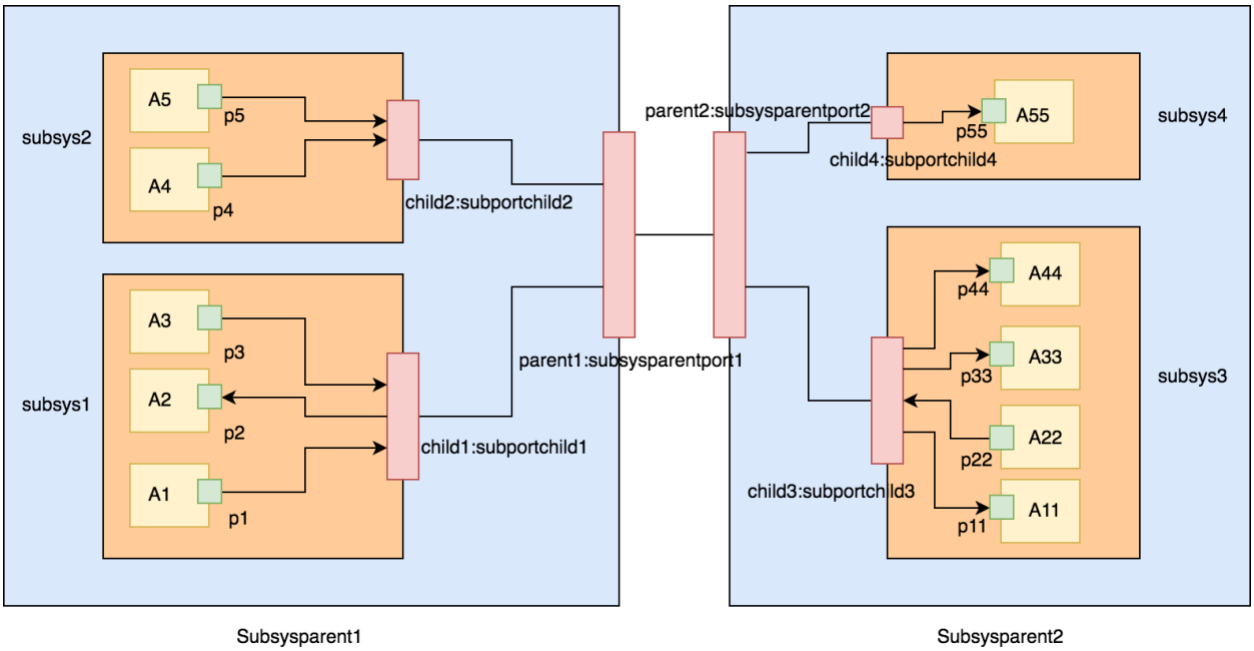


sys1



Some other scenarios of Subsystem Ports

Example 1: Different subport structure in either side of the topology



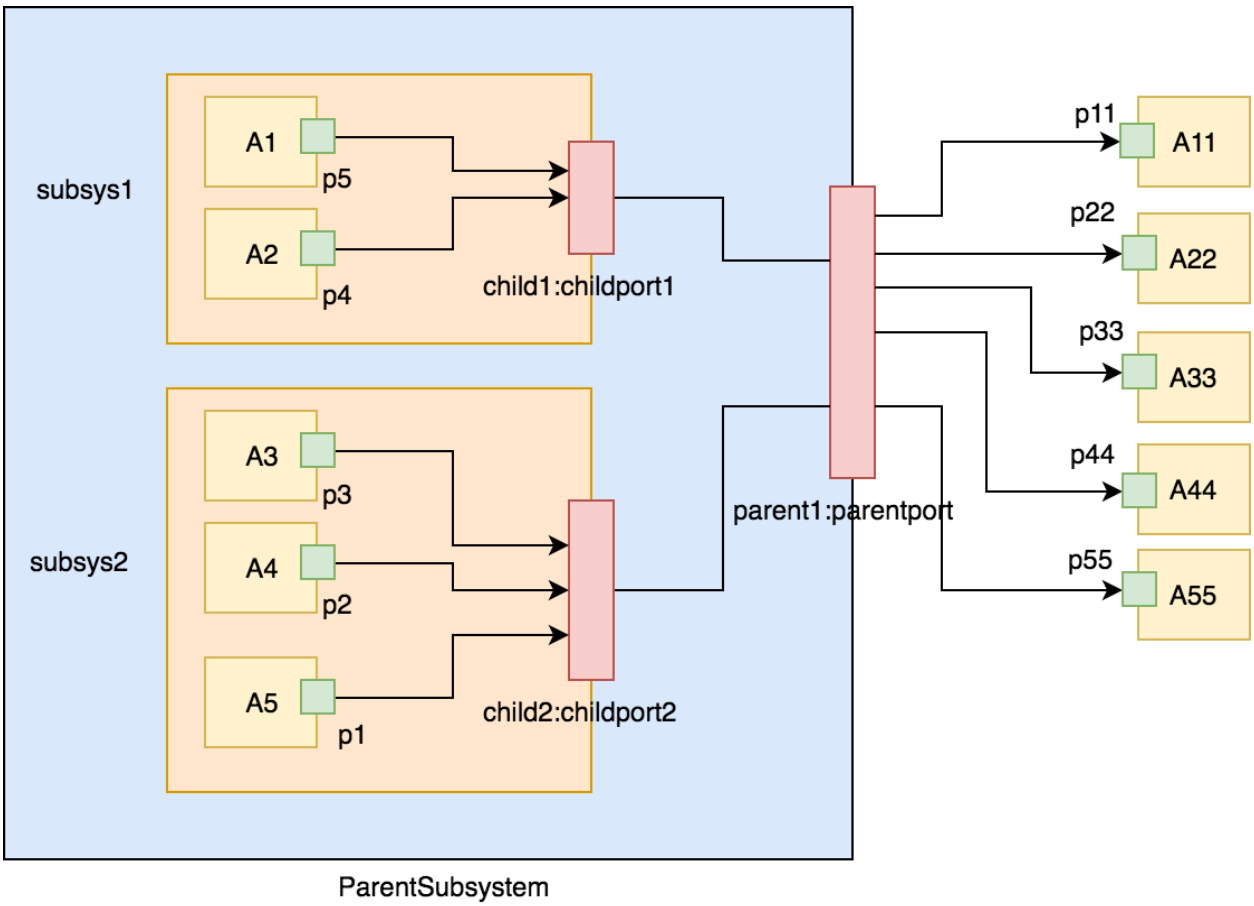
Example 1 Source Lang

```
//Component and Instance definition are declared same as explained above in
the document.
System sys1{
        //Subsystem Port definition
        SubPortType subchildport1{
                c1:Svc.Sched    //p1
                c2:Svc.Tlm              //p2
                c3:Fwd.Reg              //p3
        }

        SubPortType subchildport2{
                c4:Svc.Sched    //p4
                c5:Svc.Tlm              //p5
        }



        SubPortType subchildport3{
                c1:Svc.Sched    //p11
                c2:Svc.Tlm              //p22
                c3:Fwd.Reg              //p33
                c4:Svc.Sched    //p44
        }


        SubPortType subchildport4{
                c5:Svc.Tlm              //p55
        }

        SubPortType subsysparentport1{
                c6:subchildport1
                c7:subchildport2
        }
```

```
SubPortType subsysparentport2{
        c8:subchildport3
        c9:subchildport4
}



//Subsystem 1 definition
Subsystem subsys1{
        SubPortInstance child1:subchildport1;

        Mapping map1{
                A1.p1 to child1.c1
                child1.c2 to A2.p2
                A3.p3 to child1.c3
        }

        Topology topo1{
                //some topology defined
        }
}


//Subsystem 2 definition
Subsystem subsys2{
        SubPortInstance child2:subchildport2;

        Mapping map2{
                A4.p4 to child2.c4
                A5.p5 to child2.c5
