An Overview on VHDL2SC1.0 Translator

Università di Verona
Dipartimento di Informatica
Verona, ITALY

1 VHDL2SC1.0 Introduction

VHDL2SC1.0 is an automatic tool which translates VHDL models into equivalent SystemC models. VHDL2SC1.0 extends the SAVANT environment (see http://www.ececs.uc.edu/ paw/savant/) adding the ability to translate the In-memory Intermediate Rapresentation (IIR) created by the SAVANT's VHDL parser into SystemC code.
The current version of VHDL2SC has been developed considering SystemC 1.0.2, the operative system *Linux RedHat 7.0* and the c++ compiler *gcc-2.96*. We are working to modify the translator in order to obtain SystemC code consistent with release 2.0 (see the section VHDL2SC State of Art and Actual Limitations).
VHDL to SystemC translation rules will be presented in the next sections with the actual limitations of the tool, the installation instructions and the command line usage.
Other information about the translator could be found in the paper "On the Reuse of VHDL Modules into SystemC Designs" presented at Forum on Design Language, Lyon, September 3-7 2001 (http://www.systemc.org/technical_papers.htm).

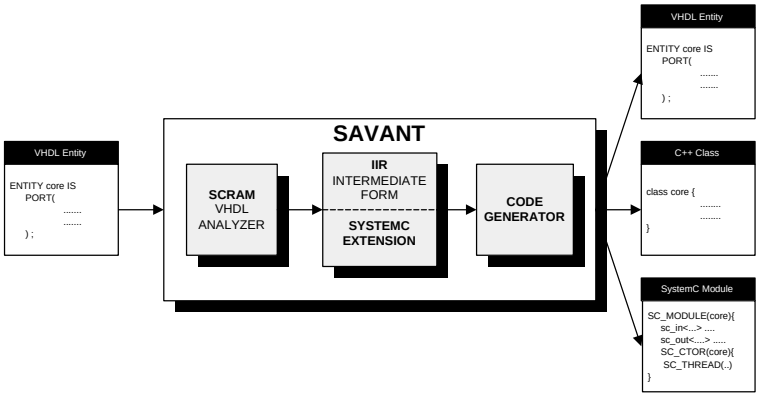

Figure 1: Program architecture based on the Savant environment.

2 VHDL to SystemC Translation Rules

To obtain the VHDL to SystemC translation a set of translation rules has been developed and implemented in C++ classes extending the SAVANT's IIR hierarchy, but to allow an easy extension

of VHDL2SC1.0 a *Rules.dat* file has been created. In the next paragraphs the translation rules and how to use the Rules.dat file will be expalined.

2.1 Rule's Lecture Guide

In section 2.2 the translation tables from VHDL constructs to SystemC ones will be presented. For every construct the following information are explained:

- Description
- Syntax
- Example
- Notes (optional)

Figure 2 shows the table describing how to translate a VHDL instruction into the corresponding SystemC instruction. The italic font is used for the language keywords.
Note that the tables do not include information about the semantics of the VHDL and SystemC constructs.

Description: Brief description of the construct.

Syntax
VHDL
VHDL Syntax
SystemC
SystemC Syntax

Example
VHDL
`VHDL Example`
SystemC
`SystemC Example`

Notes: Notes related to the construct

Figure 2: Template of the translation table from VHDL to SystemC.

2.2 The Translation Rules

Description: Entity definition.

Sintax
VHDL
entity entity_name *is* entity's declarations *end* entity_name;
SystemC
SC_MODULE (entity_name){ entity's declarations };

Example
VHDL
`entity gcd is` `  port(xi :  in unsigned(7 downto 0);` `  yi :  in unsigned(7 downto 0);` `  clk:  bit;` `  output :  out unsigned(7 downto 0));` `end gcd;`
SystemC
`SC_MODULE(gcd){` `  sc_in <sc_uint<8>> xi;` `  sc_in <sc_uint<8>> yi;` `  sc_in <sc_bit> clk;` `  sc_out <sc_uint<8>> out;` `};`

Description: Entity definition with generic constants.

