EDA User's Manual

Tool name (Version):

C++/SystemC Coding Rule

(ver1.1)

Overview

This document decides coding rules for unifying coding style aiming to develop hardware models efficiently and to promote reusing the models as defacto standard style in Renesas.

Related Documents

+ C/C++/SystemC Code Checker 1Team:System User's Manual (ver 1.16) [RS-EZ-DM-0310]

Introduction

■Purpose of this document

As electronic designs become increasingly large and complicated, RTL-based flows face a design productivity crisis. To address the limitations of a pure-RTL design flow, System Level Design based on C language is being spread. Especially, the main language of System Level Design is SystemC. SystemC is based on C++, nonprofit organization OSCI(Open SystemC Initiative) is managing and developing. SystemC is standardized as IEEE Std.1666 in 2005 and becomes defacto standard language of System Level Design for SoC. SystemC is supplied as C++ class library which has definitions of some functions for hardware description.

Also in Renesas, some hardware modeling techniques by using SystemC are being used widely such as system performance evaluation, early software development and generating RTL by high level synthesis. As RTL design flow is used mainly now in Renesas, It is need to actively promote C base design flow because of increasing design productivity.

In light of this situation, we consider that one of the most important things is using only one coding style for hardware modeling by C++/SystemC. Good coding style brings good effects such as follows.

+ It's easy to maintenance models by making design consistent
+ It's possible to increase interoperability and reusability
+ It's useful for improving readability and preventing code bugs

This document decides coding rules for unifying coding style aiming to develop hardware models efficiently and to promote reusing the models as defacto standard style in Renesas.

■Audience

This document is intended to address the needs of these designers involved in the development of SystemC Model:

+ Algorithm designers
+ System designers
+ Hardware and IP designers
+ Tool developers and support engineers

■Objectives and Organization of this document

Coding rules defined in this document are separated 4 categories.

1. Hint
   Rules of this category are difficult to define it quantitatively and to check it objectively. Thus every designer needs to read and understand them before design.

2. Common
   Rules of this category are possible to check automatically by some tools and must be obeyed by every designer regardless of a kind of developing model.

3. Simulation
   Rules of this category must be obeyed when designers develop the model for performance evaluation in system level.

4. Synthesis
   Rules of this category must be obeyed when designers develop the model of high level synthesis for generating RTL.

At first, we recommend you reading the rules included in Hint Category and Common category. Because these are basic rules and must be common knowledge by every designer who engages to develop models.

If you are developers of the model for performance evaluation in system level, refer to the rules of Simulation category.

If you are developers of the model of high level synthesis for generating RTL, refer to the rules of Synthesis category.

■Notation of rules

A notation example of rule:

Classification1	Classification2	Classification3

Title

Appropriate Example

Example code

Inappropriate Example

Example code

Comment

Details of rule

How to check

RuleName (1Team:System)

+ Classification 1

There are 4 kinds "Hint", "Common", "Simulation" and "Synthesis". Please refer to the above "Objectives and Organization of this document " for each meaning.

+ Classification 2

There are 2 kinds "MUST" and "Required". "MUST" means to should be always obeyed, and "Required" means to recommend to be obeyed.

+ Classification 3

There are 2 kinds "C++" and "SystemC". "C++" means coding rule regardless of SystemC, and "SystemC" means coding rule regarding SystemC.

+ Title

This mentions the abstraction of rule. The index of title means the category of coding rule following keywords.

Knowledge	… HNT
Common	… COM
Performance evaluation	… MOD
High level synthesis	… SYN

+ Appropriate Example

This mentions an example code appropriate for coding rule.

+ Inappropriate Example

This mentions an example code inappropriate for coding rule.

+ Comment

This mentions the detail of rule.

+ How to check

This mentions the way for checking whether or not you obey coding rule. Basically, if there exists the tool which check the rule, the tool name is shown.

The description such as "Rulename (1Team:System)" means that you can check the coding rule by the rule named "Rulename" of 1Team:System which is intoroduced at below "Tool Names".

■Tool Names

This document uses following tool names without the refusal. Please inquire of the person in charge about details of each tool.

+ **1Team:System**	C/C++/SystemC code checker developed in Atrenta Co.
+ **CtoS**	High level synthesis tool developed in Cadence Co.
+ **GCOV**	Code coverage test tool developed in GNU. Uses it with gcc/g++.
+ **Purify**	Dynamic code analysis tool developed in IBM Co. It can detect memory leak, memory destruction, etc.
+ **Valgrind**	Dynamic code analysis tool of freeware. It can detect memory leak, memory destruction, etc.

List of Contents

Hint	Required	C++

HNT1 Do not disclose internal implementation

Appropriate Example

```
class A {
public:
   int get_attribute() {
      return attribute;
   }
   void set_attribute(int value) {
      if(attribute < 0) {
         cerr << "do not set minus value" << endl;
      } else {
         attribute = value;
      }
   }

private:
   int attribute;
   ...
};
void B::func() {
   A a;
   a.set_attribute(-100);
   ...
```

Inappropriate Example

```
class A {
public:
   int attribute;
   ...
};
void B::func() {
   A a;
   a.attribute = -100;
```

```
    ...
```

Comment

By defining "Object" which integrates data and method that operates it, the detailed specification and the structure in the object can be hidden from the outside. This is called encapsulation.Using public method is the only way to access the internal data from outside, so this enables to improve independecy of object.

Actively using this encapsulation enables to reduce the infulence of changing the specification in the object for outside, and to improve maintainability, developping efficiency and reusability of software.

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT2 Using a lot of member variables in class is bad. Reduce them as much as possible

Appropriate Example

```
// Member variable, member function are necessity minimum.
class Base {
protected:
   int base_attr;
   ...
public:
   void base_func();
   ...
};
class A : public Base {
private:
   int attr1;
   int attr2;
   ...
public:
   int func1();
   int func2();
   ...
};
```

Inappropriate Example

```
// Member variable, member function are unnecessary many.
class A {
private:
   int base_attr;
   int attr1;
   int attr2;
   ...
public:
   void base_func();
   int func1();
```

```
    int func2();
    ...
};
```

Comment

When there are many member variables/functions, as they are object for verification, completion of the program becomes difficult. Also bug will be difficult to be fixed when detected.

Consider if it can be divided using inheritance. For example it is one way first to make only class, which is used often and make inheritedderived class and implement them extension function. Or if it can be divided into several classes, you should consider it.

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT3　Relative variables should be defined by "class"/"struct"

Appropriate Example

```
struct {
   int color;
   int hsize;
   int vsize;
   int type;
} pic_info;
```

Inappropriate Example

```
int color;
int hsize;
int vsize;
int type;
```

Comment

"class"/"struct" is a very effective way to put related variables together. When related variables coincide, actively use "class"/"struct" to put them together.

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT4 Do not use too many members in "class"/"struct"

Appropriate Example

```
struct Color {
   int color_R;
   int color_G:
   int color_B;
};
struct Size {
   int hsize;
   int vsize;
};
struct Pos {
   int x;
   int y;
};
struct ...
```

Inappropriate Example

```
struct MainData {
   int color_R;
   int color_G:
   int color_B;
   int hsize;
   int vsize;
   int x;
   int y;
   ...
};
```

Comment

Too Many members in "class"/"struct" make the interpretation and improvement of the program difficult. You cannot do anything if the number of members truly necessary is many. But otherwise you should reconsider if it can be divided into several "class"/"struct" or if you can approach completely different way.

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT5 Gather the common code

Appropriate Example

```
int get_area(int addr)
{
   if ( addr >= 0 && addr < 0x80000000 )
      return 0;
   else if ( addr >= 0x80000000 && addr < 0xC0000000 )
      return 1;
   else if ( addr >= 0xC0000000 && addr < 0xF0000000 )
      return 2;
   else return 3;
}
void main_rountine()
{
   ...
   if ( direction == WRITE ) {
      area = get_area(addr);
      ...
   }
   else {
      area = get_area(addr2);
      ...
```

Inappropriate Example

```
int get_area(int addr)
{
   if ( addr >= 0 && addr < 0x80000000 ) return 0;
   else if ( addr >= 0x80000000 && addr < 0xC0000000 )
      return 1;
   else if ( addr >= 0xC0000000 && addr < 0xF0000000 )
      return 2;
   else return 3;
}
```

```
void main_rountine()
{
   ...
   if( direction == WRITE ) {
      if ( addr >= 0 && addr < 0x80000000 )
         area = 0;
      else if ( addr >= 0x80000000 && addr < 0xC0000000 )
         area = 1;
      else if ( addr >= 0xC0000000 && addr < 0xF0000000 )
         area = 2;
      else area = 3;
      ...
   }
   else {
      if ( addr2 >= 0 && addr2 < 0x80000000 )
         area2 = 0;
      else if ( addr2 >= 0x80000000 && addr2 < 0xC0000000 )
         area2 = 1;
      else if ( addr2 >= 0xC0000000 && addr2 < 0xF0000000 )
         area2 = 2;
      else area2 = 3;
      ...
```

Comment

If you do not gather the common part, it makes interpretation and improvement of the program difficult.Also it makes bug hard to be treated. It doesn't matter when doing trial and error or working on code during bug inspection, but at final version, they should be modified.

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT6 Avoid the complex expression such as deep nest of condition operator("? :")

Appropriate Example

```
if ( size == BYTE )
   data &= 0x000000FF;
else if ( size == WORD ) {
   if ( addr%4 == 0   )
      data &= 0x0000FFFF;
   else data &= 0xFFFF0000;
}
else
   data &= 0xFFFFFFFF;
```

Inappropriate Example

```
data &= ( size == BYTE ) ? 0x000000FF
         : ( size == WORD ) ? ( addr%4 == 0 ) ? 0x0000FFFF : 0xFFFF0000
         : 0xFFFFFFFF;
```

Comment

Condition operator "? :" is effective description when describing simple branch condition. However avoid using it when the condition is so complicated that it become deep nest. In this case, describing by using "if " description etc. makes it easier to interpret.

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT7 The specification which uses several member functions complexly should be avoided

Appropriate Example

```
class CFileCopy {
  class CallBack {
  protected:
    void* mMisc;
  public:
    CallBack(){ mMisc = NULL; };
    virtual ~CallBack();
    virtual bool UserCancel(void*) = 0; //accept of cancel entry during copy
    virtual void DirDown(const char*,void*) = 0; //Directory down
    virtual void DirUp(const char*,void*) = 0; //Directory rise
    virtual void FileCopyStart(const char*,void*) = 0; //Start copying a file
    virtual void FileCopyEnd(const char*,void*) = 0; //Complete copying a file
    void SetMisc(void* ioMisc){ mMisc = ioMisc; }; //Set of Misc argument
  };
public:
  CFileCopy();
  virtual ~CFileCopy();
  int SetParam(const char*,const char*,bool,CallBack*); //Set parameter
  int ExecCopy(void); //copy
};
class MyFileCopyCallBack : public CFileCopy::CallBack {
public:
  virtual bool UserCancel(void*);
  virtual void DirDown(const char*,void*);
  virtual void DirUp(const char*,void*);
  virtual void FileCopyStart(const char*,void*);
  virtual void FileCopyEnd(const char*,void*);
};
static void TestMain(void)
{
  CFileCopy theFC;
  MyFileCopyCallBack theCallBack;
  if (0 != theFC.SetParam("XXX","YYY",true,&theCallBack)) {
    ...
  }
  else {
    if (0 == theFC.ExecCopy()) {
        ...
    }
  }
}
```

Inappropriate Example

```
class CFileCopy {
    typedef bool (*UserCancelFunc)(const CFileCopy&,void*);
    typedef void (*DirDownFunc)(const CFileCopy&,const char*,void*);
    typedef void (*DirUpFunc)(const CFileCopy&,const char*,void*);
    typedef void (*FileCopyStartFunc)(const CFileCopy&,const char*,void*);
    typedef void (*FileCopyEndFunc)(const CFileCopy&,const char*,void*);
    ...
public:
    CFileCopy();
    virtual ~CFileCopy();
    void SetSourceDirName(const char*); //Set directory name of copy source
    void SetDestDirName(const char*);     //Set directory name of copy destination
    void SetCreateDir(bool); //Specify whether to create or not at copy destination directory doesn't exist.
    void SetUserCancelFunc(UserCancelFunc); //Set callback function to accept entry of user cancel of copy
being done.
    void SetUCFuncMisc(void*); //set second callback argument to accept entry of user cancel.
    void SetDirDownFunc(DirDownFunc); //Set callback function to call at directory down.
    void SetDDFuncMisc(void*); //Set thired callback argument at directory down.
    void SetDirUpFunc(DirUpFunc); //call when directory rise.
    //Set callback function.
    void SetDUFuncMisc(void*); //Set thried callback argument at directory rise.
    void SetFileCopyStartFunc(FileCopyStartFunc); //Set callback function to call when starting a file copy.
    void SetFCStartFuncMisc(void*); //Set thired callback argument at starting a file copy.
    void SetFileCopyEndFunc(FileCopyEndFunc); //Set callback function to call at finishing a file copy.
    void SetFCEndFuncMisc(void*); //Set thired callback argument at finishing a file copy.
    void ExecCopy(void); //Copy
    int GetErrorCode(void); //Get error code
};
static void TestMain(void)
{
    CFileCopy theFC;
    theFC.SetSourceDirName("XXX");
    theFC.SetDestDirName("YYY");
    theFC.SetCreateDir(true);
    theFC.SetUserCancelFunc(MyUserCancelFunc);
    theFC.SetUCFuncMisc(NULL);
    theFC.SetDirDownFunc(MyDirDownFunc);
    theFC.SetDDFuncMisc(NULL);
    theFC.SetDirUpFunc(MyDirUpFunc);
    theFC.SetDUFuncMisc(NULL);
    theFC.SetFileCopyStartFunc(MyFileCopyStartFunc);
    theFC.SetFCStartFuncMisc(NULL);
    theFC.SetFileCopyEndFunc(MyFileCopyEndFunc);
    theFC.SetFCEndFuncMisc(NULL);
    if(0 != theFC.GetErrorCode()){
        ...
    }
    else {
        theFC.ExecCopy();
        if (0 == theFC.GetErrorCode()) {
            ...
        }
    }
}
```

Comment

When you make a specification in which several member functions use each other complexly, the program may not operate as you expect and cause bugs. Also when you try to modify a completed program it sometimes cause bugs. It is ideal to design to complete one task with one (or few) member function.

