SystemCを使ったシステム設計

デバイスの記事　システムの記事

第7回

続・C++の基礎知識

長谷川裕恭，小川丈博

SystemCは，C++をベースに開発されたシステム・レベル言語である．前回に続いて，C++の基礎知識について解説する．メモリ領域を確保・開放する方法，上位の階層から下位のモジュールにパラメータを与える方法，構造体を使用して複数の信号をまとめる方法，仮想関数を使って派生クラスを呼び出す方法などについて説明する．（編集部）

前回（本誌2004年2月号，pp.132-138）に続いて，今回もSystemCを記述するうえで必要となるC++の基礎知識について解説します．

● メモリ領域の確保にはnew，開放にはdelete

C言語でメモリ領域を動的に確保するときはmalloc関数を使用します．C++には，このmalloc関数と同じしごとをしてくれる演算子newが組み込まれています．new演算子では型の管理をコンパイラが行ってくれるため，malloc関数のようにキャストやsizeof関数の呼び出しが必要なく，簡単に記述することができます．

new演算子を使用してメモリ領域を動的に確保するには，以下のように記述します．

```
int* p;
p = new int; // new 型名
```

メモリの確保と同時に初期値を与える場合は，初期化子を使用して以下のように記述します．

```
int* p;
p = new int(10); // new 型名（初期値）
```

new演算子で確保したメモリ領域を開放するにはdelete演算子を使用します．上記のnew演算子で確保したメモリ領域を開放する場合は，以下のように記述します．

```
delete p; // delete ポインタ名
```

また，配列の場合は以下のように記述します．

```
int* p;
p = new int[100]; // new 型名[要素数]
...
delete [] p;       // delete [] ポインタ名
```

リスト1　ポインタを使用したモジュール・インスタンス

```
#include "test.h"
#include "adder.h"

SC_MODULE(top)
{
  sc_in_clk clk;
  sc_signal<int> i1,i2,s1;

  test*  utest;
  adder* uadder;

  SC_CTOR(top)
  {
    utest  = new test("test");
    uadder = new adder("uadder");

    utest->clk(clk);
    utest->in1(i1);
    utest->in2(i2);
    utest->sum(s1);
    uadder->in1(i1);
    uadder->in2(i2);
    uadder->sum(s1);
  }

 ~top()
  {
    delete utest;
    delete uadder;
  }
};
```

①モジュールをポインタで宣言

②new演算子を使用して，ポインタを初期化

③モジュールをポインタで宣言した場合は，ポート接続にアロー演算子（->）を使用する

④デストラクタ

⑤new演算子で割り当てたメモリはdelete演算子で開放する

リスト2　コンストラクタを使用してパラメータを与える（コンストラクタの定義）

```
#include <systemc.h>

SC_MODULE( memory )
{
  sc_in<unsigned> address;
  sc_in<bool>     ceb,web;
  sc_in<int>      datai;
  sc_out<int>     datao;

  SC_HAS_PROCESS( memory );

  memory( sc_module_name name, int size ) : sc_module(name),
                                            m_size( size )
  {
    m_mem = new int[size];
    SC_METHOD( process );
    sensitive << address << ceb << web << datai;
  }

  ~memory() { delete [] m_mem; }

protected:
  void process()
  {
    if ( address > m_size - 1 ){
      cout << "address error" << endl;
      return;
    } else {
      if ( !ceb ) {
        if ( !web )
          m_mem[address] = datai;
        else
          datao = m_mem[address];
      } else
        datao = 0;
    }
  }

protected:
  int  m_size;
  int* m_mem;
};
```

①コンストラクタ内にプロセス宣言がある場合は，SC_HAS_PROCESSが必要

②メンバ変数を初期化する場合は，コンストラクタ初期化子を使用

③上位階層からパラメータ値を受け取るため，コンストラクタに引き数を追加

④受け取ったパラメータ値をメンバ変数で使用

⑤パラメータ値を保存するためのメンバ変数

リスト1は，前回に紹介したadderモジュールとこれをテストするためのtestモジュールをtopモジュールで接続した記述です．

リスト1の①では，testモジュールとadderモジュールをtopモジュールからポインタを使用してインスタンスしています．モジュールをポインタでインスタンスする場合は，まず，モジュールをポインタとして宣言します．

次に，コンストラクタの中でnew演算子を使用してモジュールを初期化します（②）．このとき，testモジュールとadderモジュールのコンストラクタが呼び出されるので，初期値としてインスタンス名を与える必要があります．

モジュールをポインタでインスタンスした場合は，メンバ変数やメンバ関数にアクセスする際に，ドット演算子（.）ではなく，アロー演算子（->）を使用します．③がアロー演算子を使用したポート接続になります．

メモリ領域を動的に確保した場合，不要になった段階で

リスト3　コンストラクタを使用してパラメータを与える（コンストラクタの呼び出し）

```
#include "memory.h"
#include "test.h"

SC_MODULE( top )
{
  sc_in_clk clk;

  sc_signal<unsigned> address;
  sc_signal<bool>     ceb,web;
  sc_signal<int>      datai,datao;

  memory umemory;
  test   utest;

  SC_CTOR( top ) : umemory("umemory", 0x100), utest("utest")
  {
    umemory.address(address);
    umemory.ceb(ceb);
    umemory.web(web);
    umemory.datai(datai);
    umemory.datao(datao);

    utest.clk(clk);
    utest.address(address);
    utest.ceb(ceb);
    utest.web(web);
    utest.datai(datai);
    utest.datao(datao);
  }
};
```

①パラメータ（メモリ・サイズ）をコンストラクタ引き数に指定

これを開放することができます．コンストラクタによってメモリ領域を確保した場合，デストラクタと言われるものでメモリ領域を開放します．デストラクタは，モジュール（クラス）が消滅する（スコープから外れる）際に呼び出される関数です．ここに，メモリを開放するためのdelete演算子を記述します．

④がtopモジュールのデストラクタの記述になります．この中で，newで確保したtestモジュールとadderモジュールのメモリ領域をdelete演算子を使用して開放しています（⑤）．

SC_MODULEは，sc_mainの中で静的にインスタンスされています．このため，デストラクタが呼ばれるのはsc_main終了時のみとなります．したがって，実際には，SC_MODULEを開放する意味はありません．

● コンストラクタを使用してパラメータを与える

上位の階層から下位のモジュールにパラメータを与える方法は2通りあります．一つはコンストラクタを使用する方法で，もう一つはテンプレート（詳細は後述）を使用する方法です．

リスト2の記述は，上位の階層からコンストラクタの引き数でパラメータを与えたものです．メモリのワード数（サ