Sintax
VHDL
entity entity_name *is*   *generic* (variable1 : type1:=default1;   …;   variableN : typeN:=defaultN);   entity's declarations *end* entity_name;
SystemC
template<type1 variable1=default1,   …,   type2 variable2=default2 > *SC_MODULE* (entity_name){   entity's declarations };

Example
VHDL
`entity clock_gen is` `  generic(tpw :  integer; value :  integer:=100);` `  ⋮` `end clock_gen;`
SystemC
`template <int tpw=0,int value=100>` `SC_MODULE(clock_gen){` `  ⋮` `};`

Notes: The default values are optionals. When only one default value is specified in VHDL with more than just one generics, all values must be specified in the SystemC translation.

Description: Architecture definition.

Sintax
VHDL
architecture architecture_name *of* entity_name *is* architecture's declarations *begin* architecture's instructions *end* architecture_name;
SystemC
SC_MODULE (entity_name){ architecture's declarations *SC_CTOR*(entity_name){ *SC_<Process Type>* (proc_id1); sensitivity_list1 *SC_<Process Type>* (proc_id2); sensitivity_list2 ... } }; architecture's instructions

Example
VHDL
`architecture arch of gcd is` `begin` `  evalGDC:process(clk)` `  begin` `  ...` `  end process;` `end arch;`
SystemC
`SC_MODULE(gcd){` `  void evalGCD();` `  SC_CTOR(gcd){` `  SC_METHOD(evalGCD);` `  sensitive << clk;` `} };` `void gcd::evalGCD(){ ...}`

Notes: <*Process Type*> could be `METHOD`, `THREAD` or `CTHREAD`.
`SC_CTHREAD` hasn't got the *sensitivity list*.

Description: Process *Sensitivity list*.

Sintax
VHDL
process_id:*process*(signal_id1,signal_id2,...)
SystemC
SC_<Process Type>(process_id); *sensitive*<<signal_id1 <<signal_id2...;

Example
VHDL
`evalGCD:process(clk)`
SystemC
`SC_METHOD(evalGCD);` `sensitive<< clk;`

Description: Process definition.

Sintax
VHDL
process_id:*process*(sensitivity_list)   process's declarations *begin*   process's instructions *end process*;
SystemC
void entity_name::process_id(){   process's declarations   process's instructions }

Example
VHDL
`evalGCD : process(clk)` `  variable x, y, temp :  unsigned (7 downto 0);` `begin` `  ...` `end process;`
SystemC
`void gcd::evalGCD(){` `  ...` }

Notes: The variables included into a SystemC process must be declared into the declaration part of the corresponding SC_MODULE in order to preserve their value between different execution of the process during the same simulation session.

Description: Signal definition.

Sintax
VHDL
signal signal_id : signal_type;
SystemC
sc_signal < signal_type > signal_id;

Example
VHDL
`signal x :  unsigned(31 downto 0);`
SystemC
`sc_signal<sc_uint<32> > x;`

Description: Signal attributs.

Sintax
VHDL
signal_id'*event* signal_id'*length*
SystemC
signal_id.*event*() signal_id.*length*()

Example
VHDL
`if(clock'event) then...`
SystemC
`if(clock.event())...`

Description: Port declaration.

Sintax
VHDL
port(port_id1 : *in* port_type); *port*(port_id2 : *out* port_type); *port*(port_id3 : *inout* port_type);
SystemC
sc_in<port_type> port_id1; *sc_out*<port_type> port_id2; *sc_inout*<port_type> port_id3;

Example
VHDL
`port(xi :  in unsigned (7 downto 0);` `     yi :  in unsigned (7 downto 0);` `     clk :  in bit;` `     output :  out unsigned (7 downto 0));`
SystemC
`sc_in<sc_uint<8>>    xi;` `sc_in<sc_uint<8>>    yi;` `sc_in<sc_bit>        clk;` `sc_out<sc_uint<8>>  output;`

Description: Component instantiation.

Sintax
VHDL
for component_instance : component_name *use entity* entity_name (architecture_name);
SystemC
entity_name component_instance(“entity_name”);

Example
VHDL
`for utpg : dtpg use entity work.gcd(bhv);`
SystemC
`gcd utpg("gcd");`

Description: Component’s generic variables definition.

Sintax
VHDL
c_instance : c_name *generic map*(val1, . . . , valN);
SystemC
entity_name<val1, . . . , valN> c_instance(“entity_name”);

Example
VHDL
`cg : clock_genl generic map(10,15);`
SystemC
`clock_gen<10,15> cg("clock_gen");`

Description: Explicit named connection between signal and port.