        }

        Topology topo2{
                //some topology defined
        }
}


//Subsystem 3 definition
Subsystem subsys3{
        SubPortInstance child3:subchildport3;

        Mapping map3{
                child3.c1 to A11.p11
                A22.p22 to child3.c2
                child3.c3 to A33.p33
                child3.c4 to A44.p44
        }

        Topology topo3{
                //some topology defined
        }
}
```

```
        //Subsystem 4 definition
        Subsystem subsys4{
                SubPortInstance child4:subchildport4;

                Mapping map4{
                        A55.p55 to child4.c5
                }

                Topology topo4{
                        //some topology defined
                }
        }


        //Parent Subsystem 1 definition
        Subsystem subsysparent1{
                SubPortInstance parent1:subsysparentport1;

                Mapping map11{
                        parent1.c6 to subsys1.child1
                        parent1.c7 to subsys2.child2
                }

                Topology topo11{
                        //some topology defined
                }
        }


        //Parent Subsystem 2 definition
        Subsystem subsysparent2{
                SubPortInstance parent2:subsysparentport2;

                Mapping map22{
                        parent2.c8 to SG3.child3
                        parent2.c9 to SG4.child4
                }

                Topology topo22{
                        //some topology defined
                }
        }


        Topology topo1{
                subsysparent1.parent1 -> subsysparent2.parent2
        }
}
```