Many member functions are problem not only because it is difficult to see entire picture, but also it has many combinations of member function use, which case lot of work to verify them later. Using several functions for one task seems "advanced" technique. However most of the case come from such reason as lack of design or wrong self-content or self-defense to cover the fact that the specification is not well-fixed etc.

In the inappropriate example, intentionally complex specification is used for one task(copy file here).You should rather make it done by one member function and try not to make problems happen such as wrong execution order or missing of calling necessary function. When the number of callback function setting is many, preparation for executing the copy itself is a lot of task. And there are lot of things to check such as which is the must call and in what order to call them etc.

As you see in the adapted example, as for callback function, make class exclusive for callback and have them all function. As for parameter setting, gather all to make one member function. (It solves the problem of missing of call and call order)

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT8 The specification which create/remove/copy the object complexly should be avoided

Appropriate Example

```
class CParent {
public:
  CParent() : child(NULL) {
    ...
  }
  ~CParent() {
    if (child)
      delete child;
  }
  ...
private:
  CChild* child;
  ...
};
void main() {
  CParent* parent = new CParent;
  ...
  delete parent;
}
```

Inappropriate Example

```
class CParent {
public:
  CParent() : child(NULL) {
    ...
  }
  ~CParent() {
  }
  ...
public:
  CChild* child;
  ...
};
void main() {
  CParent* parent = new CParent;
  ...
  if (parent->child)
    delete parent->child;
  delete parent;
}
```

Comment

When you use several objects, consider how these objects will be related in advance and during coding, always check how they are related. Degree of complexity is taken differently by each designger. Consider a case third party refer to it, make it simple dependence.

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT9 The order of function definition should be up or down regularly

Appropriate Example

```
void sub_sub_func() {
   ...
}
void sub_func() {
   ...
   sub_sub_func()
   ...
}
void my_func() {
   ...
   sub_func();
   ...
```

Inappropriate Example

```
void sub_func() {
...
sub_sub_func();
   ...
}
void my_func() {
   ...
   sub_func();
   ...
}
void sub_sub_func() {
   ...
}
```

Comment

When defining function (method) in source file, make the definitions order (call relation) have rules, and make them up or down regularly. Define function called from uppermost function first, and follow called function definition, or the reverse.

In the appropriate example, definition is made in a order of sub_sub_func -> sub_func -> my_func,which coincide with bottom-up. When you define it like this, it becomes easier to refer definition place.

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT10 Define main code in source file(*. cpp file) and declare the methods and variables in header file(*.h file)

Appropriate Example

```
*** sample.h ***
class sample
{
private:
   void main_func();
public:
   sample();
   ...
};
*** sample.cpp ***
void sample::main_func() {
      ...
}
sample::sample() {
      ...
}
...
```

Inappropriate Example

```
*** sample.h ***
class sample
{
private:
   void main_func() {
      ...
   }
public:
   sample() {
      ...
   }
```

```
    ...
  };
```

Comment

Do not write function definition within the header file. In a header file only declare function and describe main code in source file(extension is c or cpp).

When main code is described in a header file, not only compile time of the source file to include the header increases, but also there will be a problem in security. However, simple function such as which can be described in one line is an exception.

As note necessary, it is disable to divide template class into header file and source file. As template class definition has not been finished before gets types information when it is instantiated, the souce file that has undefined types of template class causes compile error. However, when the types of template class argument are limited to several kinds, you can avoid compile error by defining the using types on the top of source file as following formula.

```
template class class_name <type1>;
template class class_name <type2>;
...
template <int TYPE>
void sample<TYPE>::func()
{...}
```

How to Check

There is no tool to check it. Check it by designer.

Hint	Required	C++

HNT11　Do not use ordinary name for variable/function/macro name (ex."WORD")

Appropriate Example

```
#define min_data(a, b) ¥
...
min_data( left_data, right_data);
```

Inappropriate Example

```
#define min(a, b) ¥
...
min(left_data, right_data);
```

Comment

There is no problem when function names and macro names are NOT reserved word ruled by C/C++ language specification. However when you use all-too common names, depending on a compiler, they may coincide with the names that already defined which lead to error when compiling. Therefore use names, which properly describe the contents even though they may be a bit long.

In compiler of VisualC++ of Windows, "min" and "max" are macros that are already defined. So if you execute coding as showed in the inappropriate example, compile error occurs.

How to Check

There is no tool to check it. Check it by designer.

Common	MUST	SystemC

COM1 Put sentry code to header file because of avoiding load header twice or more

Appropriate Example

```
#ifndef __CPU_H
#define __CPU_H
// header body
#endif//__CPU_H
```

Inappropriate Example

```
// header body
```

Comment

Be sure to put include guard code at the beginning of a header file, as you see in the appropriate example. This is because if same header file is called several times during a compile of a source code, method that is declared/defined in the header file will be called twice and this causes compile error. Macro name to define shall be "__FILENAME_H". It is recommended to write "name" by comment in #endif to put on the end of the file.

How to Check

MRC-19.15 (1Team:System)

Common	MUST	SystemC

COM2 Do not access the area which is not allocated

Appropriate Example

```
int dat[3];
...
dat[0] = 1;
dat[1] = 2;
dat[2] = 3;
```

Inappropriate Example

```
int dat[3];
...
dat[1] = 1;
dat[2] = 2;
dat[3] = 3;    // out of range
```

Comment

Do not access array index which is not allocated. It is a matter of course, but sometimes when dependant on OS, array area more than allocated by use is allocated. So the user can't notice the bug with is accessing the area which is not allocated. Also, when variables are used in index, in a bug in which the variable is not initialized and access, the variable is dependant on OS and maybe initialized to 0.This is also a bug difficult to find. You should pay enough attention to value of array index.

How to Check

MRC-1.2.3 (1Team:System)

Purify/Valgrind

Common	MUST	SystemC

COM3 Take care that in the condition code of "if" description is not "=" but "=="

Appropriate Example

```
if ( cond == 1 )
   ...
```

Inappropriate Example

```
if ( cond = 1 )
   ...
```

Comment

It may happen that you were intended to do comparison in a condition expression, but a type miss make it assignment. It becomes bug difficult to find, because it is not a violation of syntax and does not become error at compiling. You should be more careful when coding.

How to Check

MRC-13.2 (1Team:System)

Common	MUST	SystemC

COM4　Surround by brace when you define the formula by macro

Appropriate Example

```
#define SHIFT(n, m) (n << m)
...
val = SHIFT(fifo_size, 3) * 4;        //It means quadruplicate SHIFT result.
```

Inappropriate Example

```
#define SHIFT(n, m) n << m
...
val = SHIFT(fifo_size, 3) * 4;        //It means 12 shift fifo_size value.
```

Comment

When you define by using macro, depending on the priority of the operator used in the formula to define and that is used in the macro, the calculation may mean different than you intended. In the inappropriate example, because of the priority of shift operation defined in the macro and multiplication operation specified in the formula with the SHIFT macro, it becomes fifo_size << (3*4), which is not intended. Therefore, be sure to surround by brace when n you define the formula by macro, so that you get intended operation with any operator.

How to Check

MRC-19.10 (1Team:System)

Common	MUST	SystemC

COM5 Return the variable absolutely from the function with return value

Appropriate Example

```
int myfunction( int add )
{
  if ( add > 0 ) {
    add--;
  }
  else
    add = -1;
  return add;
}
```

Inappropriate Example

```
int myfunction( int add )
{
  if ( add > 0 ) {
    add--;
    return add;
  }
  else
    add = -1;
}
```

Comment

Warning appears when there is a path that doesn't return return-value in a function whose return value is not void type. Execution is possible, but when the processing is passed to the path, it often cause bug because the return value is not guaranteed.

How to Check

MRC-16.8 (1Team:System)

Common	MUST	SystemC

COM6 Write all conditions with "default" in "switch-case" statement

Appropriate Example

```
switch( code ) {
   case HEADER : ...
   case PID       : ...
   case DATA     : ...
   case EOP       : ...
   default:
      fprintf(stderr, "Error: Invalid code (%x)¥n", code);
}
```

Inappropriate Example

```
switch( code ) {
   case HEADER : ...
   case PID       : ...
   case DATA     : ...
   case EOP       : ...
   // No default sentence.
}
```

Comment

When all conditions are included in case specification in switch description, default line is not necessary. However, it may happen that a condition is not included in any case statement due to specification change and temporary description. For that reason, write "default" in case statement.

How to Check

MRC-15.3 (1Team:System)

Common	MUST	SystemC

COM7 Like "foo" and "Foo" (the different point is just upper/lower) such a variable name should not be used

Appropriate Example

```
int val, data;
```

Inappropriate Example

```
int Val, val;           // only difference is upper/lower
int val, value;         // On first sight, it can easily misunderstood as same variables
int addr_val, addrval;  // Evade distinguish by underscore
int val, va1;            // Evade similar shape character such as l and 1
```

Comment

Misread of variables and functions during analysis may reduce efficiency. Therefore, do not use names that look similar. The title points out example of names that can only be distinguished by upper or lower. Other cases such as "names which completely include character string", "names which can only be distinguished by underscore" also should not be used. Furthermore characters which appear same at first look such as "l" and "1" should not be used.

How to Check

emb.ccpp.bp.8 (1Team:System)

Common	MUST	SystemC

COM8　Do not use "union structure"

Appropriate Example

```
// Do not use union
```

Inappropriate Example

```
union {
   struct {
      unsigned char L;
      unsigned char H;
   } b;
   unsigned short w;
} HL;
```

Comment

Union means structure in which two or more variables share same memory region and if defined as a member it can have different type of value. However the value it can have is only one member value and a compiler does not understand which member's value it has. So users have to use while managing which kind of value is assigned in the union structure. Because the Union sometimes cause endian problem, do not use Union.

How to Check

MRC-18.4 (1Team:System)

Common	MUST	SystemC

COM9 Three expressions in "for" formula, use only loop control variable not use the other variables

Appropriate Example

```
for (i=0 ; i<MAX ; i++) {
   j++;
   ...
}
```

Inappropriate Example

```
for (i=0 ; i<MAX ; i++, j++) {
   ...
}
```

Comment

Three expressions in "for" formula can have some processing by using comma formula, but should have only processing related to loop control. The others should be described in "for" block.

How to Check

MRC-13.5 (1Team:System)

Common	MUST	SystemC

COM10　Do not use "goto" expression

Appropriate Example

```
// Do not use "goto" expression
```

Inappropriate Example

```
NEXT_ACTION:
...
goto NEXT_ACTION;
```

Comment

Abuse of "goto" may make it difficult to follow the process flow and make it difficult for a third party and the developer himself to interpret. Therefore do not use "goto" expression.

How to Check

MRC-14.4 (1Team:System)

Common	MUST	SystemC

COM11 The calculation that may be error dynamically (for example division and residual with 0 used in right-hand) should be checked in advance

Appropriate Example

```
if (y != 0 ) {
   ans1 = x/y;
   ans2 = x%y;
```

Inappropriate Example

```
ans1 = x/y;
ans2 = x%y;
```

Comment

Except when it is clearly not 0, calculate after checking the right side number of division or residual is not 0. If you do not, it may cause 0 division error at execution.

How to Check

There is no tool to check it. Check it by designer.

Common	MUST	SystemC

COM12 Do not use "bit-field"

Appropriate Example

```
// Do not use bit-field.
```

Inappropriate Example

```
struct S {
   unsigned int bit1:1;
   unsigned int bit2:1;
};
```

Comment

Following operations of bit- field differ depending on a compiler.

1. "int" type bit-field without specification of sign is treated as sign or not.
2. Allocation order of bit-field within a unit.
3. Border of memory region unit

Like this a code can be environment-dependant. Therefore basically do not use bit-field.

How to Check

There is no tool to check it. Check it by designer.

Common	MUST	SystemC

COM13　Use "read" and "write" method when you refer or set value of port/signal

Appropriate Example

```
sc_signal<sc_uint<32> > pa;
sc_uint<32> buf;
...
buf = !pa.read();
...
pa.write(buf);
```

Inappropriate Example

```
sc_signal<sc_uint<32> > pa;
sc_uint<32> buf;
...
buf = pa;
...
pa = buf;
```

Comment

When you refer and set value of port/signal, use "read" and "write" method. If you do not use "read" method for reference, it may cause compile error by some kind of compiler.

And using these methods enables to improve readability and to reduce potential bug by clearly distinguishing using port/signal from using the other types.

How to Check

sysc.dsgn.2 (1Team:System)

Common	MUST	SystemC

COM14 Do not forget to set static sensitivity list to SC_METHOD/SC_THREAD process

Appropriate Example

```
...
SC_METHOD(main_method);
dont_initialize();
sensitive << clk.pos();
...
```

Inappropriate Example

```
...
SC_METHOD(main_method);
dont_initialize();
// sensitive missing of specifying
...
```

Comment

When you register thread/ method process in constructor, be careful not to miss the description of the static sensitivity list. In case of called by different thread/method process, the description of sensitive is unnecessary. However if not called by anywhere, not executed even once, there may be a case when more than expected number of analysis man hour will be taken in order not to be error during compiling and execution.

When you specify several conditions by sensitivity list, be careful not to miss the description. For example , when sensitive signal are many control signals such as interrupt signal, be careful not to forget to add them to the sensitivity list when you add model.

How to Check

sysc.lang.57 (1Team:System)

sysc.lang.67 (1Team:System)

sysc.rtl.1 (1Team:System)

Common	MUST	SystemC

COM15　Must call "wait()" method from infinite loop of SC_THREAD process

Appropriate Example

```
// member method of SC_THREAD
...
while ( 1 ) {
   out.write(current_data);
   wait();
}
```

Inappropriate Example

```
// member method of SC_THREAD
...
while ( 1 ) {
   out.write(current_data);
   // There is no wait
}
```

Comment

Be sure to call wait() method in infinite loop of SC_THREAD/SC_CTHREAD process. If not, the simulation will not go on. If there is a condition branch such as "if" sentence, be sure to call wait() at all each branch paths always. But it may happen that 1Team:System reports false error about the rule sysc.lang.58. The case of describing all branch cases and calling wait() on each path by "if"　and "else-if" sentence, 1Team:System may report false error because of specific restriction that it can't detect all conditions covered. For avoiding this trouble, you should include "else" sentence for branch condition or should call common wait() outside branch condition.

In SC_METHOD process, wait() cannot be called , so do not describe infinite loop itself. Be careful about these difference between SC_THREAD and SC_METHOD.