Sintax
VHDL
component_instance : component_name *port map*(port_a=>signal_1, port_b=>signal2, ...);
SystemC
component_instance.port_a(signal_1); component_instance.port_b(signal_2); ...

Example
VHDL
`utpg : dtpg port map(yo=>opy,clock=>clk,` `xo=>opx);`
SystemC
`utpg.yo(opy);` `utpg.clock(clk);` `utpg.xo(opx);`

Description: Positional connection between port and signal.

Sintax
VHDL
component_instance : component_name *port map*(signal1, signal2, ...);
SystemC
component_instance (signal1, signal2, ...);

Example
VHDL
`utpg : dtpg port map (clk,opx,opy);`
SystemC
`utpg(clk,opx,opy);`

Description: Reading from port or signal.

Sintax
VHDL
input_port_id input_signal_id
SystemC
input_port_id.read() input_signal_id.read()

Example
VHDL
`x:=xi;`
SystemC
`x=xi.read();`

Description: Writing on port or signal.

Sintax
VHDL
output_port_id <=value output_signal_id <=value
SystemC
output_port_id.write(value) output_signal_id.write(value)

Example
VHDL
`out<=y;`
SystemC
`out.write(y);`

Description: Constant declaration.

Sintax
VHDL
constant constant_id : constant_type;
SystemC
const constant_id constant_type;

Example
VHDL
`constant dimension :  integer := 8;`
SystemC
`const int dimension = 8;`

Description: Variable declaration.

Sintax
VHDL
variable variable_id : variable_type;
SystemC
variable_type variable_id;

Example
VHDL
`variable x :  integer;`
SystemC
`int x;`

Description: Integer with range.

Sintax
VHDL
integer range lower *to* higher; *integer range* higher *downto* lower;
SystemC
sc_int$< \log_2(max\{\|higher\|, \|lower\|\} + 1) >$ *sc_int*$< \log_2(max\{\|higher\|, \|lower\|\} + 1) >$

Example
VHDL
`variable x : integer range 0 to 15;`
SystemC
`sc_int` $< 4 >$ `x;`

Notes: If the value of $\log_2$(higher$-$lower$+1$) isn't a natural it's necessary to round it up to the nearest natural value. *VHDL2SC1.0* translates every VHDL integer with range into C++ `int` type.

Description: Array declaration.

Sintax
VHDL
variable variable : type(lower *to* higher); *variable* variable : type(higher *downto* lower);
SystemC
type variable[higher$-$lower$+1$]; type variable[higher$-$lower$+1$];

Example
VHDL
`variable str : string(1 to 20);` `variable bv : bit_vector(1 to 8);`
SystemC
`char str[20];` `sc_bv<8> bv;`

Notes: The sintax for some SystemC array type (`sc_lv`, `sc_bv`, `sc_int`, `sc_uint`, `sc_bigint` e `sc_biguint`) is different with respect to the standar C array type.

Description: New data type definition.

Sintax
VHDL
type type_id *is* type;
SystemC
typedef type type_id;

Example
VHDL
`type word is range -32768 to 32767;`
SystemC
`typedef sc_int`$< 16 >$ `word;`

Description: *Array* data type definition.

Sintax
VHDL
type type_id *is array*(inf to sup) of array_type; *type* type_id *is array*(infA to supA,infB to supB) of array_type; ...
SystemC
typedef array_type type_id[sup−inf+1]; *typedef* array_type type_id[supA−infA+1][supB−infB+1]; ...

Example
VHDL
`type array_of_char is array(1 to 4) of character;` `type matrix is array(1 to 4, 1 to 7) of integer;`
SystemC
`typedef char array_of_char[4];` `typedef int matrix[4][7];`

Notes: In the case of not defined interval (i.e. `natural range<>`), this construct will be not translated, but it's necessary to save informations about *type_id*, *array_type* and size, in order to obtain a correct translation when an array of this type will be instanced.

Description: Enumeration data type definition.

Sintax
VHDL
type type_id *is*(element1, element2, . . .);
SystemC
typedef enum type_id {element1, element2, . . . };

Example
VHDL
`type istr is (add, load, store);`
SystemC
`typedef enum istr {add, load, store};`

Description: Subtype definition.

Sintax
VHDL
subtype type_id *is* subtype;
SystemC
typedef subtype type_id;

Example
VHDL
`subtype word is bit_vector(31 downto 0);`
SystemC
`typedef sc_bv<32> word;`

Description: Variable assignment.

