## 1.1 Csl_language command summary

### *1.1.1 Constants*

CSL allows for definition of constant integers just like in the C++ programming language; The syntax for declaring a const int is also borrowed from the aforementioned language and is shown in the example below:

```
    const int const_name = numeric_expression;
```

Note for example above: **bold** text represents language reserved syntax, *italics* are user defined variables and blue text is a short BNF representation of CSL commands/declarations.

In the const int definition, `numeric_expression` can be anything from a plain number to an expression (unary, binary, ternary or a function call) that results in a number.

#### *1.1.1.1 Const int usage and rules*

Constants can be declared in the global scope (outside the scope of any class) or inside certain CSL classes as can be seen in Table 1.1:

**TABLE 1.1** CSL const int declaration in other CSL classes

| CSL class | Accepts const int declaration |
|---|---|
| CSL Unit | YES |
| CSL Testbench | - |
| CSL Vector | - |
| CSL State Data | - |
| CSL Register | - |
| CSL Register File | - |
| CSL Fifo | - |
| CSL Memory Map | - |
| CSL Memory Map Page | - |
| CSL Isa Element | - |
| CSL Isa Field | - |
| CSL Field | - |

#### *1.1.1.1.1 Places where bitrange methods are used*

| | cnst self br | cnst other br | gbl inst self br | cnst inst other br | field | signal | sig_gr | port | ifc | unit | rf | fifo |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **set_msb** | X | | | | | | | | | | | |
| **get_msb** | X | X | X | X | | X | X | X | X | X | X | X |
| **set_lsb** | X | | | | | | | | | | | |
| **get_lsb** | X | X | X | X | | X | X | X | X | X | X | X |
| **get_width** | X | X | X | X | | X | X | X | X | X | X | X |
| **set_offset** | | X | | X | X | | | | | | | |
| **get_offset** | | X | | X | X | | | | | | | |

Memory elements
```
   csl_rd_interface ifc_name;
```
Prefix
```
   set_prefix(string “prefix”);
   string get_prefix();
   string str = string;
   namespace.any_object;
   csl_define
```
Compile time constants
```
   if-else, while,do-while, for, foreach,switch,case
```
String Operations
Regular Expressions
```
   ir.instr.field.create_signal(signal_name);
   ir.inst_fmt_name.field_name.connect(signal_name);
```
Pipeline Positioning {
```
   signal_name.set_pipestage(numeric_expression);
   numeric_expression signal_name.get_pipestage();

   register_name.set_pipestage(numeric_expression);
   numeric_expression register_name.get_pipestage();

   list l = {p, q};
   csl_for(int i = 0; i <= 7; i++) {}
   exp(base, constant_numeric_expression);
   foreach a (pc, id, rf) {}
   log(numeric_expression);
```

## Events

```
csl_event event_object_name;
event_object_name.add_equation(boolean_expresion);
rf.event(wr_en& wr_addr);
```

## CSL get methods

### Single dimension bitrange commands

```
bitrange_name.get_upper_index(numeric_expression);

bitrange_name.get_lower_index(numeric_expression);
int bitrange_name.get_width();
```

**Single dimension bitrange commands**

### *1.1.1.1.2 Multidimension bitrange commands*

The following commands are specific to multidimension bitranges
Multidimensional bitranges can only be associated with signals and ports. The declaration syntax for a multidimensional bitrange is shown below:

```
    csl_multi_dim_bitrange bitrange_name(constructor_parameters);
```

**Bold** text represents language reserved syntax, *italics* are user defined variables.

The constructor parameters can be a single numeric expression for setting the number of dimensions or an identifier that can be a numeric value (e.g. const int) or the name of another multidimensional bitrange object (copy constructor - the properties of the copied object are replicated under a new object with the specified name):

```
    constructor_parameters ::= numeric_expression
                              |multidimensional_bitrange_object_name
```

A short example is provided below

```
    csl_multi_dim_bitrange mbr1(3);
    csl_multi_dim_bitrange mbr2(mbr1);
    csl_unit u{
      csl_signal s1(mbr1), s2(mbr2);
      u(){
      s1.set_dim_range(0,0,3);
      s1.set_dim_range(1,1,3);
      s1.set_dim_range(2,4,6);
    }
};
```

Note that in the example above, the widths for each dimension have to be manually set using the appropriate commands.

**Multidimension bitrange commands**

```
    object_multi_dim.set_dim_bitrange(num_expr,br_obj_name);
    object_multi_dim.set_dim_range(num_expr,num_expr,num_expr);
    csl_multi_dim_bitrange obj_name(num_expr);
```

**if-else, while,do-while, for, foreach,switch,case**

---

```
    object_multi_dim.set_dim_bitrange(num_expr,br_obj_name);
    object_multi_dim.set_dim_range(num_expr,num_expr,num_expr);
    object_multi_dim.set_dim_offset(num_expr,num_expr);
    object_multi_dim.get_dim_upper(num_expr);
    object_multi_dim.set_dim_offset(num_expr,num_expr);
    object_multi_dim.get_dim_upper(num_expr);
    object_multi_dim.set_dim_offset(num_expr,num_expr);
    object_multi_dim.get_dim_offset(num_expr);
    int field_name.get_width();
    int field_name.get_upper_index();
    int field_name.get_lower_index();
    [field_name.]get_offset();
    field_name.get_enum();
    field_name.get_value();
    object_multi_dim.get_dim_offset(num_expr);
```