How to Check

sysc.lang.58 (1Team:System)

Common	MUST	SystemC

COM16 Use recommended description styles not depending on SystemC version

Appropriate Example

```
sc_string str;
...
sensitive << clk.pos();
...
out.initialize();
...
sc_start(100);
```

Inappropriate Example

```
std::string str;
...
sensitive_pos( clk );
...
sc_initialize(out);
...
sc_cycle(100);
```

Comment

There are some syntaxes which are supported by old SystemC version but not recommended to use by latest SystemC version. For compatibility, you should not use these.

Non-recommended syntaxes are shown below that are typical items excerpted from IEEE Std 1666-2005 Annex C (SystemC Language Reference Manual).

Class:

sc_bit, sc_string, sc_simcontext, sensitive_pos, sensitive_neg

Function:

wait_until(), sc_cycle(), sc_initialize(), sc_event::notify_delayed(), sensitive_pos(), sensitive_neg(), sc_module::end_module(), sc_simulation_time(), sc_set_default_time_unit(), sc_start(double), vcd_trace_file::sc_set_vcd_time_unit()

How to Check

sysc.lang.46 (1Team:System)

sysc.migrate_to_21.* (1Team:System)

sysc.migrate_to_ieee_1666.* (1Team:System)

Common	MUST	SystemC

COM17 Do not access port in constructor

Appropriate Example

```
...
Cmod( sc_module_name name_) : sc_module(name_)
, in_port("in_port")
, out_port("out_port")
{
   // tmp = in_port.read();   // Do not access
   out_port.initialize(1);     //Use initialize for initialization
```

Inappropriate Example

```
...
Cmod( sc_module_name name_) : sc_module(name_)
, in_port("in_port")
, out_port("out_port")
{
   tmp = in_port.read();   // error
   out_port = 1;           // error
```

Comment

When constructor is executed, port is not connected yet, so when you execute access at this time it becomes execution error. Do not refer port and use initialize method for initialization.

How to Check

sysc.lang.61 (1Team:System)

Common	MUST	SystemC

COM18 Define variable by sc_signal type which is accessed from two or more method

Appropriate Example

```
class sample : public sc_module {
   ...
   sc_signal<sc_uint<32> > my_var;
   sc_uint<32>   wr_data;
   sc_uint<32>   rd_data;
   ...
   sample(sc_module_name name) : sc_module (name) {
      SC_METHOD( write_my_var );
      sensitive << clk.pos();
      SC_METHOD( read_my_var );
      sensitive << clk.pos();
      ...
   }
   void write_my_var( void ) {
      wr_data++;
      my_var.write(wr_data);
   }
   void read_my_var(void) {
      rd_data = my_var.read();
   }
   ...
};
```

Inappropriate Example

```
class sample : public sc_module {
   ...
   sc_uint<32>   my_var;
   sc_uint<32>   wr_data;
   sc_uint<32>   rd_data;
   ...
   sample(sc_module_name name) : sc_module (name) {
       SC_METHOD( write_my_var );
       sensitive << clk.pos();
       SC_METHOD( read_my_var );
       sensitive << clk.pos();
       ...
   }
   void write_my_var( void ) {
       wr_data++;
       my_var.write(wr_data);
   }
   void read_my_var(void) {
       rd_data = my_var.read();
   }
   ...
};
```

Comment

When a variable is accessed by several processes at a same time, the order of their execution is not defined. So if you use sc_uint etc. for the variable result may change depends on the simulation environment. In this case, when variable is signal (sc_signal), delta delay fix, which is equivalent to nonblocking assigning in Verilog-HDL is executed. By this when read and write are executed at the same time, write can read previous old value without fail.

However, using sc_signal does not guarantee order of simultaneous execution of write and write. In this case it is necessary to functionally guarantee execution order of simultaneous execution by using exclusive access control.

How to Check

sysc.struct.5 (1Team:System)

Common	MUST	SystemC

COM19　Do not use "sc_bit"

Appropriate Example

```
sc_signal<bool> dma_req_n;
dma_req_n.write(0);
```

Inappropriate Example

```
sc_signal<sc_bit> dma_req_n;
dma_req_n.write(0);     // compile error
```

Comment

It is recommended to use bool type and not use sc_bit type. Due to issue of SystemC library, compile error may occur when assigned to signal with sc_bit type.

How to Check

sysc.migrate_to_21.23 (1Team:System)

Common	MUST	SystemC

COM20 Clock(the instance of sc_clock) should be instanced in top model(testbench) or sc_main

Appropriate Example

```
int sc_main(int ac, char **av)
{
   sc_clock sclk("sclk", 10, SC_NS);
   ...
}
```

Inappropriate Example

```
class Ccpg : sc_module
{
public:
   sc_clock sclk;
   Ccpg(sc_module_name name) : sc_module(name)
   , sclk("sclk", 10, SC_NS)
   {
      ...
   }
   ...
};
```

Comment

For clock generation, which is done using sc_clock class, normally define in sc_main class and adapt it to input port of each class to use. However, when frequency changes due to register setting such as CMT, implementation in which candidate clocks are already in the class and switch them by selector is possible. However, if clock is instanced in the class, execution errors happen in some case depending on a compiler. According to the language reference manual, clock should be instanced in sc_main. Therefore sc_clock should be instanced either sc_main or top model (testbench).

How to Check

There is no tool to check it. Check it by designer.

Common	MUST	SystemC

COM21　Do not use port binding by that signal definition order

Appropriate Example

```
class Ctmu : sc_module
{
public:
  sc_in <bool> clk;
  sc_in <bool> reset;
  sc_in <bool> pms_n;
  sc_out<bool> pwait_n;
  ...
};
class top : sc_module
{
public:
  ...
  top(sc_module_name name) : sc_module(name) {
    Ctmu *tmu = new Ctmu("tmu");
    tmu->clk ( pclk );
    tmu->reset ( reset );
    tmu->pms_n ( pms_tmu_n );
    tmu->pwait_n ( pwait_tmu_n );
    ...
  }
  ...
};
```

Inappropriate Example

```
class Ctmu : sc_module
{
public:
  sc_in <bool> clk;
  sc_in <bool> reset;
  sc_in <bool> pms_n;
```

```
    sc_out<bool> pwait_n;
    ...
};
class top : sc_module
{
public:
    ...
    top(sc_module_name name) : sc_module(name) {
        Ctmu *tmu = new Ctmu("tmu");
        tmu << pclk << reset << pms_tmu_n << pwait_tmu_n;
        ...
    }
    ...
};
```

Comment

There are two ways to write wire connection between modules, same as Verilog-HDL. One way is to write ports and wires respectively(binding by name) and the other way is to make correspondence with connected wire by order of ports (positional binding). The latter has an advantage of less writing volume but it often causes human error and is decided not to be used as a language specification of SystemC in the future. Therefore use only the former binding by name.

How to Check

sysc.style.2 (1Team:System)

Common	MUST	SystemC

COM22 Use "sensitive << " not "sensitive ()"

Appropriate Example

```
SC_THREAD(main_routine_thread);
dont_initialize();
sensitive << clk.pos();
SC_METHOD(int_routine_method);
dont_initialize();
sensitive << intreq_tmu_n << intreq_dmac_n;
```

Inappropriate Example

```
SC_THREAD(main_routine_thread);
dont_initialize();
sensitive(clk.pos());
SC_METHOD(int_routine_method);
dont_initialize();
sensitive(intreq_tmu_n);
sensitive(intreq_dmac_n);
```

Comment

There are two ways to write static sensitivity list of thread/method process. The one way is "sensitive <<...,"in which shift operator is overloaded. The another way is "sensitive(...)",which is a way to specify it by argument. Use the former way as it has an advantage of doing several events by connecting shift operators.

Moreover, do not use below descriptions too because of becomming non-recommended in SystemC Language Reference Manual.

sensitive_pos(), sensitive_neg(), sensitive_pos, sensitive_neg

How to Check

sysc.migrate_to_21.9 (1Team:System)

Common	MUST	SystemC

COM23　Process(method/thread) should not be called from other process or function

Appropriate Example

```
...
SC_THREAD(main_routine_thread);
sensitive << clk.pos();
SC_THREAD(sub_routine_thread);
sensitive << clk.pos();
...
void main_routine_thread()
{
    // Do not call sub_routine_thread in this
}
void sub_routine_thread()
{
    // Do not call main_routine_thread in this
}
...
```

Inappropriate Example

```
...
SC_THREAD(main_routine_thread);
sensitive << clk.pos();
SC_THREAD(sub_routine_thread);
sensitive << clk.pos();
...
void main_routine_thread()
{
    sub_routine_thread();
}
...
```

Comment

Process (SC_THREAD or SC_METHOD) can be called through other process or function, however, do not do this as it makes reference relation complicated.

How to Check

sysc.lang.59 (1Team:System)

Common	MUST	SystemC

COM24 Remove all warning messages during compiling with "-Wall"(when using Visual C++, Level3(/W3)) option

Appropriate Example

```
% gmake
g++ -Wall cpu.cpp ...
g++ -Wall dmac.cpp ...
// measure when warning appears
```

Inappropriate Example

```
% gmake
g++   cpu.cpp ...
g++   dmac.cpp ...
// do not attach –Wall and do not measure when warning appears
```

Comment

When code is compiled, be sure to attach "-Wall"(when using Visual C++, Level3(/W3)) option and remove warning appears. The warning is very important and lead to potential bug prevention. When you remove, shortsighted modification without investigating the cause is useless. Removing after understanding the each warning.

How to Check

There is no tool to check it. Check it by designer.

Common	MUST	C++

COM25 If you use template with nesting, add one more spaces between two consecutive ">" characters

Appropriate Example

```
sc_in< sc_uint<16> > in1;
```

Inappropriate Example

```
sc_in<sc_uint<16>> in1;
```

Comment

If you use template with nesting, add one more spaces between two consecutive '>' characters. As ">>" is taken as the right shift operator, it causes a compile error.

How to Check

sysc.lang.8 (1Team:System)

Common	Required	C++

COM26 Keep naming rules for readability

Appropriate Example

```
// 1. data member
int m_val;
// 3. define name
#define BUS_WIDTH 32
enum command {CMD_READ, CMD_WRITE, ...};
// 5. add thread or method at the end of name
void main_thread();
void reset_metod();
...
SC_THREAD(main_thread);
SC_METHOD(reset_method);
```

Inappropriate Example

```
// 2. do not use name started "sc_"
int sc_val;
// 4. If all characters are upper (or lower), same string
int Size, size;
```

Comment

To improve readability, keep the following naming rules.

1. To each data member of class, add "m_" keyword as prefix, or add "m" keyword as prefix and set next word as capital letter
2. Class name and instance name should not start "sc_" keyword
3. The constant variable name (which is defined by #define) and the enumeration of enum type is in uppercase
4. Do not define a variable whose name is same as the existing variable name except the difference in lower or upper case.
5. Include "thread"/"method" keyword in the name of THREAD/METHOD function

How to Check

No.2 ... sysc.style.3 (1Team:System)

No.4 ... emb.ccpp.bp.8 (1Team:System)

Check others by designer.

Common	MUST	C++

COM27　Take care of "cast"

Appropriate Example

```
// 3. add cast
int data = val;
short data2 = (short)data;
// 4. remove redundant cast
unsigned int addr, size;
int length;
addr = size*4 + (unsigned int)length;
// 6. cast with result of bitwize operator
unsigned short b = 0xffff;
unsigned int a = (unsigned short)(b << 2);
```

Inappropriate Example

```
// 1. different class cast
class A {...};
class B {...};
A a;
B *b = (B *)&a;
// 2. different pointer cast
int *data;
char *val;
data = (int *)val;
// 3. need cast
int data = val;
short data2 = data;
// 4. redundant cast
unsigned int addr, size;
int length;
addr = (unsigned int)( size*4 + (unsigned int)length);
// 5. do not use void* type cast
my_struct = (void *)calc_struct;
// 6. non cast with result of bitwize operator
```

```
    unsigned short b = 0xffff;
    unsigned int a = b << 2;    // "b << 2" needs cast "unsigned short", or "b" needs cast "int"
```

Comment

If you need to assign the variable to other type variable, it is very useful to use "cast". But it is difficult to detect the bug in it. Add the concrete rules as follows.

1. Do not use "cast" to assign different class type forcibly
2. Do not use "cast" to assign between other pointer type
3. Add "cast" just in case when you assign the variable to smaller size variable
4. Do not use "cast" for redundant assignment
5. Do not use void* type "cast"
6. When you use bit operator (~ and <<) to variable whose type is smaller than "int" type(char, short, etc), cast the result of operation to the type, or operate after cast the variable to int type when you purposely assume integer promotion.

How to Check

No.2 ... MRC-11.4 (1Team:System)

No.5 ... MRC-11.3B (1Team:System)

No.6 ...MRC-10.5 (1Team:System)

Check others by designer.

Common	Required	SystemC

COM28　Bitwise operation should not be performed on signed integer type

Appropriate Example

```
unsigned int data = 0x00010000;
data >>= 16;
```

Inappropriate Example

```
int data = -1;
data >>= 16;
```

Comment

Bitwise operation performed on signed integer type is not recommended.

A bitwise operation for signed integer types makes a result depends on the internal description of compiler. The top bit is used for signed description. Especially the case of right shift operation, you should be careful the empty bits are filled in the value of signed bit(which is called Arithmetic shift).

How to Check

MRC-12.7 (1Team:System)

Common	MUST	SystemC

COM29　Bit shifting should not cause data loss

Appropriate Example

```
unsigned int out;
unsigned int val_int = 0x00001000;
unsigned char val_char = 0x01;
out = val_int << 16;
out = (unsigned char)(val_char << 4);
out = (unsigned int)val_char << 16;
```

Inappropriate Example

```
unsigned int out;
unsigned int val_int = 0x00001000;
unsigned char val_char = 0x01;
out = val_int << 32;
out = val_char << 4;
out = val_char << 16;
```

Comment

Bit shifting should not cause data loss.

The data loss is that parts of the number is lost because the size of the variable that stores the operation result is smaller than the number of operation results.

When left shift operation on char types and short types, the operation is performed after implicitly casting to integer types. So integer type casting is needed to the variable using the type if the left shift operation is over the bit width of the type, and the char or short type casting is needed if the operation is not over the bit width of the type.

How to Check

There is no tool to check it. Check it by designer.

