Frédéric Boulanger
(Frederic.Boulanger@supelec.fr)
Supélec - Service Informatique
Plateau de Moulon
F - 91192 Gif-sur-Yvette Cedex

OCC++ version 4.8.1

OC to C++ translator

Available by anonymous ftp at:
`ftp://ftp.supelec.fr//pub/cs/distrib/occ++`
and on:
`http://wwwdi.supelec.fr/fb/recherche.html`

Version 4.8.1 January 2010

Contents

1 Syntax

```
occ++ [-v] [-version] [-B baseName] [-D outDir]
      [-H defFilename | -Hnone] [-cut nbAct] [-debug]
      [-tab | -tabB stateTableBasename] [-mdf | -mdfB mdfFilename]
      [-setparam] [-param] [-noref] [-inc libSyncInterface] [-awaited]
      [-notick] [-access] [-info] ocFilename
```

OCC++ is normally called by the Esterel compiler. To compile 'foo.strl' into C++, use the command: `esterel -Ac++ foo.strl` or `esterel -Sc++ foo.strl`. To give options to OCC++ on the esterel command line, put them in a quoted string after a colon, as in: `esterel -Ac++:"occ++ options" foo.strl`.

2 Description

OCC++ translates the oc code of an Esterel module or a Lustre node into a C++ class. Versions 2, 3, 4 without tasks, and 5 of oc are accepted. OC 5 code from Esterel v5.01 (with initialized signals) is now supported.

The name of the class is the name of the module. OCC++ produces two files:

***moduleName*.H** contains the interface of class *moduleName*.

***moduleName*.C** contains the code of class *moduleName*.

The class *moduleName* produced by OCC++ is a subclass of `Esterel`. Class `Esterel` is defined in libSync.H and libSync.a (see libSync).

Note: classes produced by OCC++ are always subclasses of Esterel, even if the oc code was produced by Lustre. Class Esterel should be renamed OcMachine, but keeps its name for historical reasons.

If there are several root modules in the oc source file, the base name of the files is the name of the first root module. These files will contain the interface and the code of all the corresponding C++ classes. For example, if 'foo.strl' describes the module `foo` and the module `bar`, 'foo.H' will contain the interface of class `foo` and the interface of class `bar`.

If the module uses external declarations (types, functions, constants or procedures) a file *moduleName*Def.H is `#included` by *moduleName*.H

3 Options

The following options are supported:

-v verbose output. With this option, OCC++ will show what it sees in the oc source file (submodules, root module, signals, variables etc...)

-version prints the version number of OCC++ (currently version 4.8.1)

-B *baseName* set the base name for output files. For example, the command `occ++ -B bar foo.oc` will produce 'bar.H' and 'bar.C'. If the name of the module defined in 'foo.oc' is foo, the name of the class will be 'foo' because the -B option only sets the base name of the files.

Note that the esterel compiler uses this option to give the basename of the output files to OCC++. So, the command `esterel -Ac++:"-B bar" foo.strl` will not produce 'bar.H' and 'bar.C' because esterel forces the `-B foo` option.

-D *outDir* set the output directory. The files *moduleName*.H and *moduleName*.C will be placed in the directory *outDir*.

-H *defFilename* set the name of the auxiliary definition file. If the module uses external definitions, *moduleName*.H will `#include` *defFilename* instead of *moduleName*Def.H

-Hnone do not use an auxiliary definition file. *moduleName*.H will not `#include` an external definition file, even if the module uses external definitions. This option allows the use of constants, types, functions and procedures of the C++ class in the Esterel code. Those entities are declared in the definition of the class and do not need to be declared as external definitions.
Note that the -Hnone option is inherited from previous versions of OCC++ and will probably disappear since a new scheme is developed to support built-in types, C++ operators and member access.

-cut *nbAct* cut long lines in state tables to *nbAct* actions per line. This option is useful for state machines with numerous actions to avoid too long lines. (some preprocessors do not support very long lines).

-debug insert debugging code in the state machine. This code will be compiled if the symbol `_SYNC_DEBUG_` is defined. The debugging code will print the name of the module and the current operation on the standard error file. The debugged operations are:

- reset of output signals. In this phase, the module resets its output signals to the non-present state.
- reset of input signals. In this phase, the module resets its input signals to the non-present state.
- set of input signals. In this phase, the module gets the value of its input signals. The debugging code will display the value of each input signal.
- state machine actions. The debugging code will display the name of the module, the current state and the name of the action. The name of the actions are `k`*`Type`*`_`*`argument`*`_`*`actionNumber`*. For example if action 3 is the output of signal `toc`, its name will be `kOutput_toc_3`.