Sintax
VHDL
variable_id := value;
SystemC
id_variabile = value;

Example
VHDL
`x := 3;`
SystemC
`x = 3;`

Description: Type conversion.

Sintax
VHDL
type(variable_id)
SystemC
(type) variable_id

Example
VHDL
`x := unsigned(y);`
SystemC
`x = (sc_uint<16>)y;`

Description: Relational operators.

Sintax
VHDL
A = B A / = B A < B A <= B A > B A >= B
SystemC
A == B A ! = B A < B A <= B A > B A >= B

Example
VHDL
x = 3
SystemC
x == 3

Description: Logic operator.

Sintax
VHDL
not A A *or* B A *nor* B A *and* B A *nand* B A *xor* B A *xnor* B
SystemC
! A A || B ! (A || B) A && B ! (A && B) ((A || B) && (! (A && B))) ! ((A || B) && (! (A && B)))

Example
VHDL
`if (x nand (not(y))) then ...`
SystemC
`if (!(x && (!y))) ...`

Description: Bitwise operator.

Sintax
VHDL
not A A *or* B A *nor* B A *and* B A *nand* B A *xor* B A *xnor* B A *sll* valore A *srl* valore
SystemC
$\sim$ A A $\mid$ B $\sim$ (A $\mid$ B) A & B $\sim$ (A & B) (A $\wedge$ B) $\sim$ (A $\wedge$ B) A << valore A >> valore

Example
VHDL
`x := y xor z;`
SystemC
`x = y ∧ z;`

Notes: For shift operators *value* must be an integer value.

Description: *Slice* operator.

Sintax
VHDL
object_id(inf to sup)
SystemC
object_id.*range*(inf , sup)

Example
VHDL
`mar_in<=ir_out(6 to 15);`
SystemC
`mar_in.write(ir_out.read().range(6,15));`

Description: Arithmetic operators I.

Sintax
VHDL
+ A − A A + B A − B A ∗ B A / B A *rem* B
SystemC
+ A - A A + B A − B A ∗ B A / B A % B

Example
VHDL
`x := y+(x rem z);`
SystemC
`x = y+(x % z);`

Description: Arithmetic operators II.

Sintax
VHDL
A ** B *abs* A
SystemC
pow(A,B) *abs* B

Example
VHDL
`x := y**(abs(z));`
SystemC
`x = pow(y,(abs(z)));`

Notes: For these two operators the C++ library *math.h* is necessary.

Description: Arithmetic operators III.

Sintax
VHDL
A *mod* B
SystemC
int mod(*int* A,*int* B){   *if*(A∗B > 0) *return* (A%B);   *else*{   *if*((A%B)==0) *return* 0;   *else return* (B+(A%B));   } }

Example
VHDL
`x := y mod z;`
SystemC
`int mod(int A,int B){` `  if(A∗B > 0) return (A%B);` `  else{` `  if((A%B)==0) return 0;` `  else return (B+(A%B));` `  }` `}` `⋮` `x := mod(y,z);`

Description: If statement.

Sintax
VHDL
if (condition)*then*   instructions_sequence1 *else*   instructions_sequence2 *end if*;
SystemC
if(condizione){   instructions_sequence1 }*else*{   instructions_sequence2 }

Example
VHDL
`if(x>= 2) then` `  x := 1;` `else` `  x := 3;` `end if;`
SystemC
`if(x>= 2) {` `  x = 1;` `} else {` `  x = 3;` }

Description: Case statement I.

Sintax

VHDL

case(variable) *is*
    when value1 => instructions_sequence1
    when value2 => instructions_sequence2
    ...
    when other => default_instructions_sequence
end case;

SystemC

switch(variable) {
    case value1 : instructions_sequence1
                break;
    case value2 : instructions_sequence2
                break;
    ...
    default: default_instructions_sequence
                break;
}

Example

VHDL

```
case(x) is
     when 1 => y:=1;
     when 2 => y:=2;
     when other => y:=3
end case;
```

SystemC

```
switch(x) {
      case 1 :  y=1;
                break;
      case 2 :  y=2;
                break;
      default:  y=3;
                break;
}
```

Notes: VHDL is more flexible than SystemC with respect to the data type used as selection variable in this construct.