Common	MUST	SystemC

COM30 Do not compare between pointer variables. When need to compare, the both pointers must indicate the consecutive data area

Appropriate Example

```
int *p1, *p2;
int arr1[10];
p1 = arr1[index1];
p2 = arr2[index2];
if (p1 >= p2) //addressing elements of the same array
   printf("p1 is greater¥n");
```

Inappropriate Example

```
int *p1, *p2;
int arr1[10], arr2[10];
p1 = arr1;
p2 = arr2;
if (p1 >= p2) //not addressing arrays
   printf("p1 is greater¥n");
```

Comment

It is very difficult to find the bug about comparison between pointer variables. When having no choice to do it, attend not to be applied to objects of pointer types that do not address arrays or elements of the same array.

The value of pointer types is automatically assigned by processing system, so the relation is nonsense between pointers which indicate different arrays.

How to Check

MRC-17.3 (1Team:System)

Common	MUST	SystemC

COM31 Arithmetic operation on constant expressions must not lead to wraparound or sign change

Appropriate Example

```
unsigned int u = 1+2;
char c3 = 127-12;
```

Inappropriate Example

```
unsigned int u = 1-2;
char c3 = 127+12;
```

Comment

Arithmetic operations on constant expressions must not lead to wraparound or sign change.1Team:System's rule 'MRC-12.11A' enables the check. About arithmetic operations on variable expressions, this rules can't work, so need to choose a type which has enough bit width.

How to Check

MRC-12.11A (1Team:System)

Common	MUST	C++

COM32　Do not use address of auto variable for function return value

Appropriate Example

```
int* mod::my_func()
{
  if (condition == 1)
    return ...
  else
    return &m_val; // m_val is data member
}
```

Inappropriate Example

```
int* mod::my_func()
{
  int val = 0;
  if (condition == 1)
    return ...
  else
    return &val;
}
```

Comment

When function return value is pointer, do not return the address of auto variable(local variable) of the function. The auto variable is valid inside only the function, and escape the fucntion scope, the address becomes invalid. If function must return pointer, use the address of dynamic memory allocated by new or malloc, data member or static variable which are secured effective.

How to Check

MRC-17.6B (1Team:System)

Common	MUST	C++

COM33　Do not redefine the same name variable in different scope

Appropriate Example

```
//Not use same name variable
for (int i = 0 ; i < x_size ; i++) {
   ...
   if (data[i] > max) {
      int j = data2;
      ...
   }
}
```

Inappropriate Example

```
// Variable i is used in lower scope
for (int i = 0 ; i < x_size ; i++) {
   ...
   if (data > max) {
      int i = data2;
      ...
   }
}
```

Comment

You can redefine the same name variable in different scope, but this decreases readability and makes it difficult to maintain the source code. This issue is not only for the inside function, but also for global variable and data member of class. Take care of not naming the same variable name used in different scope even if the usage of variable is similar.

How to Check

MRC-5.2 (1Team:System)

Common	MUST	C++

COM34　Do not use signed variable and unsigned variable mixed in description

Appropriate Example

```
unsigned char flag1, flag2;
...
flag1++;
if (flag2 > flag1) {
   ...
```

Inappropriate Example

```
unsigned char flag1;
char flag2;
...
flag1++;
if (flag2 > flag1) {
   ...
```

Comment

Signed char type variable is from -128 to 127 value, and unsigned char type variable is from 0 to 255. In "Appropriate code", if flag1 and flag2 are 127, "flag2(127) > flag1(128)" is false, but in "Inappropriate code", "flag2(127) > flag1(-128)" is true. This is not expected behavior.

When you need to use signed variable and unsigned variable mixed in description, you should use cast for matching sign type.

How to Check

sysc.lang.63 (1Team:System)

Common	MUST	C++

COM35 Do not use trivial logic expression in condition statement

Appropriate Example

Refer to "Inappropriate example"

Inappropriate Example

```
if ( a=4 )    // assignment is always true. Maybe it is a == 4
if ( b>128 && b<1) // This expression is always false
```

Comment

For improving readability and maintainability, remove the condition expression which is always approved and remove the condition expression with processing body which is not always approved.

However, "while (1)" description needed as describe infinite loop in the SC_THREAD/SC_CTHREAD process is out of this rule.

How to Check

MRC-13.7A (1Team:System)

Common	MUST	C++

COM36 Do not use "long" and "long double" type

Appropriate Example

```
//32bit OS
int val;  // 4byte variable
long long int val2; // 8byte variable
//64bit OS
int val;  // 4byte variable
long long int val2; // 8byte variable
```

Inappropriate Example

```
//32bit OS
long val;  // 4byte variable
//64bit OS
long val;  // 8byte variable
```

Comment

The size of "long" and "long double" is depend on OS, and do not use this type variable. This is example for x86 type Linux.

Type	32bit OS	64bit OS
char	1 byte	1 byte
short	2 byte	2 byte
int	4 byte	4 byte
long	4 byte	8 byte
float	4 byte	4 byte
double	8 byte	8 byte
long long	8 byte	8 byte
long double	12 byte	16 byte

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++
Synthesis	Required	

COM37　When even the type is same, a declaration not related is written in another line

Appropriate Example

```
int hsize, vsize;      // picture size
int num_of_loop;       // the number of loop
```

Inappropriate Example

```
int hsize, vsize, num_of_loop;
```

Comment

Even if it is the same variable type, consider the way it is used and relation and if it is not related at all, write it in another line. Because it makes it easy to analogy what kind of variable it is from the declaration. Also it makes easier to put the comment.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM38 Nest level should not be too deep

Appropriate Example

```
if ( status == IDLE ) {
   check_size(size);
   set_mode(mode);
   func(addr, data, size);
}
else if (status == ACTIVE) {
   ...
```

Inappropriate Example

```
if ( status == IDLE ) {
   if (addr < 0x1000) {
      if (size == LONG) {
         if (mode == MODE1) {
            ...
         }
         else if (mode == MODE2) {
            ...
         }
         ...
      }
      else if (size == WORD) {
         ...
      }
      ...
   }
   else if (addr < 0x2000) {
      ...
   }
   ...
}
else if (status == ACTIVE) {
    ...
```

Comment

When description volume to be described in the block is big such as "if" description, it become difficult to find the following "else" sentence or what the "if" description itself is trying to process. Therefore when code is going to be deep, seek better way for entire description to be easy to understand, by such way as executing a set of process within a different function.

How to Check

MRC-1.1.2 (1Team:System)

Simulation Synthesis	MUST Required	C++

COM39 Use "enum" not "#define" as much as possible when defining related some constant numbers

Appropriate Example

```
enum ecountry {
   ENGLAND, FRANCE, ...
} country;
enum eweek {
   SUNDAY, MONDAY, ...
} day;
...
if (country == ENGLAND) {
   ...
if (day == MONDAY ) {
   ...
if (country == FRANCE ) {   // can be checked by tool
   ...
```

Inappropriate Example

```
#define ENGLAND 0
#define FRANCE   1
...
#define SUNDAY   0
#define MONDAY   1
...
int country, day;
...
if (country == ENGLAND ) {
   ...
if (day == MONDAY ) {
   ...
if (country == FRANCE ) {      // difficult to check by tool
   ...
```

Comment

Use enumeration type when defining related constant number such as set. When you define related constant number by enum, tool can check incorrect usage. Macro name defined by #define will be operated macro expansion in preprocess phase and will not be the name compiler process. However, enum constant number defined, as enum declaration will be a name that compiler process. Names that complier processes are easy to debug because they can be referred when symbolic debugging.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM40　Basically do not assign value to enumeration of enum type

Appropriate Example

```
enum etag{ E1, E2, E3, E4};
enum elist{ L1=10, L2=11, L3=12};       // when you have to specify, specify all.
```

Inappropriate Example

```
enum etag{ E1, E2=10, E3, E4=11};       // E3 and E4 are also 11.
```

Comment

When you do not assign initial value to enumeration, the value will be the 1 plus immediately preceding enumeration value(The first enumeration value is 0). If you assign initial value for some enumerations and do not assign for others, you may carelessly specify same value and it may cause unexpected behavior. Therefore, basically do not assign the value to enumeration of enum type . When you have to assign value, assign only for first or for all members.

How to Check

MRC-9.3 (1Team:System)

Common	Required	C++

COM41　When you initialize "array" or "struct", use nesting brace to show the structure

Appropriate Example

```
int arr1[2][3] = {{0, 1, 2}, {3, 4, 5}};
...
static struct record {
   char* name;
   int data;
} list[2] = {{"abc", 1}, {"def", 2}};
```

Inappropriate Example

```
int arr1[2][3] = {0, 1, 2, 3, 4, 5};
...
static struct record {
   char* name;
   int data;
} list[2] = {"abc", 1, "def", 2};
```

Comment

In initialization of "array" or "struct", at least one pair of brace is needed, you should better use nesting brace to show the structure that makes it easy to understand how the initialization data is set.

How to Check

MRC-9.2 (1Team:System)

Common	Required	C++

COM42 Do not define member variable just which can define as local variable

Appropriate Example

```
class sample {
   // Member variable
   int fifo_size;
   ...
public:
   // Member method
   void push_data( char *data ) {
      for( int loop=0 ; loop<10 ; loop++ )
         data_fifo[fifo_size++] = data[loop];
   }
   char pop_data() {
      return data_fifo[fifo_size--];
   }
   ...
};
```

Inappropriate Example

```
class sample {
   // Member variable
   int fifo_size;
   int loop;    // Not necessary to make it member method.
   ...
public:
   // Member method
   void push_data( char *data ) {
      for( loop=0 ; loop<10 ; loop++ )
         data_fifo[fifo_size++] = data[loop];
   }
   char pop_data() {
      return data_fifo[fifo_size--];
   }
```

```
    ...
};
```

Comment

Define only variables which need to be data member. When range of use of the variable is closed in one member method and is not needed to memorize the value, it should be local variable. A data member which is used in several members during the development should be changed to local variable if it came to be used by only one method finally.

If you neglect it, that may become obstacle when bug occurs or extending function.

How to Check

sysc.c_cpp.13 (1Team:System)

Common	Required	C++

COM43　Use ">", ">=", "<", "<=" in checking loop counter not use "==" and "!="

Appropriate Example

```
void func() {
   int i;
   for (i=0 ; i<9 ; i+= 2) {
…
```

Inappropriate Example

```
void func() {
   int i;
   for (i=0 ; i!=9 ; i+=2) {
…
```

Comment

Use ">", ">=", "<", "<=" in checking loop counter not use "==" and "!=".

Because it may be a infinite loop by skipping the escape condition when amount of change of loop counter is not 1.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM44　Do not compare the "TRUE" value in condition statement

Appropriate Example

```
#define FALSE 0
// func1 may return value other than 0 and 1
void func2() {
   if (func1() != FALSE ) {
      ...
   or
   if (func1()) {
      ...
```

Inappropriate Example

```
#define TRUE 1
// func1 may return value other than 0 and 1
void func2() {
   if (func1() == TRUE) {
      ...
```

Comment

In the C language, "TRUE" is indicated value other than 0 and not necessarily 1.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM45 Do not allocate huge memory statically

Appropriate Example

```
char *mem_buf = NULL;
mem_buf = new char [640*480*3*2];
if (mem_buf == NULL) {
   fprintf(stderr, "Cannot allocate the memory.¥n");
   exit(0);
}
...
```

Inappropriate Example

```
char mem_buf[640*480*3*2];
...
```

Comment

Depending on OS or environment for usage, when memory size to allocate is too big, sometimes allocating statically is not possible. Therefore you should allocate dynamically and write to check if it is allocated or not.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM46 Do not write "code clone"

Appropriate Example

```
void delete_data( famliy_type* family ) {
   for ( unsigned int i=0 ; i<family->size() ; i++ ) {
      if ( family.data[i] != NULL ) {
         delete family->data[i];
         family->data[i] = NULL;
      }
   }
}
...
delete_data( &children );
delete_data( &parent     );
```

Inappropriate Example

```
for ( unsigned int i=0 ; i<children.size() ; i++ ) {
   if ( children.data[i] != NULL ) {
      delete children.data[i];
      children.data[i] = NULL;
   }
}
for ( unsigned int i=0 ; i<parent.size() ; i++ ) {
   if ( parent.data[i] != NULL ) {
      delete parent.data[i];
      parent.data[i] = NULL;
   }
}
```

Comment

Code clone is similar or same part in a source code. Code clone is created in a program text by programming by "copy and paste" and by intentional writing to repeat same processing. Generally, code clone is one of factors that makes software maintenance difficult. For example, when a bug is found in a code clone, the developer has to check all corresponding code clones and ,if necessary, have to modify all the code clone similarly. Especially in a large-scale system, where development is often done by group, it is very difficult for one developer to check sub system and consistently modify all code clone.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM47　Do not write too long history comment

Appropriate Example

```
// 1 Initial entry
// 2 Change bus I/F
```

Inappropriate Example

```
// 1 Initial entry
// 1.1 Add register I/F
// 1.2 Data bus I/F bug fix
// 1.3 change internal FIFO spec.
// 2.0 Change bus I/F
// 2.1 add enable signal
// 2.2 bug fix about address width
```

Comment

History code is useful information and it is important to in-line it to code. However when version-up is done many times and added history comment become hundreds of lines, the code turns to be difficult to read. Therefore you should take measure such as to simplify past history.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM48　Avoid using the global variable/function

Appropriate Example

```
// int g_value;      // Do not define global variable
class sample
{
  int m_value;
  ...
  void func() {
    data = m_value << 2;
    ...
  }
};
```

Inappropriate Example

```
int g_value;         // Define/use global variable
class sample
{
  ...
  void func() {
    data = g_value << 2;
    ...
  }
};
```

Comment

When C++ is used, global variables make program decoding and improvement difficult and also it makes difficult to take countermeasure for bug when it occurs. To search a global variable, such as where it is read and where it is rewritten, you have to look for wider area compared to local variable. Therefore bug damage related to global variable is serious. Also considering model to be developed will be operated combined with a model developed by a third party, this problem tends to increase. However use of global variable and function is not prohibited as there is a case where only global variable and function can solve, or use of global variable and function make maintenance easy.

The rule sysc.lang.81 of 1Team:System can detect the use of global variable only.

How to Check

sysc.lang.81 (1Team:System)

Common	Required	C++

COM49　Remove unused code

Appropriate Example

```
void func(void) { // remove argument
  ...
```

Inappropriate Example

```
void func(int arg) {
  // argument "arg" is not used.
  ...
```

Comment

Not used codes should be removed because of decreasing Maintainability.