-tab produces state and dag tables in separate files. OCC++ will place each state table in a separate file named *moduleName*Tab*stateNum*.C, where *stateNum* is the number of the corresponding state, and each dag table in a separate file named *moduleName*Dag*dagnum*.C, where dagnum is the number of the dag. This option is useful for huge state machines because some preprocessors do not handle very big files. Furthermore, it makes it easier to edit the C++ file.

-tabB *baseName* same as -tab, but set a prefix for tab file names. For example, the command `esterel -Ac++:"-tabB moof" foo.strl` will produce 'foo.H', 'foo.C' and the files 'mooffooTab*stateNum*.C' and 'mooffooDag*dagnum*.C'. This option can be used to store the table files in a specific directory, for example with `esterel -Lc++:"-tabB tables/" foo.strl`, which will store the table files in the directory 'tables'.

-mdf produce a Module Description File. OCC++ will produce a file named *moduleName*.mdf that contains a description of the module to be used with **mdlc**. The name of the mdf file is not affected by the -B or -D options.

-mdfB *mdfBaseName* same as -mdf, but set a prefix for the name of the mdf file. This option works for mdf files just the same as -tabB for table files.

-param consider Esterel constants as parameters for the constructor. The default is to consider Esterel constants as class constants (static const members of the class). With this option, Esterel constants are considered as const members, and must be initialized when a new instance is created. The initialization is done with an additional parameter of the constructor. This parameter is a reference on a structure of type *moduleName*Consts, which is defined in *moduleName*.H. This structure contains members of the same type and the same name as each constant used by the module. For example, if the module foo uses two constants: moof, of type integer and waaf of type bar, we will get:

```
struct fooConsts {
  integer moof;
  bar waaf;
};
```

-setparam consider Esterel constants as settable parameters of the object. With this option, the constructor does not take the additional initialisation parameter used with -param. Instead, the C++ class has two new methods to set and get the value of the parameters:

```
virtual void setParams(const moduleNameConsts& newParams);
virtual void getParams(moduleNameConsts& theParams) const;
```

where *moduleName*Consts is the same structure as with -param.

Warning: With this option, the behavior of the object may not be exactly the same as the behavior of the Esterel module or Lustre node. Differences may appear if you modify the parameters of an object that has already reacted to events. The behavior will be correct only if the `setParams()` method is called on new objects.

-noref Asks OCC++ not to use C++ references to pass argument by reference in procedure calls. The default is to use references, so that the Esterel procedure `myproc(int a)(int b)` should be declared as `void myproc(int& a, int b)` in C++. With the -noref option, pointers are used instead of references, so the same procedure should be declared as `void myproc(int *a, int b)`.

-inc ***libSyncInterface*** Use libSyncInterface as the name of the file that gives the interface of the Synchronous Library. The default value is <Esterel/libSync.H>. The command `occ++ -inc libSync.H x.oc` or `occ++ -inc '"libSync.H"' x.oc` will use "libSync.H". The command `occ++ -inc "<anotherFile.H>" x.oc` will use <anotherFile.H>. Be sure to quote the name '<xxx.H>' to prevent the shell from interpreting < and > as input/output redirection.

-awaited Add a `waitSigList()` method to the class if the oc code contains information on awaited input signals. This method returns a null terminated list of the awaited input signals in the current state of the module. If there is no information on awaited signals in the oc code, or if this option is not given, the `waitSigList()` method is not defined in the class. In versions of libSync greater or equal to 4.1, this method is defined in class `Synchronous` and returns the list of all input signals as a default behavior.

-notick Overload the `activate()` method of class `Esterel` so that the automaton engine is not called when no input signal is present. The module will not receive the `tick` signal if no other input signal is present. Do *not* use this option for a module that uses the Esterel `tick` signal.

-access Display access rights to OCC++ (included for Esterel compatibility).

-info Display information about the compilation of OCC++ (when, where, by whom).