Description: Case statement II.

Sintax
VHDL
case(variable) *is*   *when* value1 =>   *null*;   *when* value2 | value3 | ... | valueN =>   instructions_sequence *end case*;
SystemC
switch(variable) {   *case* value1 :   *break*;   *case* value2 :   *case* value3 :   ...   *case* valueN :   instructions_sequence   *break*; }

Example
VHDL
`case(x) is` `  when 0 => null;` `  when 1 | 2 => y:=1;` `  when other => y:=3` `end case;`
SystemC
`switch(x) {` `  case 0 :  break;` `  case 1 :  y=1;` `  break;` `  case 2 :  y=1;` `  break;` `  default:  y=3;` `  break;` }

Description: *Loop* statement.

Sintax
VHDL
loop instructions_sequence *end loop*;
SystemC
while(*true*) { instructions_sequence }

Example
VHDL
`loop` `x:=x-1;` `if(x<0) then` `return;` `end if;` `end loop;`
SystemC
`while(true) {` `x=x-1;` `if(x<0) return;` }

Description: *While* statement.

Sintax
VHDL
while(condition) *loop* instructions_sequence *end loop*;
SystemC
while(condition) { instructions_sequence }

Example
VHDL
`while(x > 0) loop` `x:=x-1;` `end loop;`
SystemC
`while(x > 0) {` `x=x-1;` }

Description: *For* statement.

Sintax
VHDL
for(variable *in* value1 *to* value2) *loop* instructions_sequence *end loop*;
SystemC
for(*int* variable = value1; variable <= value2; variable ++) { instructions_sequence }

Example
VHDL
`for(x in 1 to 10) loop` `y:=y*x;` `end loop;`
SystemC
`for(int x=1; x<=10; x++) {` `y=y*x;` }

Notes: The range's value (*value1* e *value2*) must be a discrete value.
If the *for* iteration scheme used *downto* instead of *to* the <= operatot will becames >= and the iteration variable will be decremented.

Description: Statement used to modify the instructions control flow.

Sintax
VHDL
next; *next when* condition; *exit*; *exit when* condition;
SystemC
continue; *if* (condition) continue; *break*; *if* (condition) break;

Example
VHDL
`loop{` `  x:=x+1;` `  exit when(x=100);` `end loop;`
SystemC
`while(true){` `  x=x+1;` `  if( x == 100) break;` `}`

Description: Wait statements.

Sintax
VHDL
wait; *wait until* condition;
SystemC
wait(); *do*{   *wait*(); }*while*(! condition);

Example
VHDL
`wait until clock = '1';`
SystemC
`do{` `  wait();` `} while(!(clock==1));`

Notes: Every signal involved into the *condition* are added to the correspondig process's *sensitivity list*.

Description: Function and procedure declarations.

Sintax
VHDL
function function_id(par1:type1; par2:type2; ...)*return* return_type; *procedure* procedure_id(par1:*inout* type1; par2:*inout* type2; ...);
SystemC
return_type function_id(type1 par1, type2 par2, ...); *void* procedure_id(type1 *par1, type2 *par2, ...);

Example
VHDL
`procedure swap(x:inout integer; y:inout integer);`
SystemC
`void swap(int *x,int *y);`

Description: Function and procedure calls.

Sintax
VHDL
function_id(par1, par2, ...); procedure_id(par1, par2, ...);
SystemC
function_id(par1, par2, ...); procedure_id(&par1, &par2, ...);

Example
VHDL
`swap(x,y);`
SystemC
`swap(&x,&y);`

Description: Function definition.

Sintax
VHDL
function function_id(par1:type1; par2:type2; . . .)*return* return_type *is* function_declarations *begin* function_statements *end* function_id;
SystemC
return_type function_id(type1 par1, type2 par2, . . .){ function_declarations function_instructions }

Example
VHDL
`function successor(value:integer) return integer is` `begin` `value:=value+1;` `return value;` `end successor;`
SystemC
`int successor(int value){` `value=value+1;` `return value;` }

Description: Procedure definition.