Example cases of easily remaining dead code are by changing in specification and by adding debug code temporarily.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM50 Output of the debugging message should be easily controlled by using the macro etc

Appropriate Example

```
#include <cstdarg>
void d_printf(char *format, ... )
{
#ifdef DEBUG
   va_list argptr;
   va_start(argptr, format);
   vprintf(format, argptr);
   va_end(argptr);
#endif
}
...
int func() {
   ...
   d_printf("return value is %d¥n", value);
   return value;
}
```

Inappropriate Example

```
int func() {
   ...
   printf("return value is %d¥n", value);
   return value;
}
```

Comment

When developing a model, sometimes print internal status by using printf() temporarily to check the behavior. This is a very important measure but sometimes it continues to output even after the model development is finished. Sometimes it becomes necessary again after it is removed. Therefore you should create the gadget that can control whether output or not message for debug. It is desirable for them to be controlled at once.

The appropriate example is to control messages by attaching DEBUG macro or not.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++
Synthesis	Required	

COM51 Error/Warning message should be followed by the position of source code

Appropriate Example

```
printf("<Warning: Specified size is not supported. (%s : %d line)>¥n", __FILE__, __LINE__);
=>
...
Warning: Specified size is not supported. (forest.h : 384 line)>
```

Inappropriate Example

```
printf("<Warning: Specified size is not supported. >¥n");
=>
...
Warning: Specified size is not supported. >    // difficult to know where from
```

Comment

Error/Warning message may be output for cases when use of model is wrong or when there is illegal access from other model. If the source file name and line number follows the error/warning message, taking measure can be easier. In the appropriate example, macro to print file name and line number on printf() is used. assert function etc. is also useful measure.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM52 Do not open namespace in header file

Appropriate Example

```
#include "myheader.h"
...
int val = myname::mydata;
```

Inappropriate Example

```
#include "myheader.h"
using namespace myname;
...
int val = mydata;
```

Comment

Name space is to prevent identification names to be unidentifiable by naming spaces which include type, variable and function. Therefore after space name and two colons(::), specify variable and function. Specifying the space name can be omitted, which is a status that name space is open. By this, it is not necessary to specify the space which is made to open.

However when you open namespace in a header file which is used by unspecified majority, it will be opened in all file which include the file. To avoid problem from it, do not open namespace in header file. However within source file (.cpp file) it will not include in other file, so there is no problem to open namespace at a developer's discretion.

How to Check

There is no tool to check it. Check it by designer.

Simulation Synthesis	MUST Required	C++

COM53 Do not use "TAB"

Appropriate Example

```
if (num == MAX) {
   for (loop=0 ; loop<10 ; loop++ ) {
      switch( addr ) {
         case BYTE      : size = 1; break;
         case WORD      : size = 2; break;
         case LONGWORD: size = 4; break;
...
```

Inappropriate Example

```
if (num == MAX) {
[TAB]for (loop=0 ; loop<10 ; loop++ ) {
[TAB][TAB]switch( addr ) {
[TAB] [TAB][TAB]case BYTE[TAB][TAB]: size = 1; break;
[TAB] [TAB][TAB]case WORD[TAB][TAB]: size = 2; break;
[TAB] [TAB][TAB]case LONGWORD[TAB] : size = 4; break;
...
```

Comment

Although using tab enables to justify lines effectively than using space, change of environment and setting of the user cause different tab width setting, which may cause disruption of the line, which has been justified. This does not happen when you use space. For this reason use space instead of tab to justify lines.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM54 In a part to declare "class"/"struct", write each data member in one line with comment

Appropriate Example

```
struct {
    int hsize;     // height size ( 8-1024)
    int vsize;     // width size ( 8-1024)
    int format;    // 0: RGB, 1:YUV444, 2:YUV422, 3:YUV420
    ...
```

Inappropriate Example

```
struct {
    int hsize, int vsize, int format, ...
```

Comment

In a part to declare "class"/"struct", put comment for each data member. Therefore, write each data member in one line. You should name it so that one can tell contents of the data. If you also describe range and candidate of the value, it will increase maintainability.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM55 The brief comment should explain contents not operation (Bad example: /*split by n* /, Good example:/* calculate the average*/)

Appropriate Example

```
// calculate the average
sub_func(total, size);
```

Inappropriate Example

```
// call sub_func function with total and size arguments
sub_func(total, size);
```

Comment

Comment is worthwhile when it explains the source contents from other side of the view, and not worthwhile to write comment of information that can be told by the code. Contents such as "split the total by n" is an information that can be told by the code. Rather, you should write "Calculate the average" when n is a total number. In other words, comment should not be the operation itself but the meaning of the operation.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	C++

COM56　Write comment in English

Appropriate Example

```
//
// check the file format
//
```

Inappropriate Example

```
//
// fairu no kata wo tyekku suru
//
```

Comment

When you write comment, use only English, must not use two-byte codes such as Japanese, and explanation by Roman alphabet, This is because engineer who sees the source code is not only Japanese and the envirionment exists where you cannot watch Japanese comment at all. When two-byte codes can not be displayed correctly, the program action such as ddd debugger etc may cause problem.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	SystemC

COM57 When you do not use "SC_CTOR" macro at constructor declaration part of module class, do not forget to add constructor-initializer of base class sc_module

Appropriate Example

```
simple_bus_arbiter(sc_module_name name_, bool verbose = false)
: sc_module(name_)     // must
, m_verbse(verbose)
{
   ...
```

Inappropriate Example

```
simple_bus_arbiter(sc_module_name name_, bool verbose = false)
, m_verbse(verbose)
{
   ...
```

Comment

When use SC_CTOR macro at constructor declaration part of module class, do not have to take care.

However, when use normal description at constructor declaration instead of the SC_CTOR macro, You have to define module name to the argument of module constructor, and to pass the name to constructor-initializer of base class sc_module.

For example, you can not to use SC_CTOR macro when the module have the other arguments excluding module name.

Even if you forget to add constructor-initializer of base class sc_module, it will not be compile error, so it may become bug which is difficult to detect.

How to Check

sysc.c_cpp.14 (1Team:System)

Common	Required	SystemC

COM58 Need cast of sc_logic type when assign '0' / '1' /'Z' / 'X' to port/signal with sc_logic type

Appropriate Example

```
sc_out<sc_logic> out1;
...
out1 = (sc_logic)'0';
...
out1.write(sc_logic)'Z');
```

Inappropriate Example

```
sc_out<sc_logic> out1;
...
out1 = ‘0';    // compile error
...
out1.write('Z');
```

Comment

When assigning value to port/signal with sc_logic type, if you use '0'/'1'/'Z'/'X' as it is, it causes compile error because of treated as character. So you have to use sc_logic type casting when assigning.

To begin with, If you do not need to use 4 value, you should better use bool type instead of sc_logic type.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	SystemC

COM59 The order of data member initialization in constructor-initializer must be same to the declaration order

Appropriate Example

```
class Cmod : public sc_module
{
public:
   sc_in< sc_int<32> > in1;
   sc_in< sc_int<32> > in2;
   sc_out< sc_int<32> > out;
   ...
   Cmod(sc_module_name name_)
    : sc_module(name_)
    ,in1("in1")
    ,in2("in2")
    ,out("out")
   {
      out.initilaize(0);
      ...
   }
   ...
```

Inappropriate Example

```
class Cmod : public sc_module
{
public:
   sc_in< sc_int<32> > in1;
   sc_in< sc_int<32> > in2;
   sc_out< sc_int<32> > out;
   ...
   Cmod(sc_module_name name_)
    : sc_module(name_)
    ,out("out")
    ,in1("in1")
    ,in2("in2")
   {
      out.initilaize(0);
      ...
   }
   ...
```

Comment

The initialization order of data member in constructor-initializer must be same to the declaration order of data member in class.

Otherwise, warning occurs during compiling (the case of using -Wall option of g++).

Be careful to describe constructor-initializer when you add new data member.

How to Check

There is no tool to check it. Check it by designer.

Common	Required	SystemC

COM60　Module name should be same as instance name

Appropriate Example

```
Cmodule *mod1, *mod2;
...
mod1 = new Cmodule ("mod1");
mod2 = new Cmodule ("mod2");
```

Inappropriate Example

```
Cmodule *mod1, *mod2;
...
mod1 = new Cmodule ("mod1_inst");
mod2 = new Cmodule ("mod2_if");
```

Comment

Because the module name is used in error message, wave view and so on, naming same name for module name and instance name makes it efficient to debug.

How to Check

There is no tool to check it. Check it by designer.

Common	MUST	C++

COM61　Auto/const variable should be initialized in declaration

Appropriate Example

```
int count = 0;
const char my_name = "Taro";
```

Inappropriate Example

```
int count;
const char my_name;
```

Comment

Bugs are easily caused by uninitialized variable and it is difficult to analyze them. Therefore initializing at the same time when a variable is declared enables to prevent it.

How to Check

MRC-9.1 (1Team:System)

sfs.bp.ini.1 (1Team:System)

Common	MUST	SystemC

COM62 All port/signal should be named in constructor

Appropriate Example

```
class Cmod1 : public sc_module
{
public:
   sc_in<bool> clk;
   sc_in<bool> rst_n;
   sc_in<sc_uint<8> > in;
   sc_out<sc_uint<8> > out;
   sc_signal<sc_uint<8> > my_signal;
   Cmod1( sc_module_name name_ ) : sc_module(name_)
   , clk("clk")
   , rst_n("rst_n")
   , in("in")
   , out("out")
   , my_signal("my_signal")
   {
      ...
```

Inappropriate Example

```
class Cmod1 : public sc_module
{
public:
   sc_in<bool> clk;
   sc_in<bool> rst_n;
   sc_in<sc_uint<8> > in;
   sc_out<sc_uint<8> > out;
   sc_signal<sc_uint<8> > my_signal;
   Cmod1( sc_module_name name_ ) : sc_module(name_)
   // Not initialize
   {
      ....
```

Comment

Name all port/signal the same name as the instance name in constructor. If not, the default name will be such as "port_0" or "signal_0" which consists of "port" or "signal" and serial number, which makes debugging difficult.

However, you can not name array port/signal, pay attention.

How to Check

sysc.c_cpp.14 (1Team:System)

Common	Required	C++

COM63 You must describe separately condition ("if" and "else if") and processing for another line

Appropriate Example

```
if (condition == 0)
   data1 = 1;
else if (condtion2 == 1)
   data1 = 2;
else data1 = 0;
```

Inappropriate Example

```
if (condition == 0) data1 = 1;
else if (condtion2 == 1) data1 = 2;
else data1 = 0;
```

Comment

To get the result of line coverage with gcov correctly, it is necessary to describe separately condition ("if" and "else if") and processing for another line. Otherwise, whether only the condition judgment was done or processing was done together cannot be distinguished. For synthesis, keep this rule because you must get line coverage.

For simulation, keep this rule if you need to get line coverage.

However, you may write condition and processing in the same line for the initialization check of the pointer passed by the function argument if you are sure that the pointer never become NULL.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++

MOD1 Take care whether the variable should be "public" or "private"

Appropriate Example

```
public:
   int reg_access(int addr, int data, int type);      // Member method which have access from outside.
private:
   int calc_addr(int addr);     // member method which is used only inside.
```

Inappropriate Example

```
public:
   int reg_access(int addr, int data, int type);      // Member method which have access from outside.
   int calc_addr(int addr);     // Member method which is used only inside.
```

Comment

The public method is one of class specification. A lot of methods with internal implementation methods and data members confuse class user. Besides, when updating model internal implementation not external specification, that modification affects class user too. Thus class has public method as few as possible.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++

MOD2 The argument of function should be "class"/"struct" pointer not copy of "class"/"struct"

Appropriate Example

```
typedef struct stag {
   int mem1;
   int mem2;
   ...
} STAG;
int func (const STAG *p) {
   return p->mem1 + p->mem2;
}
```

Inappropriate Example

```
typedef struct stag {
   int mem1;
   int mem2;
   ...
} STAG;
int func (const STAG p) {
   return p.mem1 + p.mem2;
}
```

Comment

When passing a copy of "class"/"struct" as argument of function, processing of coping the all data to domain for argument occurs when function is called. So if the "class"/"struct" is big in size, it causes to deteriorate speed performance.

When passing "class"/"struct" used only for reference, make it not simply be pointer but also add const modification.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++

MOD4 Do not use brace when you allocate pointer array by "new"

Appropriate Example

```
pst_list = new sc_uint<16> *[10];
```

Inappropriate Example

```
pst_list = new (sc_uint<16>*) [10];
```

Comment

Do not use brace when you allocate pointer array by "new". It may cause compile error depending on the compiler.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++

MOD5　Check the function's return value

Appropriate Example

```
FILE *fp = NULL;
fp = fopen("test.scr", "r");
if (fp == NULL) {
   fprintf(stderr, "Cannot open ¥"%s¥" file to read.¥n", "test.scr");
   exit(1);
}
...
```

Inappropriate Example

```
FILE *fp = NULL;
fp = fopen("test.scr", "r");
...
```

Comment

Regardless of input file or output file, be sure to check if the file open is succeeded or not. Generally, it is difficult to associate bug to be occur onward and file open error and taking measure is difficult.

Not only incorporated function in this file, if there is interface to judge the proceeding content, be sure to check it before proceeding to next processing.

How to Check

MRC-16.8 (1Team:System)

Simulation	MUST	C++

MOD6　Check "memory leak"

Appropriate Example

```
char *buf = NULL;
buf = new char [10];
...
delete [] buf;
```

Inappropriate Example

```
char *buf = NULL;
buf = new char [10];
... // Not release
```

Comment

Memory leak may cause unintended behavior when the variable is used without release. Even if there is no problem in function, speed to execute program may drop.

An example of a case to incur memory leak is a case when memory have to be allocated in a function and have to be released in a side the function is used. That is, it often happens when situation is different for allocation and release. Users have to understand the way of use.

How to Check

Purify/Valgrind

Simulation	MUST	C++

MOD7 Should study the difference between "new" and "new[]" ("delete" and "delete[]" too)

Appropriate Example

```
static void TestMain(void)
{
  TestClass* theT;

  theT   = new TestClass[8];
  ...
  delete[] theT;
}
```

Inappropriate Example

```
static void TestMain(void)
{
  TestClass* theT;

  theT   = new TestClass[8];
  ...
  delete theT;
}
```

Comment

You should understand well that new and new[](delete and delete[]) is different. The bug may cause memory destruction and forget of resource release , which cause another bugs. The program may even do not operate properly.