Example 2: Subport connecting directly with port

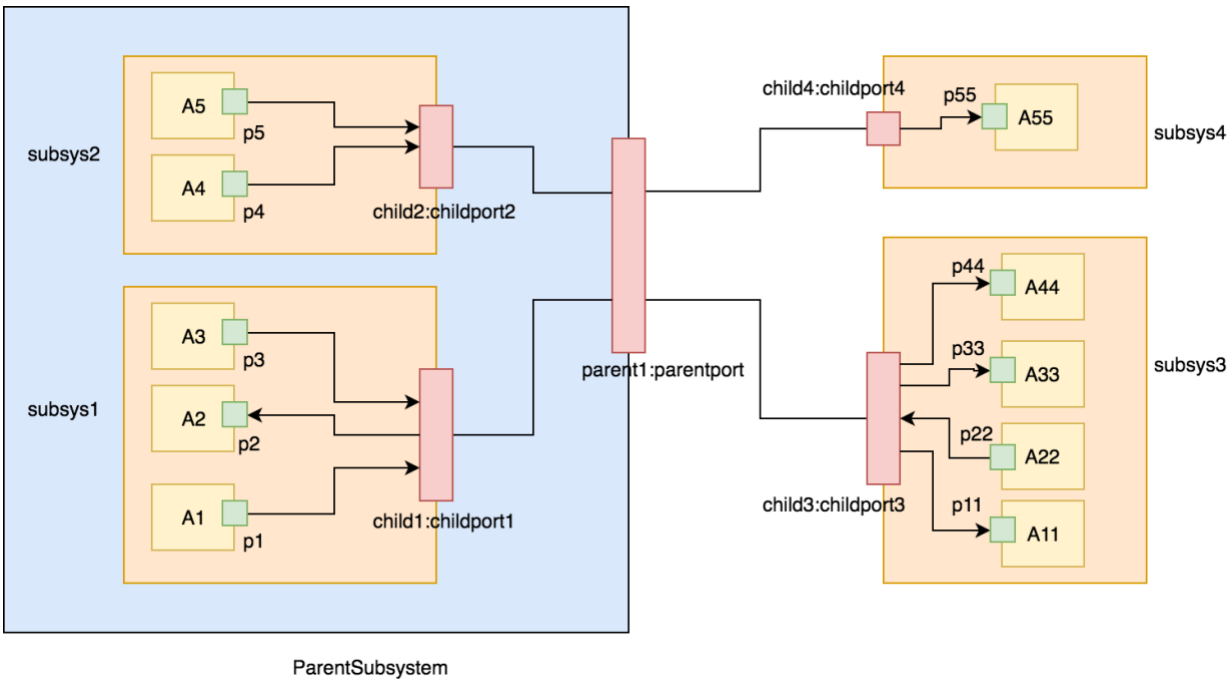


Example 2 Source Lang

```
//System Definition
System sys1{
        //Instance definition for A1-A5 and A11-A55
        //Subsystem port definition
        subPortType childport1{
                c5:Svc.Sched
                c4:Svc.Tlm
        }
        subPortType childport2{
                c3:Svc.Sched
                c2:Svc.Sched
                c1:Svc.Sched
        }

        subPortType parentport{
                c6:childport1
                c7:childport2
        }

        Subsystem subsys1{
                SubPortInstance child1:childport1

                Mapping map1{
                        A1.p5 to child1.c5
```

```
                A2.p4 to child1.c4
        }

        Topology topo1{
                //some topology defined
        }
}


Subsystem subsy2{
        SubPortInstance child2:childport2

        Mapping map2{
                A3.p3 to child2.c3
                A4.p4 to child2.c4
                A5.p5 to child2.c5
        }

        Topology topo2{
                //some topology defined
        }

}

Subsystem parentsubsystem{
        SubPortInstance parent1:parentport;

        Mapping map22{
                parent1.c6 to subsys1.child1
                parent1.c7 to subsys2.child2
        }

        Topology topo22{
                //some topology defined
        }
}

//Topology of the top view
Topology parentTopo{
        parentsubsystem.parent1 -> A11.p11
        parentsubsystem.parent1 -> A22.p22
        parentsubsystem.parent1 -> A33.p33
        parentsubsystem.parent1 -> A44.p44
        parentsubsystem.parent1 -> A55.p55
}
}
```