Sintax
VHDL
procedure procedure_id(par1:*inout* type1; par2:*inout* type2; . . .)*is*   procedure_declarations *begin*   procedure_statements *end* procedure_id;
SystemC
void procedure_id(type1 par1, type2 par2, . . .){   procedure_declarations   procedure_instructions }

Example
VHDL
`procedure swap(x:inout integer; y:inout integer) is` `  variable tmp:integer;` `begin` `  tmp:=x;` `  x:=y;` `  y:=tmp;` `  end scambia;`
SystemC
`void swap(int *x, int *y) {` `  int tmp;` `  tmp=*x;` `  *x=*y;` `  *y=tmp;` }

Description: Package inclusion.

Sintax
VHDL
library library_name; *use* library_name.package_name.*all*;
SystemC
#include ”package_name.h” *using namespace* package_name;

Example
VHDL
`library logiblox;` `use logiblox.mvlutil.all;`
SystemC
`#include "mvlutil.h"` `using namespace mvlutil;`

Description: Package definition.

Sintax
VHDL
package package_name *is* package_declarations *end* package_name; *package body* package_name *is* package_definitions *end* package_name;
SystemC
//file: package_name.h *#IFNDEF* package_name_H *#DEFINE* package_name_H *namespace* package_name { package_declarations }; *#ENDIF* //file: package_name.cpp *#include* ”package_name.h” package_definitions;

Example
VHDL
`package mvlutil is` `  function stdbit2mvl(b:std_logic) return std_logic;` `end mvlutil;` `package body mvlutil is` `  function stdbit2mvl(b:std_logic) return std_logic is` `  ...` `  end stdbit2mvl;` `end mvlutil;`
SystemC
`//file:  mvlutil.h` `#IFNDEF mvlutil_H` `#DEFINE mvlutil_H` `namespace mvlutil { sc_logic stdbit2mvl(b:sc_logic); };` `#ENDIF` `//file:  mvlutil.cpp` `#include "mvlutil.h"` `sc_logic mvlutil ::  stdbit2mvl(sc_logic b) {...}`

Table 1 shows the correspondence between VHDL and SystemC data types.

VHDL	SystemC
boolean	bool
bit	sc_bit
std_logic	sc_logic
std_ulogic	sc_logic
character	char
integer	int
natural	int
positive	int
signed	sc_int
unsigned	sc_uint
bit_vector	sc_bv
std_logic_vector	sc_lv
std_ulogic_vector	sc_lv

Table 1: Correspondence between VHDL and SystemC data types.

2.3 The file Rules.dat

Rules.dat is a simple text file, included in the directory VHDL2SC1.0/src/aire/iir/IIRSystemC/Rules, that contains some informations used by VHDL2SC1.0 to obtain a semantically correct SystemC translation and to indicate the correct matching between some VHDL standard functions and the corresponding SystemC ones.
This file is independent from the IIR hierarchy, thus two important advantages are reached:

- **Flexibility:** It's possible to extend the functionalities of the translator adding new informations into Rules.dat without modifying and recompiling the source code. In this way the user of VHDL2SC can extend and customize the tool.
- **Clarity:** The file's structure allows to easily identify the correspondence between VHDL element and SystemC ones.

The informations included in Rules.dat are divided in six different parts:

1. **Dictionary:** Mapping between VHDL and SystemC for language's operators and types.
2. **Need cast:** The SystemC types that need an explicit cast for some arithmetic operations are listed.
3. **SystemC array type:** The SystemC array types with different sintax with respect to standard C array types are listed.
4. **Object initialization:** The SystemC types that need an explicit initialization to preserve the VHDL corresponding semantics are listed with the relative initialization value.
5. **Default generic:** The default value used to initialize the VHDL generic constants are listed. Because the generic constants are translated using C++ template it's necessary the presence of a default value even if the original VHDL code doesn't include it.
6. **Function mapping:** Information about VHDL procedures and functions and their SystemC implementation are presented. This section of Rules.dat helps to translate VHDL functions and procedures defined in standard libraries as ieee, synopsy, mentor etc... This is not for user defined packages.
   The information included are the followings:
   - SystemC function implementation file;
   - VHDL package name;
   - function name;
   - return type (”void” for no return type);
   - parameters type (”void” for no parameters).