In the inappropriate example, as you can easily see when you check putting display routine in TestClass constructor and destructor, the destructor boot only once while the constructor boot eight times. That means, seven objects remain without being release, causing memory leak. To put it correctly, write "delete[] theT;" as you see in the appropriate example.

How to Check

elm.rule.51a (1Team:System)

elm.rule.51b (1Team:System)

Simulation	MUST	C++

MOD8 Do not delete allocated variable twice or more

Appropriate Example

```
delete mPtr;
...                    // Do not delete mPtr after this.
```

Inappropriate Example

```
delete mPtr;
...
delete mPtr;      // delete twice or more
```

Comment

As you see in the following example, you can forcibly assign NULL for deleted variables to prevent double delete.

```
delete mPtr;
          mPtr = NULL;
```

This makes it no problem when you delete mPtr again. This is because in C++ specification, "Nothing happens when you apply delete to 0" ("programming language C++ 3rd edition"6.2.6"Allocating free memory area"), so "delete NULL;" should not be a problem.

How to Check

Purify/Valgrind

Simulation	MUST	C++

MOD9 Do not delete unallocated variable

Appropriate Example

```
class TestClass {
   char* mText;
public:
   TestClass() { mText = NULL }
   TestClass(const char* inT) { mText = strdup(inT); }
   ~TestClass(){ delete mText; }
   void Set(const char* inT) {
      delete mText;
      mText = strdup(inT);
   }
   const char* Get() const { return mText; }
   ...
};
static void TestMain(void)
{
   TestClass theT2;
   theT2.Set("TEST");
   cout << theT2.Get() << endl;
}
```

Inappropriate Example

```
class TestClass {
   char* mText;
public:
   TestClass() {}
   TestClass(const char* inT) { mText = strdup(inT); }
   ~TestClass() { delete mText; }
   void Set(const char* inT) {
      delete mText;
      mText = strdup(inT);
   }
```

```
    const char* Get() const { return mText; }
    ...
};
static void TestMain(void)
{
    TestClass theT2;
    theT2.Set("TEST");
    cout << theT2.Get() << endl;
}
```

Comment

It will be a program that evaluate with unallocated pointer, and it may be low in portability and become unsolvable bug. You should take some measures not to unallocated value happens, by initializing in NULL in a constructor in advance etc.

In the inappropriate example, when Set member function is called it is "delete mText". mText remains unallocated value(because "mText = NULL;" is not written in a default constructor.), so it is a problem.

How to Check

Purify/Valgrind

Simulation	MUST	C++

MOD10　Take care of "endian"

Appropriate Example

```
unsigned char mem[4];
   unsigned int reg;
#ifdef BIG_ENDIAN
   mem[0] = 0x00;
   mem[1] = 0x11;
   mem[2] = 0x22;
   mem[3] = 0x33;
#elif LITTLE_ENDIAN
   mem[0] = 0x33;
   mem[1] = 0x22;
   mem[2] = 0x11;
   mem[3] = 0x00;
#endif
   reg = *((unsigned int *) &mem[0]);
   printf("reg = 0x%08X¥n", reg);
// In both Big Endian and Little Endian environment, reg = 0x00112233.
```

Inappropriate Example

```
unsigned char mem[4];
   unsigned int reg;
   mem[0] = 0x00;
   mem[1] = 0x11;
   mem[2] = 0x22;
   mem[3] = 0x33;
   reg = *((unsigned int *) &mem[0]);
   printf("reg = 0x%08X¥n", reg);
// In Big Endian environment, reg = 0x00112233,in Little Endian environment, reg = 0x33221100.
```

Comment

It is necessary to exchange data through memory according to the execution environment endian（Kind of method to assign multi byte data on memory). Programmer have to write code considering endian when access unit at data storing and data acquisition are different(when acquiring data assigned by byte unit as longword unit etc.), though it is not necessary when the access unit is same. Above inappropriate example is a case where variable reg differ under Big Endian and Little Endian. On the other hand, in the appropriate example, desired value can be acquired by using #ifdef, not depending on the endian of the execution environment.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++

MOD11 Do it to fatal error when memory allocation is fail

Appropriate Example

```
#define BUF_SIZE 0x1000000
char *pic_buf = NULL;
pic_buf = new char [BUF_SIZE];
if ( pic_buf == NULL ) {
    fprintf(stferr, "Cannot alloc memory (%d size)¥n", BUF_SIZE);
    exit(0);
}
result = fread(pic_buf, 1, size, fp);
```

Inappropriate Example

```
#define BUF_SIZE 0x1000000
pic_buf = new char [BUF_SIZE];
result = fread(pic_buf, 1, size, fp);
```

Comment

Memory allocation failure (new, malloc) merely happens with today's PC/EWS spec. However, when you made a mistake to make the size too big, you should stop the processing. Although code to allocate memory after checking upper limit of the size using assertion can be possible, checking error returned by new/malloc is necessary because useable memory size changes depending on the situation.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++

MOD12 File name, line number, and other variable should not have local limitation

Appropriate Example

```
char *str_copy(const char* str) {
   int str_length = strlen(str) + 1;
   char *new_str = new char [str_length];
   strcpy(new_str, str);
   return new_str;
}
```

Inappropriate Example

```
char *str_copy(const char* str) {
   char *new_str = new char[16];
   strcpy(new_str, str);
   return new_str;
}
```

Comment

To write a good program, do not set limitation to data structure (file name, string) to be entered for its length and number, as much as possible. If it is difficult, when unexpected size is entered, do not stop the processing on your own but it is necessary to take measure such as to output message.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++

MOD13 "{" brace should be at next line at function definition. Other brace should be same line

Appropriate Example

```
void myFunc (int length)
{
  for (int i=0 ; i<100 ; i++ ) {
  ...
}
```

Inappropriate Example

```
void myFunc (int length) {
  for (int i=0 ; i<100 ; i++ )
  {
    ...
}
```

Comment

Rule position to write "{", which indicates block. When you define function, write"{" to next line. Inappropriate example shows example of writing "{" same line as the function name. Write all other "{" same line as control description. Description like the appropriate example is called K&R style, and the inappropriate example is called GNU style.

This rule is effective in saving unnecessary work caused by difference of style, when you divert code within the group.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++

MOD14　Define the variable in least scope

Appropriate Example

```
int size = 0;
for (int loop=0 ; loop<10 ; loop++)
   size += loop;
size += 10;
```

Inappropriate Example

```
int size = 0;
int loop = 0;
for (loop=0 ; loop<10 ; loop++)
   size += loop;
size += 10;
```

Comment

In the C language specification, it is necessary to write variable declaration altogether before starting to write control statement, in C++ language specification the scope of description is widen. Therefore define the variable in the least scope. For example in the appropriate example, the variable" loop" is a variable only used in the for description. Therefore, it should be defined in the for sentence. By this, scope of variable used can be controlled to be minimum. If you declare variables in vast area which is not in use, it may become difficult to tell where they are referred and updated.

How to Check

There is no tool to check it. Check it by designer.

Simulation	MUST	C++

MOD15　The line number of each function is within 200 lines

Appropriate Example

```
// The line number of the function is within 200 lines
```

Inappropriate Example

```
// The line number of the function is more than 200 lines
```

Comment

It often happens that function to implement is so complicated that its code becomes very big. When the number of the lines is too many, interpretation and improvement of the program becomes difficult, also becomes difficult to take measure for bug when it happens. To prevent this, limit the line number of each function within 200 lines. When exceeding this number, define the processing with another function and call it.200 lines is 2.5 times of 80 lines which is the number of lines that can be seen at once. That's why 200 is regarded as a appropriate number of lines to see overall picture.

When the phase of trial and error and changing code in bug investigation are exception. Measures should be taken before release.

How to Check

sfs.bp.fnc.4 (1Team:System)

Simulation	MUST	SystemC

MOD16 Never define an inherited non virtual method

Appropriate Example

```
// parent class
void set_data1(char);
virtual void set_data2(char);
virtual void set_data3(char) = 0;
...
// child class
void set_data2(char);
void set_data3(char);
```

Inappropriate Example

```
// parent class
void set_data1(char);
virtual void set_data2(char);
virtual void set_data3(char) = 0;
...
// child class
void set_data1(char); // over write
void set_data2(char);
void set_data3(char);
```

Comment

The method defined in parent class can be overwrite in child class. Both virtual method and non virtual method can be. But it is out of rule for parent class to over write the non virtual method, and it causes maintenance falls off. If you need to change the method, you should change the parent's method specification.

How to Check

emb.cpp.bp.5 (1Team:System)

Simulation	MUST	SystemC

MOD17 When you read port/signal (a lot of times at same cycle), at first assign the value to variable of C++ native type

Appropriate Example

```
sc_signal<sc_uint<32> > pa;
unsigned int buf;
buf = pa.read();
pd = buf;
pc = buf + (buf<<16) + var;
...
```

Inappropriate Example

```
sc_signal<sc_uint<32> > pa;
pb = pa.read();
pc = pa.read() + (pa.read()<<16) + var;
```

Comment

When referring the value of port/signal, you should use read, however, read method call is a little heavy load process. So when value is frequently referred, store the value in variable of C/C++ native type and refer it later.

However, after stores the value in the variable, wait call is stepped over, it is necessary to store the value again after wait call. Because there is a possibility that the value of the port/signal has changed by exceeding the cycle boundary.

How to Check

sysc.dsgn.1 (1Team:System)

Simulation	MUST	SystemC

MOD18 Use "posedge_event()" not "pos()" for the wait() method's argument

Appropriate Example

```
wait( clk.posedge_event());
```

Inappropriate Example

```
wait(clk.pos());
```

Comment

When you specify rising event of bool type input port for argument of wait () function, call posedge_event() not pos(). Even if you use pos(), it will not cause compiler error but the simulation stops there. Same can be said to falling edge negedge_event() and neg(). Same for value_changed_event() and value_changed().

How to Check

sysc.lang.24 (1Team:System)

Simulation	Required	SystemC

MOD19 All data member should be initialized in constructor

Appropriate Example

```
class Cmod1 : public sc_module
{
public:
   sc_out<sc_uint<8> > out_port;
   sc_signal<sc_uint<8> > my_signal;
   sc_uint<8> val;
   Cmod1( sc_module_name name_ ) : sc_module(name_)
   , out_port("out_port")
   , my_signal("my_signal")
   , val(0)
   {
      out_port.initialize(0);
      my_signal = 11;
      ...
```

Inappropriate Example

```
class Cmod1 : public sc_module
{
public:
   sc_out<sc_uint<8> > out_port;
   sc_signal<sc_uint<8> > my_signal;
   sc_uint<8> val;
   Cmod1( sc_module_name name_ ) : sc_module(name_)
   // Not initialize
   {
      // Not initialize
      ....
```

Comment

Initialize all data members by constructor. Member not initialized may cause bug and sometimes makes it difficult to debug.

Note that initialization of output port by constructor is only enable by using "initialize" method.

How to Check

sysc.c_cpp.14 (1Team:System)

Synthesis	MUST	SystemC

SYN2　A module must be directly inherited from sc_module

Appropriate Example

```
SC_MODULE(mod1)
{
   SC_CTOR(mod1) {}
   ...
};
class mod2 : public sc_module
{
   ...
};
```

Inappropriate Example

```
SC_MODULE(mod1)
{
   SC_CTOR(mod1){}
   ...
};
class mod3 : public mod1
{
   ...
};
```

Comment

The synthesis target module should be directly inherited from sc_module.

The high level synthesis tool(CtoS) supports to make a module which is inherited from the existing other module. However, it is not recommended to use this feature because of not tested at Renesas.

How to Check

sysc.synth1.1 (1Team:System)

Synthesis	MUST	SystemC

SYN3　Do not use SC_THREAD

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
    SC_METHOD(my_method);
    sensitive << in1 << in2;
  }

  void my_thread() {
    ...
  }

  void my_method() {
    ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  SC_CTOR(mod) {
    SC_THREAD(my_thread);
    sensitive << event1;
  }

  void my_thread() {
    ...
  }
```

```
    };
```

Comment

By the restriction of the high level synthesis tool(CtoS), It is only SC_CTHREAD and SC_METHOD that is available for process of module.

Do not use SC_THREAD basically, however you can use it in a testbench module which is not a synthesis target or in a behavior level module.

How to Check

sysc.synth1.8 (1Team:System)

Synthesis	MUST	SystemC

SYN4 Do not use inout port (sc_inout)

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  SC_CTOR(mod) {
    ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_inout<int> io1;
  SC_CTOR(mod) {
    ...
  }
};
```

Comment

By the restriction of the high level synthesis tool(CtoS), you can use only input port (sc_in) and output port (sc_out) for the port of a module.

Do not use inout port (sc_inout).

When you want to use sc_inout, need to divide into two ports of sc_in and sc_out.

How to Check

sysc.synth1.9 (1Team:System)

Synthesis	MUST	SystemC

SYN5 When registering SC_CTHREAD process in module constructor, set only one clock port with edge and one or more reset ports

Appropriate Example

```
SC_MODULE(mod) {
  sc_in<bool> clk;
  sc_in<bool> rst_n1;
  sc_in<bool> rst_n2;
  sc_in<bool> arst;
  ...
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread1, clk.pos());
    reset_signal_is(rst_n1, false);
    reset_signal_is(rst_n2, false);
    SC_CTHREAD(my_thread2, clk.pos());
    async_reset_signal_is(arst, false);
    reset_signal_is(rst_n1, false);
    ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
  sc_in<bool> clk;
  sc_in<bool> arst1;
  sc_in<bool> arst2;
  ...
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread1, clk.pos());
    SC_CTHREAD(my_thread2, clk.pos());
    async_reset_signal_is(arst1, false);
    async_reset_signal_is(arst2, false);
    ...
  }
```

```
    ...
};
```

Comment

When registering SC_CTHREAD process in module constructor, set only one clock port with edge(pos() or neg()) and one or more reset ports.

You can use synchronous reset(reset_signal_is) and asynchronous reset(async_reset_signal_is) as setting reset port, however, only one or none can be set about asynchronous reset.