Example 3: Subsystemport connecting with subsystemport of child type



Example 3 Source Lang

```
//Component and Instance definition are declared same as explained above in
the document.
System sys1{
        //Subsystem Port definition
        SubPortType subchildport1{
                c1:Svc.Sched    //p1
                c2:Svc.Tlm              //p2
                c3:Fwd.Reg              //p3
        }

        SubPortType subchildport2{
                c4:Svc.Sched    //p4
                c5:Svc.Tlm              //p5
        }



        SubPortType subchildport3{
                c1:Svc.Sched    //p11
                c2:Svc.Tlm              //p22
                c3:Fwd.Reg              //p33
                c4:Svc.Sched    //p44
        }


        SubPortType subchildport4{
                c5:Svc.Tlm              //p55
        }

        SubPortType subsysparentport1{
                c6:subchildport1
                c7:subchildport2
```

```
}

//Subsystem 1 definition
Subsystem subsys1{
        SubPortInstance child1:subchildport1;

        Mapping map1{
                A1.p1 to child1.c1
                child1.c2 to A2.p2
                A3.p3 to child1.c3
        }

}

//Subsystem 2 definition
Subsystem subsys2{
        SubPortInstance child2:subchildport2;

        Mapping map2{
                A4.p4 to child2.c4
                A5.p5 to child2.c5
        }
}


//Subsystem 3 definition
Subsystem subsys3{
        SubPortInstance child3:subchildport3;

        Mapping map3{
                child3.c1 to A11.p11
                A22.p22 to child3.c2
                child3.c3 to A33.p33
                child3.c4 to A44.p44
        }
}


//Subsystem 4 definition
Subsystem subsys4{
        SubPortInstance child4:subchildport4;

        Mapping map4{
                A55.p55 to child4.c5
        }
}


//Parent Subsystem 1 definition
Subsystem subsysparent1{
        SubPortInstance parent1:subsysparentport1;

        Mapping map11{
                parent1.c6 to subsys1.child1
                parent1.c7 to subsys2.child2
        }
```

```
                Topology topo11{
                        //some topology defined
                }
        }

        Topology topo1{
                subsysparent1.parent1 -> subsys3.child3
                subsysparent1.parent1 -> subsys4.child4
        }
}
```

Project Hierarchy

- Project
    - Import other projects
    - Component definitions
    - Port Definitions
    - SubsystemPort Definitions
    - Data Type definitions
    - System
        - all instance declarations
        - Subsystem ParentSub //this is the subsystem which contains Sub1 and Sub2
            - Subsystem port declaration
            - Mapping
            - Topology (which consists instances inside subsystem) //Assuming that subsystems are inside this parent subsystem
        - Subsystem Sub1
            - Subsystem port declaration
            - Mapping
            - Topology (which consists instances inside subsystem)
        - Subsystem Sub2
            - Subsystem port declaration
            - Mapping
            - Topology
        - Other Topologies(if they exists)

Primitive Semantic Rules for Source Language

Below are the primitive rules for the source language:

1. Basic rules(Components/Subsystem/System):
    1. Component should have port definitions.
    2. Instance can only be of Component type.
    3. Subsystem port can be only of SubsystemPort type.

4. Component port can be only of Port type.
5. Connected ports should have same subsystemport type or port type definition.
6. A port can have one or more connection depending on number property.
7. PortType is mandatory attribute for port definition.
8. There can be only one "system" in the namespace.
9. Subsystems can have multiple subsystem inside it.
10. Constants should be declared before used as an identifier in Subsystem, System or Component.

2. Components and Ports:
    1. Active component should have at least one asynchronous port.
    2. Queue component should have at least one synchronous or guarded port.
    3. Queue component should have at least one asynchronous port.
    4. Active component can have Synchronous, Asynchronous, Guarded and Output kind of ports.
    5. Passive component can have Synchronous, Guarded and Output kind of ports.
    6. Queue component can have Synchronous, Asynchronous, Guarded and Output kind of ports.