The following is the actual versione of Rules.dat:

```
#This file is a collection of rules to support VHDL to SystemC
#translation. There are six kind of rules: dictionary, need cast,
#SystemC array type, object initialization, default generic and
#functions mapping.

#Dictionary: mapping between VHDL_word and SystemC_word
#DICTIONARY
bit sc_bit
std_logic sc_logic
std_ulogic sc_logic
std_logic_vector sc_lv
std_ulogic_vector sc_lv
boolean bool
integer int
natural int
positive int
character char
unsigned sc_uint
signed sc_int
bit_vector sc_bv
"abs" abs
"-" -
"+" +
"*" *
"**" pow
"&"  IIRSystemCFunctionCall
"/" /
">" >
"<" <
"<=" <=
"<=" <=
"=" ==
"/=" !=
"rem" %
"mod" mod
"and" &
"nand" IIRSystemCFunctionCall
"or" |
"nor" nor
"xor" ^
"xnor" IIRSystemCFunctionCall
"not" ~
"sll" <<
"srl" >>
"sla" sla
```

```
"sra" sra
"rol" rol
"ror" ror
halt halt_sc

#SystenmC data types which need cast
#NEED_CAST
sc_logic
sc_bv
sc_lv
sc_uint


#Array type wich need a different sintax, i.e. "sc_lv<3> var"
#insted of "char a[3]"
#SC_ARRAY_TYPE
sc_lv
sc_bv
sc_int
sc_uint
sc_bigint
sc_biguint


#A VHDL std_logic_vector is translated into a SystemC sc_lv. If
#no initialization to std_logic_vector is given, the default value
#is composed of 'U'. For sc_lv the default value is composed of
#'0'. Next information help to maintain in SystemC the same
#default value as in VHDL.
#OBJECT_INITIALIZATION
sc_lv x


#This is referred to translation of VHDL generic in C++ template.
#The need for a default generic value is due to C++ limitation:
#when in a template a default value is set for a variable , you
#have to do the same for all the other parameters of the template.
#Unlike C++, VHDL allows you to specify only certain default value.
#DEFAULT_GENERIC
bool 0
int 0

#Information about functions and their implementation. This section
#of database rules helps to translate VHDL functions and procedures
#defined in  ieee, synopsy, mentor etc. library.
```

```
#The information are:
#1)function implementation file
#2)package name
#3)function name
#4)return type ("void" for no return type)
#5)parameters type ("void" for no parameters)

#FUNCTION_MAPPING
mod.dat
standard
"mod"
*
* *


#FUNCTION_MAPPING
shr.dat
std_logic_arith
shr
unsigned
unsigned unsigned

#FUNCTION_MAPPING
conv_unsigned.dat
std_logic_arith
conv_unsigned
unsigned
integer integer

#FUNCTION_MAPPING
write.dat
textio
write
void
line string side width
```

3 VHDL2SC State of Art and Actual Limitations

VHDL2SC1.0 is able to translate automatically VHDL models with some limitations. In the next paragraphs the current state of the translator and its limitations will be explained thus users can accordingly modify rightly the VHDL source code in order to obtain a correct SystemC translation.

The VHDL constructs actually managed are the followings:

- entity declarations;

- architecture declarations;
- port, signal and variable declarations;
- generic constants;
- types and subtypes;
- standard libraries (ieee, synopsys,... . These libraries aren't translated but they are managed by using the file Rules.dat);
- user packages (they must be into a library named "work");
- processes;
- concurrent statements;
- sequential statements;
- functions and procedures;
- component instantiations;
- event and length attribute;
- logic and arithmetic operator.

A note is necessary about the translation of concurrent statements: because this kind of VHDL construct semantically has the same meaning of a corresponding sequential statement included into a process, first they are translated into VHDL processes and then theyare translated in SystemC code.

The actual VHDL2SC1.0 limitations are the followings:

- Every VHDL design file must contain just one single entity with one single architecture.
- User's packages must belong to a library named work, while standard packages can stay in the usual libraries (ieee, std, ...).
- The following VHDL constructs are not translated:
  - blocks;
  - labels;
  - attributes except length and event;
  - resolved types;
  - floating point types;
  - physical types;
  - strings;

- the *wait for* construct;
- the concatenation operator;
- the *case* construct when the selection variable is obtained applying the slice operator to a vector;
- the *others* construct.