4 Interface of the C++ classes produced by OCC++

The C++ classes produced by OCC++ are subclasses of `Esterel`, which is itself a subclass of `Synchronous`. Those classes are defined in libSync.H and libSync.a. For example, the following Esterel module:

```
module foo:
  input inp(integer);
  output out(integer);

  every inp do
    emit out(2 * ?inp);
  end every
end module
```

becomes the following class through OCC++:

```
class foo : public Esterel {
  protected:
    enum {                      // Automaton action numbers
      kSequence,
      kExecuteODAG,
      kGotoCDAG,
      kBranchAlways,
      kPresent_inp_4,
      kOutput_out_5,
      kCall__assign_integer_6
```

```
    };

    static const Esterel::tStateNumber *const *const cfooStates;
    void doAction(Esterel::tActionNumber op, Esterel::tActionNumber *i);
    InSignal<integer> finp;
    OutSignal<integer> fout;

  public:
    virtual InSignal<integer>& inp();
    virtual OutSignal<integer>& out();
    virtual void resetOutputs();
    virtual void resetInputs();
    virtual void setInputs();
    virtual const InputSignal** inSigList() const;
        // returns a null terminated list of input signals
    virtual const OutputSignal** outSigList() const;
        // returns a null terminated list of output signals
    virtual const InputSignal** waitSigList() const;
        // returns a null terminated list of awaited signals
    foo(const char* name = "foo", const boolean schedule = true);
                                  // Constructor
    ~foo();                       // Destructor
  private:
    foo(const foo &);             // Private copy constructor
};
```

In the protected part, the enum gives the names of the actions of the state machine. This state machine is described in `cfooStates` which is the table of the actions to do for each state of the machine. The automaton engine is provided by the class `Esterel`.

The `doAction()` method is called by the automaton engine to execute the actions associated to the action numbers of the state table (and of the dag table if there are dags).

`finp` and `fout` are the signals of the module. Their class is a template which is instantiated with the type of the signal (`integer` in our example).

If the state of the module is not limited to the state of its automaton, a pointer named `pf_backup_copy_` is used to point to the storage area that contains the saved state of the module. This area will be allocated when `backup()` is called for the first time, and released when the module is destroyed. The `Esterel` class manages the backup and restore of the state machine.

In the public part, the methods `inp()` and `out()` give access to the signals of the module.

The `setInputs()` method fetches the value of the input signals of the module.

The `resetOutputs()` method is called to reset the output signals of the module to their non-present state.

The `resetInputs()` method is called to reset the input signals of the module to their non-present state. This is necessary to reset input signals that have lost their source and will not be updated.

The `inSigList()` method yields a null terminated list of these input signals. In this list, signals are considered as instances of `InputSignal`, because they are only used to walk through the interconnection graph of the modules.

The `outSigList()` method yields a null terminated list of these output signals. In this list, signals are considered as instances of `OutputSignal`, because they are only used to know how many signal are connected to them (`cref()` method).

The `waitSigList()` method yields a null terminated list of the awaited input signals. This method is defined only if the -awaited option was given and there is information on which signals are awaited in each state in the oc source code.

The `backup()` method calls the backup method of class `Esterel`, allocates the backup storage area if it has not been allocated yet, and saves the current state of the module in this area. This method is defined only if the state of the module is not limited to the state of the automaton.

The `restore()` method calls the restore method of class `Esterel` and sets the state of the module to the saved state. If there is no saved state, `restore()` does nothing. This method is defined only if the state of the module is not limited to the state of the automaton.

The constructor `foo()` is used to build new instances of `foo`. The first argument is the name of the object. This name is mainly used for debugging purposes and is optional. The second argument is optional too, and tells if we want the new object to register itself on the current clock and to be automatically scheduled by this clock. If the module uses constants with the -param option, the first argument is a reference on a struct `fooConsts` that contains the values of the constants. `name` and `schedule` become the second and third arguments.

The copy constructor is declared private to prohibit its use. Copying synchronous objects has no meaning. Since no `operator=` is declared, synchronous objects cannot be assigned either.

5 Interface to asynchronous tasks (execs)

OCC++ assumes that each exec in an Esterel module is an instance of a task class. The name of the class is built from the name of the module and the name of the Esterel task as *moduleName*`Task_`*taskName*. For example, if a module `foo` has a task `T`, the class for the execs of this task will be named `fooTask_T`. This class must be defined in *moduleName*Def.H or any other file specified with the -H option. Each exec of a task is an instance of this class, and the name of this instance is *taskName*`_`*taskNum*`_`*execNum*.