How to Check

sysc.synth1.29 (1Team:System)

execution log of CtoS

Synthesis	MUST	SystemC

SYN7 You must initialize data members, output ports and signals in reset block, not in constructor

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  sc_signal<int> sig1;
  int data1;
  SC_CTOR(mod)
   , in1("in1")
   , out1("out1")
   , sig1("sig1")
  {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }

  void my_thread() {
    out1.write(0);
    sig1.write(0);
    data1 = 0;
    wait();
    while (1) {
      ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
```

```
    sc_in<int> in1;
    sc_out<int> out1;
    sc_signal<int> sig1;
    int data1;
    SC_CTOR(mod)
      , in1("in1")
      , out1("out1")
      , sig1("sig1")
    {
        out1.write(0);
        sig1.write(0);
        data1 = 0;
        SC_CTHREAD(my_thread, clk.pos());
        reset_signal_is(rstn, false);
    }
    void my_thread() {
      wait();
      while (1) {
        ...
};
```

Comment

In high level synthesis by CtoS, the initialization by the constructor is ignored. So it is necessary to describe the initialization of data members, output ports and signals in the reset block of the SC_CTHREAD process.

How to Check

sysc.synth1.15 (1Team:System)

Synthesis	MUST	SystemC

SYN8 You must not describe the function of module in constructor

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
};
```

Inappropriate Example

```
SC_MODULE(mod)
{
  ...
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
    int data = in1.read();
    for (int i =0; i<10; i++) {
      ...
  }
};
```

Comment

In high level synthesis by CtoS, you must not describe the function of module in constructor.

CtoS uses following 2 descriptions only for synthesis, the other descriptions are ignored.

・registering processes of that module

・the port bindings of modules instantiated within that module

How to Check

sysc.synth1.40 (1Team:System)

Synthesis	MUST	SystemC

SYN9 In SC_CTHREAD process, describe reset sequence at first, and call wait() only once just before infinite loop

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  sc_signal<int> sig1;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread() {
    out1.write(0);
    sig1.write(0);
    wait();
    while (1) {
      ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  sc_signal<int> sig1;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
```

```
    void my_thread() {
      while (1) {
        ...
    }
  };
```

Comment

In SC_CTHREAD process, describe reset sequence at first, and call wait() only once just before infinite loop.

This is need for high level synthesis tool(CtoS) to interpret correctly where is reset block.

How to Check

sysc.synth1.14 (1Team:System)

sysc.synth1.37 (1Team:System)

Synthesis	MUST	SystemC

SYN10 In reset block of SC_CTHREAD process, initialize all output ports/signals, and you must not describe any other function

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  sc_signal<int> sig1;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread() {
    out1.write(0);
    sig1.write(0);
    wait();
    while (1) {
      ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  sc_signal<int> sig1;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
```

```
    void my_thread() {
      if (in1.read() == 0)
        out1.write(0);
      else
        out1.write(1);
      while (1) {
          ...
    }
  };
```

Comment

In reset block of SC_CTHREAD process, describe only initializing all output ports/signals and data member and so on used in that process, and you must not describe any other function (sysc.synth1.16a of 1Team:System can check).

And do not use following syntaxes.

1. description including the condition branch such as "if", "switch", and condition operator"? :"
2. loop expression that cannot be unrolled
3. assignment value except fixed number to output ports/signals

When use these, error may be detected during CtoS execution.

How to Check

sysc.synth1.16a (1Team:System)

execution log of CtoS

Synthesis	MUST	SystemC

SYN11　Take care of access to the same signal from several parallel processes

Appropriate Example

```
SC_MODULE(mod) {
{
  sc_in<int> in1;
  sc_in<int> in2;
  sc_out<int> out1;
  sc_signal<int> sig1;
  int tmp;
  ...

  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
    SC_METHOD(my_method);
    sensitive << in2;
  }
  void my_thread() {
    ...
    while (1) {
      int data = in1.read() + sig1.read();
      out1.write(data);
      wait();
    }
  }
  void my_method() {
    tmp = in2.read() << 2;
    sig1.write(tmp);
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
```

```
{
  sc_in<int> in1;
  sc_in<int> in2;
  sc_out<int> out1;
  sc_signal<int> sig1;
  int tmp, tmp2;
  ...

  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
    SC_METHOD(my_method);
    sensitive << in2;
  }
  void my_thread() {
    ...
    while (1) {
      int data = in1.read() + sig1.read();
      sig1.write(tmp2); // NG
      out1.write(data);
      wait();
    }
  }
  void my_method() {
    tmp = in2.read() << 2;
    sig1.write(tmp); // NG
  }
};
```

Comment

When you access to the same signal (the sc_signal variable registered as data member of the module) from several parallel processes, do not write value to the signal in other processes that written in certain process.

There is risk where the racing trouble occurs.

How to Check

sysc.synth.generic.2a (1Team:System)

sysc.synth.generic.2b (1Team:System)

Synthesis	MUST	SystemC

SYN12 When using SC_METHOD process, need to set all ports and signals becoming input of combined circuit to static sensitivity list

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  SC_CTOR(mod) {
    SC_METHOD(my_method);
    sensitive << in1;
  }

  void my_method() {
    if (in1.read() == 0)
      out1.write(0);
    else
      out1.write(1);
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  SC_CTOR(mod) {
    SC_METHOD(my_method);
    sensitive << in1;
  }

  void my_method() {
    if (in1.read() == 0)
      out1.write(0);
    else if (in1.read() == 1)
      out1.write(1);
  }
};
```

Comment

When using SC_METHOD process, need to set all ports and signals becoming input of combined circuit to static sensitivity list.

And in SC_METHOD process, there exists branch description and write operation to port/signal in the branch, it is necessary to describe write operation in all branches without exception. Unless do this, the synthesis processing may generate an unexpected redundant latch.

How to Check

sysc.rtl.2 (1Team:System)

Synthesis	Required	SystemC

SYN13 Do not use sc_port<>

Appropriate Example

```
SC_MODULE(mod) {
{
  sc_in<bool> clk;
  sc_in<bool> rstn;
  sc_in<int> in1;
  sc_out<int> out1;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread();
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  sc_in<bool> clk;
  sc_in<bool> rstn;
  sc_in<int> in1;
  sc_out<int> out1;
  sc_port<tlm_blocking_put_if<int> > port;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread();
};
```

Comment

The high level synthesis tool(CtoS) supports use of sc_port with specific interface(TLM1.0 interface). However, it is not recommend to use this feature because of not tested at Renesas.

How to Check

sysc.synth1.28 (1Team:System)

Synthesis	MUST	SystemC

SYN14 Do not use a function with recursive call in SC_CTHREAD/SC_METHOD process

Appropriate Example

```
SC_MODULE(mod) {
{
   sc_in<bool> clk;
   sc_in<bool> rstn;
   sc_in<int> in1;
   sc_out<int> out1;
   SC_CTOR(mod) {
      SC_CTHREAD(my_thread, clk.pos());
      reset_signal_is(rstn, false);
   }
   void my_thread() {
      out1.write(0);
      wait();
      while (1) {
         int data = in1.read();
         data = factorial(data);
         out1.write(data);
         wait();
      }
   }
   int factorial(int n) {
      int val = 1;
      if (n == 0 || n == 1)
         return 1;
      else {
         for ( ; n > 0; n--)
            val *= n;
         return val;
      }
   }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  sc_in<bool> clk;
  sc_in<bool> rstn;
  sc_in<int> in1;
  sc_out<int> out1;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread() {
    out1.write(0);
    wait();
    while (1) {
      int data = in1.read();
      data = factorial(data);
      out1.write(data);
      wait();
    }
  }
  int factorial(int n) {
    if (n == 0 || n == 1)
      return 1;
    else
      return (n * factorial(n - 1));
  }
};
```

Comment

By the restriction of the high level synthesis tool(CtoS), do not use a function with recursive call in SC_CTHREAD/SC_METHOD process.

How to Check

sysc.synth1.18 (1Team:System)

Synthesis	MUST	SystemC

SYN15 Do not use a dynamic memory allocation in module by new or malloc/calloc

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  sub_mod *sub;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
    sub = new sub_mod("sub");
    ...
  }
  void my_thread();
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  sc_uint<30> *data;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread() {
    ...
    while (1) {
      ...
      data = new sc_uint<30>[10];
```

```
            ...
            delete [] data;
            wait();
        }
    };
```

Comment

By the restriction of the high level synthesis tool(CtoS), do not use a dynamic memory allocation in module by new or malloc/calloc.

However it is only possible to generate internal instance in the hierarchy module by new.

How to Check

sysc.synth1.20 (1Team:System)

Synthesis	MUST	SystemC

SYN16 Do not use decimal point type

Appropriate Example

```
SC_MODULE(mod) {
{
  sc_in<bool> clk;
  sc_in<bool> rstn;
  sc_in<int> in1;
  sc_out<int> out1;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread() {
    out1.write(0);
    wait();
    while (1) {
      int data = in1.read();
      data *= data;
      out1.write(data);
      wait();
    }
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_signal<sc_ufixed<16, 8> > sig1;
  sc_fixed<16, 8> val;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
```

```
        ...
    }
    void my_thread()   {
        ...
        double tmp1;
        float tmp2;
        ...
};
```

Comment

By the restriction of the high level synthesis tool(CtoS), do not use floating point type(float, double) (sysc.synth1.23 of 1Team:System can check).

Though CtoS supports fixed-point type(sc_fixed, sc_ufixed), it is not recommended to use this feature because of not tested at Renesas (sysc.synth1.12 and sysc.synth1.13 of 1Team:System can check).

How to Check

sysc.synth1.12 (1Team:System)

sysc.synth1.13 (1Team:System)

sysc.synth1.23 (1Team:System)

Synthesis	MUST	SystemC

SYN17 Do not use sc_logic type and sc_lv type

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<bool> in1;
  sc_out< sc_uint<2> > out1;
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread() {
      ...
      bool data = in1.read();
      ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<sc_logic> in1;
  sc_signal< sc_lv<2> > sig1;

  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }

  void my_thread()  {
      ...
      sc_logic data = in1.read();
```

```
        ...
    }
};
```

Comment

sc_logic type and sc_lv type can use 4 value(0, 1, X, Z), but X value and Z value are not supported by high level synthesis tool(CtoS).

So if you want to treat 2 value(0 and 1), use bool type as scalar or sc_uint type as vector, do not use sc_logic type and sc_lv type.

How to Check

There is no tool to check it. Check it by designer.

Synthesis	MUST	SystemC

SYN18 Take care of using sizeof operator and range method when you operate processing with SystemC data type

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread() {
    ...
    unsigned int data = 10;
    int a = sizeof(data);
    int b = (int)data.range(2, 0);
    ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  SC_CTOR(mod) {
    SC_CTHREAD(my_thread, clk.pos());
    reset_signal_is(rstn, false);
  }
  void my_thread() {
    ...
    sc_uint<16> data = 10;
    int a = sizeof(data);
    int b = (int)data.range(n, 0);
    bool c = (bool)data(m);
```

```
        ...
      }
    };
```

Comment

Do not use sizeof operator to the variable of SystemC data type(sc_int, sc_uint, etc). It may cause bug because of not returning precise size　considered type and bit width.

Pass only constant value into the arguments of range method of SystemC data type. If you pass variable, error is detected during CtoS execution. It's the same when using operator () instead of range method.

How to Check

execution log of CtoS

Synthesis	MUST	SystemC

SYN19 When you make a user defined type to use with port and signal, keep some decided rules

Appropriate Example

```
class UserDef {
public:
   unsigned char flg;
   int data[2];
   UserDef() {
      flg = 0;
      for (int i = 0; i < 2; i++)
         data[i] = 0;
   }
   UserDef(const UserDef& c) {
      flg = c.flg;
      for (int i = 0; i < 2; i++)
         data[i] = c.data[i];
   }
   inline bool operator == (const UserDef& rhs) const {
      bool _flg = true;
      bool _data = true;
      _flg = rhs.flg == flg;
      for (int i = 0; i < 2; i++)
         _data = _data && (rhs.data[i] == data[i]);
      return _flg && _data;
   }
   inline UserDef& operator = (const UserDef& rhs) {
      flg = rhs.flg;
      for (int i = 0; i < 2; i++)
         data[i] = rhs.data[i];
      return *this;
   }
   inline friend void sc_trace(sc_trace_file *tf, const UserDef& v, const std::string& NAME) {
      char tmp[256];
```

```
        sc_trace(tf, v.flg, NAME + ".flg");
        for (int i = 0; i < 2; i++) {
            sprintf(tmp, ".data[%d]", i);
            sc_trace(tf, v.data[i], NAME + tmp);
        }
    }
    inline friend std::ostream& operator << (std::ostream& os, const UserDef& v) {
        os << "." << v.flg;
        for (int i = 0; i < 2; i++)
            os << "." << v.data[i];
        return os;
    }
};
SC_MODULE(mod) {
{
    ...
    sc_in<UserDef> in1;
    sc_out<UserDef> out1;
    sc_sig<UserDef> sig1;
    ...
};
```

Inappropriate Example

```
class UserDef {
public:
    unsigned char flg;
    int data[2];
    UserDef() {
        flg = 0;
        for (int i = 0; i < 2; i++)
            data[i] = 0;
    }
};
SC_MODULE(mod) {
{
    ...
    sc_in<UserDef> in1;
    sc_out<UserDef> out1;
    sc_sig<UserDef> sig1;
    ...
};
```

Comment

By the restriction of the SystemC language specification and the high level synthesis tool(CtoS), when you make a user defined type to use with port and signal, keep following rules.

1. Define default constructor
2. Do overload == operator, = operator, << operator and sc_trace()

If you have other operators to use in algorithm description except the above, you need to do overload them.