As reported in the introduction, VHDL2SC has been developed considering the SystemC version 1.0.2. We tried to compile the translated code using the version 1.2 and 2.0b. In the first case the compilation worked showing some warning, in the other case, instead, the compilation didn't work because of the new caractheristics of SystemC 2.0.
For example, consider `X` as a port of type `sc_bit`: the following statement is correct in SystemC 1.0.2 but it isn't in SystemC 2.0:

```
X.write(0);
```

in SystemC 2.0 you have to put an explicit cast as it follows:

```
X.write((sc_bit)0);
```

4 VHDL2SC1.0 Known Bugs

Here there is the list of VHDL2SC1.0 known bugs. We are working to solve these problems as soon as possible. If you find some others bugs, please let us known at **pravadelli@sci.univr.it**.
For every bug it is reported an example containing the VHDL original code, the SystemC translation obtained with VHDL2SC1.0 and the noticed problem.

- Let `x` and `y` two VHDL signals of a vector type (i.e. std_logic_vector, bit_vector, ...) and cosider the following VHDL statement:

  ```
  x[i] <= y[j];
  ```

  VHDL2SC translate it into:

  ```
  x.read()[i].write(y.read()[j]);
  ```

  Obviously the C++ compiler doesn't work becasuse the `write` methods is used on the element `i` of `x` that isn't a signal.

- Constants defined into functions or procedures, inclused in packages, are not translated. For example consider the following VHDL code:

```
PACKAGE a_package IS
FUNCTION foo (arg : integer) RETURN std_ulogic_vector;
END a_package;
PACKAGE BODY a_package IS
  FUNCTION foo (arg : integer) RETURN std_ulogic_vector IS
    VARIABLE temp   : INTEGER;
    CONSTANT x : std_ulogic := '1';
    VARIABLE result : std_ulogic_vector(size-1 DOWNTO 0);
  BEGIN
    ...
    RETURN result;
  END;
END a_package;
```

In the SystemC translation there will be not the declaration of constant `x`.

- VHDL2SC translates the VHDL `range` operator into the corresponding SystemC `range` operator. This creates a problem when the `range` operator is used into a function call. Consider the following VHDL function call where the `foo`'s formal parameter and the variables `x` and `y` are, for example, of type `standard_logic_vector`:

  ```
  y = foo(x.range(4,0));
  ```

  The SystemC translation will be:

  ```
  y = foo(x.range(4,0));
  ```

  While the VHDL code is correct, unfortunatly this SystemC code doesn't work because the range operator returns an object of type `sc_range` instead of `sc_lv`. So you need to put an explicit cast to `sc_lv` before the `foo`'s actual parameter.

- A VHDL process containing a `wait` statement is translated into a SystemC `SC_THREAD`. Consider the next VHDL code included in a file named foo.vhdl:

  ```
  process(clock)
  begin
    ...
    ...
  wait;
  end process;
  ```

  The SystemC translation will be:

  ```
  // foo.h
  #include "systemc.h"
  SC_MODULE(foo) {
  ```

```
  ...
  ...
  SC_CTOR(foo) {
    SC_THREAD(process_1);
    sensitive << clock;
  }
  void process_1();
}

//foo.cc
#include foo.h
void foo::process_1() {
  ...
  ...
  wait();
}
```

When the execution restarts after the wait statement the SC_THREAD terminates forever, instead the VHDL process could be reactivated when the clock signal changes. To avoid this problem the SystemC code in foo.cc should be like the following:

```
//foo.cc
#include foo.h
void foo::process_1() {
  while (1) {
    ...
    ...
    wait();
  }
}
```

5 VHDL2SC1.0 Installation, configuration and Usage

5.1 Installation and Configuration

VHDL2SC1.0 works under Linux. Download VHDL2SC1.0 and untar the file into the destination directory. Then edit the configure script included into VHDL2SC1.0 directory and modify it as it follows:

- Modify the SYSTEMCROOT variable in order to point to the directory including your version of SystemC.
- Modify the SAVANTROOT variable in order to point to the directory including VHDL2SC1.0.

Before using VHDL2SC1.0 run the configure script: `source configure`.

5.2 Usage

Use of VHDL2SC1.0 is very simple. After the installation and the configuration the tool could be launched by the following command line:

```
scram -valid_arguments file1 [...  fileN]
```

where `valid_arguments` are at least one between:

`publish-vhdl`	to VHDL output in stdout and work._savant_lib
`publish-sc`	to SystemC output in the directory work._savant_lib
`design-library-name` *library*	to analyze the specified package included in *library*
`warning`	to print translation warning (only with -publish-sc)

For more informations about VHDL2SC1.0 installation and configuration see README file included in the VHDL2SC1.0 installation package.