Port Kind/Component	**Active**	**Passive**	**Queue**
Synchronous	YES	YES	YES
Asynchronous	YES	NO	YES
Guarded	YES	YES	YES
Output	YES	YES	YES

    7. Synchronous, Asynchronous and Guarded can only connect to Output port and vice-versa.

Port/Port	**Synchronous**	**Asynchronous**	**Guarded**	**Output**
Synchronous	NO	NO	NO	YES
Asynchronous	NO	NO	NO	YES
Guarded	NO	NO	NO	YES
Output	YES	YES	YES	NO

    8. If port has an attribute as a "role", then it should have same port type corresponding to it.

Port Role	**Port Type**
CmdRegistration	Fw.CmdReg

Telemetry	Fw.Tlm
CmdResponse	Fw.CmdResponse
LogEvent	Fw.Log
LogTextEvent	Fw.LogText
ParamGet	
ParamSetR	
TimeGet	Fw.Time
Cmd	Fw.Cmd

3. Connection and Topology Rules:
    1. There can be more than multiple topologies defined in the subsystem or system.
    2. Topologies can be inherited.
    3. If the topology is inherited, care must be taken that there should not be conflict like below:
        1. Port instance should not exceed it's maximum connection.
        2. Same port instance in both topologies should be of same type, kind and size.
    4. There can be multiple topologies inside a subsystem or system.
    5. "number" attribute is only applicable to output ports(not input ports).

Primitive Syntax Rules for Source Language

1. All block statement should have enclosed brackets.
2. All keywords should be case-insensitive.
3. All identifier should begin with letter followed by letter, digit , underscore or dash
4. All property values which have space in its value should be enclosed within double quotes
5. The file name which has space inside its path in "include" section should be enclosed in double quotes.
6. Context-free grammar:

    Context free Grammar

```
/**
 * NON-TERMINALS
 */

FPModel                          ::=     ( MODEL )? ( COMPONENT |
CONSTANT | INCLUDE | IMPORT | PORT_TYPE | SUBPORTTYPE | DATATYPE | ENUM
| SYSTEM )* <EOF>

MODEL                            ::=     <MODEL> <IDENTIFIER>
```

```
CONSTANT                ::=     <CONSTANT> ATTRIBUTE

INCLUDE                         ::=     <INCLUDE> <STRING_LITERAL>

IMPORT                          ::=     <IMPORT> ( <IDENTIFIER>
<FROM> )? <STRING_LITERAL>

COMPONENT               ::=     <COMPONENT> <IDENTIFIER>
COMPONENT_BLOCK

COMPONENT_BLOCK         ::=     <LBRACE> ( PORT | ATTRIBUTE )* <RBRACE>

PORT                            ::=     <PORT> <IDENTIFIER> <COLON> TYPE
PORT_BLOCK

PORT_BLOCK                      ::=     <LBRACE> ( ATTRIBUTE )* <RBRACE>

PORT_TYPE               ::=     <PORT_TYPE> <IDENTIFIER>
PORT_TYPE_BLOCK

PORT_TYPE_BLOCK         ::=     <LBRACE> ( ATTRIBUTE | ARGUMENT )*
( RET )? <RBRACE>

ARGUMENT                ::=     <ARG> <IDENTIFIER> <COLON> TYPE
<LBRACE> ( ATTRIBUTE )* <RBRACE>

RET                                     ::=     <RET> <LBRACE>
( ATTRIBUTE )* <RBRACE>

DATATYPE                ::=     <DATATYPE> <IDENTIFIER>

ENUM                            ::=     <ENUM> <IDENTIFIER> ENUM_BLOCK

ENUM_BLOCK                      ::=     <LBRACE> ( ATTRIBUTE )* <RBRACE>

SUBPORTTYPE                     ::=     <SUBPORTTYPE> <IDENTIFIER>
SUBPORTTYPE_BLOCK

SUBPORTTYPE_BLOCK       ::=     <LBRACE> ( ( <IDENTIFIER> <COLON> TYPE
| TYPE ) )* <RBRACE>

SYSTEM                          ::=     <SYSTEM> <IDENTIFIER>
SYSTEM_BLOCK

SYSTEM_BLOCK            ::=     <LBRACE> ( INSTANCE | SUBSYSTEM |
TOPOLOGY )* <RBRACE>

INSTANCE                ::=     <INSTANCE> <IDENTIFIER> <COLON> TYPE
INSTANCE_BLOCK

INSTANCE_BLOCK          ::=     <LBRACE> ( ATTRIBUTE )* <RBRACE>

SUBSYSTEM               ::=     <SUBSYSTEM> <IDENTIFIER>
SUBSYSTEM_BLOCK

SUBSYSTEM_BLOCK         ::=     <LBRACE> ( TOPOLOGY | SUBPORTINSTANCE |
MAPPING )* <RBRACE>
```

```
TOPOLOGY                ::=     <TOPOLOGY> <IDENTIFIER> TOPOLOGY_BLOCK

TOPOLOGY_BLOCK          ::=     <LBRACE> ( CONNECTOR <CONNECT>
CONNECTOR )* <RBRACE>

SUBPORTINSTANCE         ::=     <SUBPORTINSTANCE> <IDENTIFIER> <COLON>
TYPE

MAPPING                         ::=     <MAPPING> <IDENTIFIER>
MAPPING_BLOCK

MAPPING_BLOCK           ::=     <LBRACE> ( CONNECTOR <TO> CONNECTOR )*
<RBRACE>

TYPE                            ::=     <IDENTIFIER> ( <DOT>
<IDENTIFIER> )?

CONNECTOR               ::=     <IDENTIFIER> ( <DOT> <IDENTIFIER> )?
( <LBRACKET> <INTEGER_LITERAL> <RBRACKET> )?

ATTRIBUTE               ::=     <IDENTIFIER> <EQ> ( <INTEGER_LITERAL> |
<STRING_LITERAL> | <IDENTIFIER> )



/**
 * TOKENS
 */


/* WHITE SPACE */
<DEFAULT> SKIP : {
" "
| "\t"
| "\n"
| "\r"
}

/* COMMENTS */

<DEFAULT> MORE : {
"//" : IN_SINGLE_LINE_COMMENT
}

<IN_SINGLE_LINE_COMMENT> SPECIAL : {
<SINGLE_LINE_COMMENT: "\n" | "\r" | "\r\n"> : DEFAULT
}

<IN_SINGLE_LINE_COMMENT> MORE : {
<~[]>
}

/* RESERVED WORDS AND LITERALS */

<DEFAULT> TOKEN : {
```

```
<NAMESPACE: "namespace">
| <CONSTANT: "constant">
| <INCLUDE: "include">
| <IMPORT: "import">
| <COMPONENT: "component">
| <PORT: "port">
| <PORT_TYPE: "porttype">
| <ARG: "arg">
| <RET: "return">
| <DATATYPE: "datatype">
| <ENUM: "enum">
| <SUBPORTTYPE: "subporttype">
| <SYSTEM: "system">
| <INSTANCE: "instance">
| <SUBSYSTEM: "subsystem">
| <SUBPORTINSTANCE: "subportinstance">
| <TOPOLOGY: "topology">
| <MAPPING: "mapping">
| <FROM: "from">
| <TO: "to">
}

/* SEPARATORS */

<DEFAULT> TOKEN : {
<LBRACE: "{">
| <RBRACE: "}">
| <LBRACKET: "[">
| <RBRACKET: "]">
| <DOT: ".">
| <COLON: ":">
| <EQ: "=">
| <CONNECT: "->">
}

/* IDENTIFIERS */

<DEFAULT> TOKEN : {
<IDENTIFIER: <LETTER> (<LETTER> | <DIGIT> | <SPECIAL>)*>
| <#LETTER: ["A"-"Z","a"-"z"]>
| <#DIGIT: ["0"-"9"]>
| <#SPECIAL: ["-","_"]>
}

/* LITERALS */

<DEFAULT> TOKEN : {
<INTEGER_LITERAL: <DECIMAL_LITERAL> (["l","L"])? | <HEX_LITERAL>
(["l","L"])? | <OCTAL_LITERAL> (["l","L"])? | <FLOATING_POINT_LITERAL>
| <RANGE>>
| <#DECIMAL_LITERAL: ["1"-"9"] (["0"-"9"])*>
| <#HEX_LITERAL: "0" ["x","X"] (["0"-"9","a"-"f","A"-"F"])+>
| <#OCTAL_LITERAL: "0" (["0"-"7"])*>
| <#FLOATING_POINT_LITERAL: (["0"-"9"])+ "." (["0"-"9"])* (<EXPONENT>)?
(["f","F","d","D"])? | "." (["0"-"9"])+ (<EXPONENT>)?
(["f","F","d","D"])? | (["0"-"9"])+ <EXPONENT> (["f","F","d","D"])? |
(["0"-"9"])+ (<EXPONENT>)? ["f","F","d","D"]>
```

```
| <#EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+>
| <#RANGE: <DECIMAL_LITERAL> ".." <DECIMAL_LITERAL>>
| <STRING_LITERAL: "\"" (~["\"","\\","\n","\r"])* "\"">
}
```

References

- http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7416545