A task class must implement the following required interface:

```
"Class"(unsigned taskNum,unsigned execNum,const char* name);
                                // Constructor
void tstart(...);               // implements the Esterel start action
void tkill();                   // implements the Esterel kill action
void tsuspend();                // implements the Esterel suspend action
void tactivate();               // implements the Esterel activate action
void treturn();                 // implements the Esterel return action
void setReturn();               // set value of return signal
```

```
PureOutSignal& returnSig(); or OutSignal<"sigType">& returnSig();
                                  // access method to the return signal.
```

The `taskNum` and `execNum` arguments to the constructor are the task number (each task has a unique number in an Esterel module) and the exec number (each exec has a unique number in a module).

The `tstart` method takes arguments as specified by the task declaration in the Esterel module. For instance, if you declared:

`task add(integer)(integer,integer)`

the `tstart` method must be declared as:

`void tstart(integer& x, integer a, integer b);`

If you invoked OCC++ with the -noref option, the reference `x` becomes a pointer.

The `tstart` method must start the execution of the task with the given parameters. Note that the value of the reference arguments should not be modified until the `treturn` method is called. So you will need to work on a copy of those arguments.

The `tkill` method must kill the exec, if possible. This is OS dependent. The return signal of a killed exec should not be emitted.

The `tsuspend` method must suspend the execution of the task, if possible. This is OS dependent. The return signal of a suspended exec should not be emitted.

The `tactivate` method must re-activate a suspended execution.

The `treturn` method is called by the Esterel module when it has received the return signal of the exec. This method must update the reference arguments of the exec (they should not have been modified before this).

The `setReturn` method must set the return signal according to the status of the exec.

The `returnSig` method must return a reference to the return signal of the exec. This signal is emitted by the exec to inform the Esterel module of the completion of this exec. To handle the emission of this signal, you should store the status of the exec in a variable, and emit the return signal in the `setReturn` method when the status indicates that the exec has returned, has not been killed, and is not suspended.

Note that with the implementation of asynchronous tasks in OCC++, a task must have return signals of the same type for all its execs. This is not enforced by the Esterel compiler.

6 The future of asynchronous tasks

In a future release of libSync, there may be an `AsyncTask` abstract class for asynchronous tasks. This class should handle generic operations for asynchronous tasks (status maintenance, ...). Subclasses of `AsyncTask` may implement asynchronous tasks with different mechanisms (Unix processes, threads, RPC, ...), and a particular task could inherit from those classes and just implement what is specific to the job it is designed for.

7 Differences between occ and OCC++

The C++ code produced by OCC++ uses the standard C++ operators. So, it is not necessary to define the set of functions that implement these operators as with the C code produced by occ.

For example, the `_TYPE` function used by occ to assign a value to a variable of type `TYPE` is replaced by the `=` C++ operator, the `_eq_TYPE` function is replaced by the `==` operator, the `_cond_TYPE` function is replaced by the `?:` operator and so on. Of course, these operators must be defined for the corresponding class. The `string` type of Esterel and Lustre is implemented by the `String` class in libSync. This class defines the standard operators (`=`, `==`, >, >=, <, <=) for strings. The `boolean` type of Lustre and Esterel is implemented by the `boolean` class in libSync. This class defines the logical operators `!`, `&&` and `||`, but since the evaluation of `&&` and `||` does not require the evaluation of their two arguments, two methods `EvalBoth_AND` and `EvalBoth_OR` are defined in class `Esterel` to force this evaluation when it is required.

A word about simulation: there is no need to build synchronous classes in a special way to build simulators. To simulate a system of synchronous objects, you just have to connect the inputs of the system to instances of `InputModule`, and the outputs of the system to instances of `OutputModule` (see libSync). The `_text_to_TYPE`, `_TYPE_to_text` and `check_TYPE` functions that were required by the simulation library of occ are no longer required with OCC++. Input and output of values from a text stream rely on the << and >> C++ operators. Checking that a string is suitable for a type is the task of the `InputModule`. What you have to do is to define the << and >> operators for any class that you define and want to use for signal values. These operators are already defined for the basic C++ types (`int`, `char`, ...), and libSync defines them for `boolean` and `String` (which match the `boolean` and `string` types of Esterel and Lustre).

