Esterel V5 Syntax

% *short comment*
%{ *multilines comments* **}%**

1. Statements

module *<name>***:**		module definition
<interface-declaration>		
<statement>		
end module		
[*<statement>* **]**		bracketed statement, to remove ambiguity
<statement> **;** *<statement>*		sequence statement
<statement> **||** *<statement>*		parallel statement (concurrent threads)
<variable> **:=** *<expression>*		variable assignment
nothing		empty statement, terminates instantaneously
pause ≡ **await tick**		pause for one instant
halt		never terminates
emit *<signal>*		signal emission, terminates instantaneously
sustain *<siqnal>* ≡ **loop emit** *<signal>* **each tick**		continuous emission of a signal
loop		infinite loop
<statement>		
end loop		
repeat *<expression>* **times**		finite loop
<statement>		
end repeat		
loop	**loop abort**	temporal loop, with statement initially started
<statement> ≡	*<statement>* **; halt**	
each *<delay>*	**when** *<delay>* **end loop**	
every *<delay>* **do**	**await** *<delay>* **;**	temporal loop, with initial waiting
<statement> ≡	**loop**	
end every	*<statement>*	
	each *<delay>*	
present *<signal-expression>*		branching according to the value of the signal expression
[**then** *<statement>]*		
[**else** *<statement>]*		
end present		
present		multiple branching according to the value of the signal expressions
case *<signal-expression>* **do** *<statement>*		
…		
end present		
if *<Boolean-expression>*		branching according to the value of the Boolean expression
[**then** *<statement>]*		
[**else** *<statement>]*		
end if		
if *<Boolean-expression>* **then** *<statement>*		multiple branching according to the value of the Boolean expressions
elsif *<Boolean-expression>* **then** *<statement>*		
…		
[**else** *<statement>]*		
end if		

await *<delay>*	terminates when the delay elapses (could be instantaneous for an immediate zero delay)
await	multiple waking; the first delay in the list which terminates executes its statement
case *<delay>* **do** *<statement>*	
…	
end await	
[**weak** *]* **abort**	abortion; kills the statement when the delay elapses (could be instantaneous for an immediate zero delay); the weak variant executes the statement a last time when the delay elapses before killing it
<statement>	
when *<delay>*	
[**weak** *]* **abort**	multiple case abortion (strong or weak)
<statement>	
when	
case <delay> **do** *<statement>*	
…	
end *[* **weak** *]* **abort**	
suspend *<statement>* **when** *<delay>*	freeze the execution of the statement when the delay expression is true
trap *<exception-list>* **in**	mechanism to catch exceptions raised by the body statement
<statement>	
[**handle** *<exception-condition>* **do** *<statement>]*	
end trap	
exit *<exception>*	raises an exception
run *<module>[* **[signal** *<new>/<old>,…* **;**	creates an instance of a module with renamings of the interface; i.e., connects the current objects with the objects of the module interface; if an object is not renamed, it is implicitly connected to the object with the same name
constant *<new>/<old>,…* **;**	
function *<new>/<old>,…* **;**	
procedure *<new>/<oJd>,…* **;**	
type *<new>/<old>,…* **;**	
task *<new>/<old>,…* **]** *]*	
call *<procedure>*	synchronous execution of a external procedure
exec *<task>* **return** <signal>	asynchronous execution of an external task
exec	simultaneous execution of several external tasks
case *<task>* **return** <signal>	
…	
end exec	
signal *<local-signal-declaration-list>* **in**	local declaration of a signal
<statement>	
end signal	
var *<local-variables-declaration-list>* **in**	local declaration of a variable
<statement>	
end var	

2. Declarations

input *<signal> [[***:=** *<expression>]***:** *<type-combine>]* **,** *...***;**
output <signal> *[[***:=** *<expression>]***:** *<type-combine>]* **,** *...***;**
inputoutput <signal> *[[***:=** *<expression>], <type-combine>]* **,** *...***;**
relation <signal> **=>** *<signal>***,** <signal> **#** <signal> **#** *....* **,** *...***;**
return <signal> *[[***:=** *<expression>]***:** *<type-combine>]* **,** *...***;**
sensor *<sensor>***:** *<type>* **,** *...***;**

interface of a module, signals and sensors declaration; relations of implication or *exclusion* between signals

constant *<name>***:** *<type>, <name>* **=** *<value>***:** *<type>* **,** *...***;**
type *<type>* **,** *...***;**
function *<name>* **(***<type-by-val>***,** *...* **):***<type-by-val>* **;**
procedure *<name>* **(***<type-by-ref>***,** *...* **) (***<type-by-val>***,** *...* **);**
task *<name>* **(***<type-by-ref>***,** *...* **) (***<type-by-val>***,** *...* **);**

interface of a module, objects externally defined (e.g., in the user C code)

3. Expressions

- * / mod + <= >= = <> ( ) arithmetic operators
not and or ( ) Boolean operators
*[***immediate***] <expression>* <signal> delay expression
*[***immediate***] <expression> [<Boolean-signal-expression>]*
combine *<type>* **with** *<function-or-operator>* how to combine several simultaneous signal presence