How to Check

sysc.lang.68 (1Team:System)

execution log of CtoS

Synthesis	MUST	SystemC

SYN20 When perform pipeline synthesis by CtoS, keep some decided rules

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  void test::test_proc() {
    out1.write(0);
    wait();
    while (1) {
      int tmp;
      for (int i = 0; i < 32; i++) {
        if (cond.read() == 0) {
          tmp = in1.read();
          wait();
          tmp++;
          out1.write(tmp);
        } else {
          out1.write(tmp);
          wait();
        }
      }
      if (cond.read() == 0) {
        tmp = in1.read();
        tmp++;
        wait();
        out1.write(tmp);
      } else {
        tmp = in1.read();
        wait();
        out1.write(tmp);
      }
      wait();
    }
    }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
   ...
   void test::test_proc() {
      out1.write(0);
      wait();
      while (1) {
         int tmp;
         for (int i = 0; i < 32; i++) {
            if (cond.read() == 0) {
               tmp = in1.read();
               wait();
               tmp++;
               out1.write(tmp);
            } else {
               out1.write(tmp);
               wait();
               break;
            }
         }
         if (cond.read() == 0) {
            tmp = in1.read();
            tmp++;
            wait();
            out1.write(tmp);
         } else {
            wait();
            wait();
            tmp = in1.read();
            out1.write(tmp);
         }
         wait();
      }
   }
```

```
  };
```

Comment

By the restriction of high level synthesis tool(CtoS), when apply pipeline synthesis, keep following rules.

1. In the target pipeline loop, the use of "break" is permitted only when the break condition has not wait call, and the pipeline loop does not have escape formula by loop counter.

2. When there is access to a same port by each branch in the branch description of the target pipeline loop, you need to keep the same cycle when the port access occurs in each branch.

How to Check

execution log of CtoS (No.1)

Check No.2 by designer.

Synthesis	Required	SystemC

SYN21　On high level design flow, use commonly some macros for preprocessor

Appropriate Example

```
#include "systemc.h"
#ifdef DEBUG
#include <cstdio>
#include <cstdlib>
#endif
#ifdef __CTOS__
#include "ctosLib.h"
#endif
#ifdef ASSERT_ON_HDL
#include "dut_sva.h"
#endif
```

Inappropriate Example

```
#include "systemc.h"
#ifdef _VCS
#include <cstdio>
#include <cstdlib>
#endif
#ifdef _CtoS
#include "ctosLib.h"
#endif
#ifdef _ASSERT
#include "dut_sva.h"
#endif
```

Comment

On high level design flow, use commonly following macros for preprocessor.

1. __CTOS__：　Use for code of high level synthesis tool(CtoS). This macro is defined as tool specification.

2. MODE_SYSTEMC or MODE_SYSTEMC_XXX：　Use for code of SystemC environment　（The string "XXX" expresses various kind).

3. ASSERT_ON_HDL：　Use for code of assertion described with HDL.

4. DEBUG or DEBUG_XXX： Use for code of debug（The string "XXX" expresses a kind of debug).

How to Check

There is no tool to check it. Check it by designer.

Synthesis	Required	SystemC

SYN22 You should store a value of port and signal in a variable once and pass it to operation processing. And the operation processing should be implemented as a member function as much as possible

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  void my_thread() {
    int in1_buf, in2_buf;
    int out1_buf, out2_buf;
    out1.write(0);
    out2.write(0);
    wait();
    while (1) {
      in1_buf = (int)in1.read();
      in2_buf = (int)in2.read();
      out1_buf = func(in1_buf, in2_buf);
      out1.write((sc_int<10>)out1_buf);
      wait();
      in1_buf = (int)in1.read();
      in2_buf = (int)in2.read();
      out2_buf = func2(in1_buf, in2_buf);
      out2.write((sc_int<10>)out2_buf);
      wait();
    }
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  void my_thread() {
        int in1_buf, in2_buf;
        int out1_buf, out2_buf;
        out1.write(0);
        out2.write(0);
        wait();
        while (1) {
            in1_buf = (int)in1.read();
            in2_buf = (int)in2.read();
            out1.write((sc_int<10>)(in1.read()+in2.read()));
            wait();
            out2_buf = func2(in1_buf, in2_buf);
            out2.write((sc_int<10>)out2_buf);
            wait();
        }
    }
};
```

Comment

You should store a value of port/signal in a variable once and pass it to operation processing. And the operation processing should be implemented as a member function as much as possible.

This enables to distinguish communication part and calculation part definitely, and to improve the reusability of the calculation part.

Though you can describe calculation part in SC_CTHREAD process directly, should not do this by the above reason.

However, after stores the value in the variable, wait call is stepped over, it is necessary to store the value again after wait call. There are 2 reasons for this. One reason is that there is a possibility that the value of the port/signal has changed by exceeding the cycle boundary. And another reason is that the variable is considered FlipFlop with X value by SLEC.

How to Check

There is no tool to check it. Check it by designer.

Synthesis	Required	SystemC

SYN23 When you use SC_METHOD process, should not set clock edge or reset to static sensitivity list

Appropriate Example

```
SC_MODULE(mod) {
{
  sc_in<bool> clk;
  sc_in<bool> rstn;
  sc_in<int> in1;
  sc_out<int> out1;
  SC_CTOR(mod) {
    SC_METHOD(my_method);
    sensitive << in1;
  }
  void my_method() {
    if (in1.read() == 0)
      out1.write(0);
    else
      out1.write(1);
  }
  ...
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  sc_in<bool> clk;
  sc_in<bool> rstn;
  sc_in<int> in1;
  sc_out<int> out1;
  SC_CTOR(mod) {
    SC_METHOD(my_method);
    sensitive << clk.pos() << rstn;
  }
```

```
    void my_method() {
      if (in1.read() == 0)
        out1.write(0);
      else
        out1.write(1);
    }
    ...
  };
```

Comment

When you use SC_METHOD process with static sensitivity list which has clock edge and reset, the result of synthesis becomes RTL itself.

You should describe by SC_CTHREAD process in such a case.

How to Check

There is no tool to check it. Check it by designer.

Synthesis	Required	SystemC

SYN24 When you can store an intermediate result, you should use variable not signal (sc_signal)

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  int var;
  ...
  void my_thread() {
    ...
    var = funcA();
    wait();
    funcB(var);
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in1;
  sc_out<int> out1;
  sc_signal<int> sig1;
  ...
  void my_thread() {
    ...
    sig1.write(funcA());
    wait();
    funcB(sig1.read());
  }
};
```

Comment

When you can store an intermediate result, you should use variable not signal (sc_signal).

There are following 3 reasons.

1. The high level synthesis tool(CtoS) tends to assign a reserved register for every signal one by one, on the other hand, variable can share one register if the life time does not overlap. The latter can reduce the area of the circuit.

2. When there is no violation of data dependence to the input/output action, CtoS can do scheduling(re-timing) freely for making and using an intermediate result stored by a variable, so the share of the register is promoted.

3. Reading and writing signal are restricted by a wait call and CtoS cannot schedule freely. On the other hand, reading and writing variable are not restricted by a wait call.

How to Check

There is no tool to check it. Check it by designer.

Synthesis	MUST	C++

SYN25　Do not use embedded functions and libraries of C/C++

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  void my_thread() {
    ...
    int val = abs(data);
    #ifdef DEBUG
    printf("val = %d¥n", val);
    #endif
    out.write(val);
    ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  void my_thread() {
    ...
    double val = sqrt(2.0);
    FILE* fp = fopen(path, "w");
    fprintf(fp, "val = %f¥n", val);
    fclose(fp);
    std::cout << "the square root of 2 is " << val << std::endl;
    if (strcmp(str1, invalid) == 0) {
      strcpy(str1, "invalid");
      exit(0);
    }
    ...
  }
```

```
    };
```

Comment

By the restriction of the high level synthesis tool(CtoS), do not use following embedded functions and libraries of C/C++.

1. arithmetic functions (as for sqrt(), sin(). But abs() is available only)
2. input and output functions and libraries (as for printf(), fopen(), iostream class). However if there is a description by using printf() and cout class, it doesn't influent for result of synthesis because these descriptions are ignored. (sysc.c_cpp.4 of 1Team:System can check only the use of input and output functions of C as for printf(), fopen(), puts(), gets(), scanf())
3. string operation functions（as for strcpy(), strcmp())
4. exit function (sysc.synth1.33 of 1Team:System can check)
5. C++ STL (Standard Template Library)

How to Check

No.2...sysc.c_cpp.4 (1Team:System)

No.4...sysc.synth1.33 (1Team:System)

execution log of CtoS

Synthesis	MUST	C++

SYN26 There are some C/C++ syntaxes which should not be used

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  int var;
  sub_mod sub;
  SC_CTOR(mod) : sub("sub") {
    ...
  }
  void my_thread() {
    ...
  #ifdef DEBUG
    try {
      if (val > max)
        throw val;
    }
    catch (int e) {
      assert(0);
    }
  #endif
    ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  volatile int val;
  sub_mod sub;
  SC_CTOR(mod) : sub("sub", this) {
    ...
  }
  void my_thread() {
    ...
    try {
      if (val > max)
        throw val;
    }
    catch (int e) {
      assert(0);
    }
    ...
  }
};
...
static int rom_data[3] = {1, 2, 3};
```

Comment

By the restriction of the high level synthesis tool(CtoS), do not use following C++ syntax.

1. exception handling by "try-catch"

CtoS supports the following C++ syntaxes however, these are not recommended to use because of not tested at Renesas.

2. "volatile" modifier
3. static variable
4. this pointer
5. class inheritance
6. virtual function

How to Check

2...sysc.synth1.32

(1Team:System)

Synthesis	MUST	C++

SYN27 Keep some decided rules when using pointer

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  void func(sc_int<24> *a) {
    // workaround of avoiding No.2
    sc_uint<24> tmp = *a;
    tmp.range(12,8) = 0;
    *a = tmp;
    ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  sc_in<int> in;
  sc_signal<int> sig;
  int m_data;
  void my_thread() {
    ...
    int in_data = in.read();
    short* tmp = (short*)in_data; // No.1
    sc_signal<int> *tmp_sig = &sig; // No.3
    int *tmp_data = &m_data; // No.3
    int (*p) (int arg); // No.4
    ...
  }
  void func(sc_int<24> *a) {
    a->range(12, 8) = 0; // No.2
    ...
    }
  };
```

Comment

By the restriction of the high level synthesis tool(CtoS), when using pointer, the following are forbidden.

1. Forcible Cast (refer to COM27's No.1 and No.2)
2. Bit selects and part selects on pointer dereferences
3. use of pointer to signal, data member
4. use of function pointer

CtoS supports use of multi-pointer, however it is not recommended to use this feature because of not tested at Renesas.

How to Check

execution log of CtoS

use of multi-pointer...MRC-17.5 (1Team:System)

Synthesis	Required	C++

SYN28　Do not use string data, and comma operator by "for" sentence

Appropriate Example

```
SC_MODULE(mod) {
{
  ...
  void my_thread() {
    ...
    int area1[10], area2[10];
    int i;
    for (i = 0; i < 10; i++) {
      area1[i] = i;
      area2[i] = i*2;
    }
    ...
#ifdef DEBUG
    if (val > max)
      std::cerr << "Error." << std::endl;
#endif
    ...
  }
};
```

Inappropriate Example

```
SC_MODULE(mod) {
{
  ...
  void my_thread() {
    ...
    int area1[10], area2[10];
    int i, j;
    for (i = 0, j = 0; i < 10; i++, j += 2) {
      area1[i] = i;
      area2[i] = j;
    }
    ...
    if (val > max)
      std::cerr << "Error." << std::endl;
    ...
  }
};
```

Comment

You can use character and string data such as 'c' and "Hello World" only when name to port, signal and internal module.

The other uses are not recommended because high-level synthesis tool (CtoS) ignores t.

If you want to use string data for debug print etc, enclose the debug code with pre-processor and should be assumed invalid code for synthesis.

CtoS supports the use of comma operator by "for" sentence however, the use is not recommended because of not tested at Renesas.

How to Check

There is no tool to check it. Check it by designer.

Synthesis	Required	C++

SYN30　Do not use condition operator"? :"

Appropriate Example

```
if (_in0 > _in1)
  _max_in = _in0;
else
  _max_in = _in1;
```

Inappropriate Example

```
_max_in = (_in0 > _in1)? _in0: _in1;
```

Comment

To get the result of line coverage with gcov correctly, the use of condition operator"? :" is not recommended.

Because it could not be distinguished whether true case processing or false case processing was done.

However, it is possible to detect both branches by inserting a dummy description, the redundant description causes problem of warning when compiling.

How to Check

There is no tool to check it. Check it by designer.

Revision History

Rev. No	Classification	Effective date	Content	Approved By	Checked By	Written By
-	Established	Approved date	Created new	SDVT Dept. K.Morimoto Mar.12.2009	SDVT Dept Asano Mar.12.2009	SDVT Dept Oshima Watanabe Mar.12.2009
1	Revised	Approved date	Add/Delete/Revise rules -All:Unite formats of the description examples (Indent match and two-byte code character use abolition), To all existing rules, correct typo, improve contents, and review "how to check" -COM61:Add (To assume MOD3 rule to be a common rule, change to this rule) -COM62:Add (Cout out a part of the content of MOD19 rule and it is assumed this rule) -COM63:Add -MOD3:Delete (To assume this rule to be a common rule, delete this) -SYN1:Delete -SYN6:Delete (Integrate this content into SYN5 rule) -SYN29:Delete (Integrate this content into SYN26 rule)	ｼ設２技GR M.Fujii Mar.10.2010	SDVT Dept Y.Nonaka Mar.10,.2010	SDVT Dept Oshima Mar.10.2010

[責任部署名：EDA 部門(内線：826-3642)]
[Responsibility Dept.:EDA department (Ext.:826-3642)]

管理部署承認 Administration Dept. approved	承認 Approved By	審査 Checked By	作成 Written By
	シ設2技GR M.Fujii Mar.10,2010	シ設2技SK Y.Nonaka Mar.10,2010	SDVT Dept Oshima Mar.10.2010

提案書 兼 来歴表
Proposal sheet and History table

区　分 Classification	制定 Established・[✓ 改正 Revised]・廃止 Abolished
文書番号 Document Number [名　称] [Title]	RS-EZ-DM-0321-E0 Rev.1 [C++/SystemC コーディング規約] [C++/SystemC Coding Rule]
理　由 Reason	Add/Delete/Revise rules

内　容 Content

No.	項番 Item Number	概要　及び　理由 Summary and Reason
1	All	Unite formats of the description examples (Indent match and two-byte code character use abolition) To all existing rules, correct typo, improve contents, and review "how to check"
2	COM61	Add (To assume MOD3 rule to be a common rule, change to this rule)
3	COM62	Add (Cout out a part of the content of MOD19 rule and it is assumed this rule)
4	COM63	Add
5	MOD3	Delete (To assume this rule to be a common rule, delete this)
6	SYN1	Delete
7	SYN6	Delete (Integrate this content into SYN5 rule)
8	SYN29	Delete (Integrate this content into SYN26 rule)

事前調整会議 Preparation adjustment meeting	None
通知先 Distributed Dept.	全設計部門，全ＥＤＡ部門 All Design department and EDA department