See the mdlc documentation for building text oriented, Motif or protocol based simulation programs.

8 Support for C++ types and operators

New in OCC++ **v4.8** a special naming scheme has been developed to support C++ built-in types, operators and member access.

8.1 Types

Type names begining with `CC_` are recognized as special types. First, the `CC_` prefix is removed, then, if what follows is `ptr_`, the type is a pointer type, if what follows is `ref_`, the type is a reference type. `'_'` are replaced by spaces in the remaining part of the type name.

For example, the Esterel type name `CC_ptr_unsigned_long` is translated into `unsigned long*`

The type name `TypeOfThis` is translated into the type of the generated class. So `CC_ptr_TypeOfThis` is the type of the built-in C++ `this` pointer.

8.2 Constants

You may declare an Esterel constant named `this` to refer to the instance of the synchronous class. OCC++ knows about it and will not look for an external definition of this constant.

8.3 Procedures

All procedures begining with `CC_op_` are recognized as C++ operators. The syntax is: `CC_op_<type name>_<op tag>` where `<type name>` is the name of the type for which the operator is defined, and `<op tag>` is one of:

`inc`	for the	`++`	postfix operator
`dec`	for the	`--`	postfix operator
`pinc`	for the	`++`	prefix operator
`pdec`	for the	`--`	prefix operator
`addd`	for the	`+=`	operator
`subb`	for the	`-=`	operator
`mull`	for the	`*=`	operator
`modd`	for the	`%=`	operator
`divv`	for the	`/=`	operator
`bitorr`	for the	`|=`	operator
`bitandd`	for the	`&=`	operator

For example, if `i` is an integer, `CC_op_int_addd(i,3)` is translated into `i += 3` by OCC++.

8.4 Functions

All functions begining with `CC_op_` are recognized as C++ operators. The syntax is: `CC_op_<type name>_<op tag>` where `<type name>` is the name of the type for which the operator is defined, and `<op tag>` is one of:

`eq`	for the	`==`	binary operator
`ne`	for the	`!=`	binary operator
`not`	for the	`!`	unary operator
`or`	for the	`||`	binary operator
`and`	for the	`&&`	binary operator
`lt`	for the	`<`	binary operator
`le`	for the	`<=`	binary operator
`gt`	for the	`>`	binary operator
`ge`	for the	`>=`	binary operator
`add`	for the	`+`	binary operator
`sub`	for the	`-`	binary operator
`mul`	for the	`*`	binary operator
`mod`	for the	`%`	binary operator
`div`	for the	`/`	binary operator
`opp`	for the	`-`	unary operator
`bitor`	for the	`|`	binary operator
`bitand`	for the	`&`	binary operator
`bitnot`	for the	`~`	unary operator
`index`	for the	`[]`	binary operator
`ls`	for the	`<<`	binary operator
`rs`	for the	`>>`	binary operator

For example, if tab is an array of integers, `CC_op_int_index(tab,3)` is translated into `tab[3]` by OCC++.

All functions begining with `CC_cons_` are recognized as constructor calls. For example, if you declare an Esterel function

```
CC_cons_Complex(integer,integer):Complex
```

and c is a complex variable, the Esterel statement

```
c := CC_cons_Complex(0,-1);
```

will be translated into

```
c = Complex(0,-1);
```

by OCC++.

All functions begining with `CC_arrow_` or `CC_dot_` are recognized respectively as operator `->` and operator `.`. If the function takes only one argument, it is considered to be a member access (no function call), else it is considered to be a member function call, the first argument (this) being skipped. This scheme does not allow the call of a member function that takes no argument. This may be fixed in a future version with a special `CC_call_` prefix.

9 Installation

You may install OCC++ in any directory that is in the path of your shell. However, if you want to be able to specify C++ as a target language to Esterel, you should install occ++ (or best: a symbolic link to it) in the distribution directory of Esterel which is `/usr/local/esterelvX_XX` by default. You can check your installation with the `esterel -version` command: if OCC++ appears in the version list, OCC++ is correctly installed.

10 Disclaimer and Copyright

Frédéric Boulanger <Frederic.Boulanger@supelec.fr>
Supélec - Service Informatique
Plateau de Moulon, F - 91192 Gif-sur-Yvette Cedex