Well-formedness Checks for mbeddr C Code (https://goo.gl/0QIqeI)

```

module ValidationPatterns

/**
  Returns the operations of a component that are declared by its
  realized interfaces but not implemented actually.

  @param c the component
  @return the incomplete operations
 */
def getIncompleteOperations(c : Component) : Operation = {
  o := getProvidedOperations(c)
  assert o != getImplementedOperations(c)
  return o
}

/**
  Returns the operations that are declared by the interfaces of
  a component.

  @param c the component
  @return the declared operations
 */
private def getProvidedOperations(c : Component) : Operation = {
  p := c.contents
  assert p instanceOf ProvidedPort
  i := p.intf
  assert i instanceOf ClientServerInterface
  o := i.contents
  assert o instanceOf Operation
  return o
}

/**
  Returns the implemented operations of a component.

  @param c the component
  @return the implemented operations
 */
private def getImplementedOperations(c : Component) : Operation = {
  r := c.contents
  assert r instanceOf Runnable
  t := r.trigger
  assert t instanceOf OperationTrigger
  o := t.calledOperation
  assert o instanceOf Operation
  return o
}

/**
  Returns pairs of global variables which have the same name
  in the context of a module. The context is used to search for
  global variables that are either in the same module or one of them
  is in the given module and the other one is in some transitively imported
  module.

  @param m the module
  @return the pair of conflicting global variables
 */
def getGlobalVariablesWithSameName(m : Module) : (GlobalVariableDeclaration,
    GlobalVariableDeclaration) = {
  v1 := getModuleContents(m)
  v2 := getModuleContents(m)
  assert v1 instanceOf GlobalVariableDeclaration
  assert v2 instanceOf GlobalVariableDeclaration
  assert v1 != v2
  v1n := v1.name
  v2n := v2.name
  assert eval(v1n.equals(v2n))
  return (v1, v2)
} alt {
  v1 := getModuleContents(m)
  i := getAllModuleDependenciesReexported(m)
  v2 := getModuleContents(i)
  assert v1 instanceOf GlobalVariableDeclaration
  assert v2 instanceOf GlobalVariableDeclaration
  assert v1 != v2
  v1n := v1.name
  v2n := v2.name
  assert eval(v1n.equals(v2n))
  return (v1, v2)
}

/**
  Returns all reexported module dependencies of a module, which consist of
  (1) the direct dependencies of the module
  (2) the transitively imported modules where the imports have the reexport flag

  @param m the module
  @return all reexported module dependencies
 */
private def getAllModuleDependenciesReexported(m : Module) : Module = {
  return getModuleDependency(m)
} alt {
  i := getModuleDependency(m)
  return getModuleDependencyReexported+(i)
}

/**
  Returns the contents of a module

  @param m the module
  @return the module contents
 */
private def getModuleContents(m : Module) : IModuleContent = {
  c := m.contents
  assert c instanceOf Section
  return getSectionContents(c)
} alt {
  c := m.contents
  assert c not instanceOf Section
  return c
}

/**
  Returns the contents of a section.
  Contents may be directly contained or nested (potentially
  multiple times) in sections.

  @param s the section
  @return the section contents
 */
private def getSectionContents(s : Section) : IModuleContent = {
  c := s.contents
  assert c not instanceOf Section
  return c
} alt {
  c := s.contents
  assert c not instanceOf Section
  return getSectionContents(c)
}

/**
```

```
  Returns all module dependencies of a module, which consist of
  (1) the direct dependencies of the module
  (2) the transitively imported modules (no reexport is needed)

  @param m the module
  @return all module dependencies
 */
def AllModuleDependencies(m : Module) : Module = {
  return getModuleDependency+(m)
}

/**
  Returns the direct, reexported dependencies of a module.

  @param m the module
  @return the direct dependency
 */
private def getModuleDependencyReexported(m : Module) : Module = {
  i := m.imports
  assert i instanceOf DefaultGenericChunkDependency
  r := i.reexport
  assert r == true
  c := i.chunk
  assert c instanceOf Module
  return c
}

/**
  Returns the direct dependencies of a module.

  @param m the module
  @return the direct dependency
 */
private def getModuleDependency(m : Module) : Module = {
  i := m.imports
  assert i instanceOf DefaultGenericChunkDependency
  c := i.chunk
  assert c instanceOf Module
  return c
}

/**
  Enumerates function call which are part of a recursive function call chain.

  @param c the function call
 */
def recursiveFunctionCall(c : FunctionCall) : Void = {
  assert c == functionCallLink+(c)
}

/**
  Returns the function calls which are called in the function of a function call.
  This function is used to construct a function call graph.

  @param c the function call
  @return the target function calls
 */
private def functionCallLink(c : FunctionCall) : FunctionCall = {
  f := c.function
  assert f instanceOf Function
  s := f.body.statements
  assert s instanceOf ExpressionStatement
  e := s.expr
  assert e instanceOf FunctionCall
  return e
}
```