---

**DESCRIPTION :**

//

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

//

CSL CODE

```
    //
```

VERILOG CODE

```
    //
```

*ir*.*instr*.*field*.**create_signal(***signal_name***);**

## DESCRIPTION :

Creates a signal a field width connected to the ir[field.get_up(): field.get_low()]
Creates a signal with the field that is connected to ir[field.lower:field.upper];

[ ***CSL Language Commands Summary*** ]

## EXAMPLE :

//

CSL CODE

Verilog code

```
ir.inst_fmt_name.field_name.connect(signal_name);
```

**DESCRIPTION :**

Creates a signal that is connected to the field in ir.inst_fmt_name.field_name
Will search for *signal_name* signal and then will connect with ir[field.upper:field.lowers];

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

//

CSL CODE

```
    csl_register ir;
    ir.add_isa(isa_name);
```

ir.inst_fmt_name.field_name**.connect(***signal_name***);**

`signal_name.`**`set_pipestage`**`(numeric_expression);`

**DESCRIPTION :**

This statement assigns the pipe stage number `numeric_expression` to the signal `signal_name.`

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Assigns a pipe stage number to the signal named *sig.*

CSL CODE

```
csl_unit u{
csl_signal sig;
u(){
sig.set_pipestage(2);
}
};
```

VERILOG CODE

```
//
```

`numeric_expression signal_name.get_pipestage();`

**DESCRIPTION :**

Returns the pipe stage number of a signal named `signal_name;`

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Sets the pipe stage for the signal sig2 using *get_pipestege()* method.

CSL CODE

```
csl_unit u{
csl_signal sig1,sig2;
u(){
sig1.set_pipestage(4);
sig2.set_pipestage(sig1.get_pipestage());
}
};
```

VERILOG CODE

```
//
```

`register_name.`**`set_pipestage`**`(numeric_expression);`

**DESCRIPTION :**

This statement assigns the pipe stage number `numeric_expression` to the register `register_name.`

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Sets the pipe stage number for the register *reg1.*

CSL CODE

```
csl_register reg1{
reg1(){}
};
csl_unit u{
reg1 reg1;
u(){
reg1.set_pipestage(4);
}
};
```

VERILOG CODE

```
//
```

```
numeric_expression register_name.get_pipestage();
```

**DESCRIPTION :**

Returns the pipestage number of a register `register_name`.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Sets the pipe stage number for the register *reg2* using *get_pipestage()* method.

CSL CODE

```
csl_register r{
r(){}
};
csl_unit u{
r reg1;
r reg2;
u(){
reg1.set_pipestage(4);
reg2.set_pipestage(reg1.get_pipestage());
}
};
```

VERILOG CODE

```
//
```

```
list l = {p, q};
```

**DESCRIPTION :**

n/a

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

//example description

CSL CODE

```
    csl_unit u{
    csl_signal s1,s2;
    u(){
    list l={s1, s2};
    }
    };
```

VERILOG CODE

```
    //verilog code goes here
```

```
csl_for(int i = 0; i <= 7; i++) {}
```

**DESCRIPTION :**

n/a

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

//example description

CSL CODE

```
   //
```

VERILOG CODE

```
   //verilog code goes here
```

```
exp(base, constant_numeric_expression);
```

**DESCRIPTION :**

Returns base " constant_numeric_expression where the ^ character Is the exponent operator.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

//example description

CSL CODE

```
    csl_unit u{
    csl_signal sig(exp(2,2));
    u(){}
    };
```

VERILOG CODE

```
    //verilog code goes here
```

```
foreach a (pc, id, rf) {}
```

**DESCRIPTION :**

n/a

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

//example description

CSL CODE

```
   //
```

VERILOG CODE

```
   //verilog code goes here
```

In the following examples n-bit_expression reduces to a constant number and can contain macros and defines.

```
log(numeric_expression);
```

**DESCRIPTION :**

Takes the log base 2 of the expression.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

//example description

CSL CODE

```
    csl_unit u{
    csl_signal sig1(log(8));
    u(){}
    };
```

VERILOG CODE

```
    //verilog code goes here
```

### *1.1.2 CSL event*

**csl_event** *event_object_name;*

**DESCRIPTION :**

Event declaration.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

CSL CODE

```
   csl_event ev;
   csl_unit a{
   csl_signal  s0, s1;
    a(){ }};
```

```
event_object_name.add_equation(boolean_expresion);
```

**DESCRIPTION :**

Add a equation(definition) to a event using boolean equation.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Define a event that detects if signal a of the unit_name unit is set.

CSL CODE

```
   csl_event event_name;
   csl_signal a1, a0;
    event_name.add_equation(a1 | a0);
```

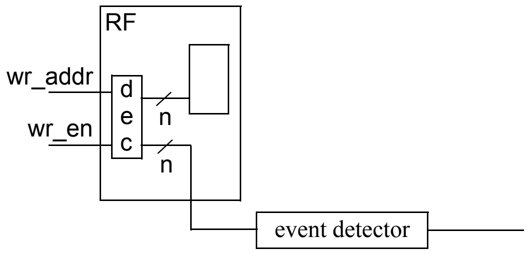
```
rf.event(wr_en& wr_addr);
```

**DESCRIPTION :**

If a event detector is added to detect a unit to a register file then the event detection is added outside of the rf and the rf's write addr decoder outputs are connected to the event detector.

[ ***CSL Language Commands Summary*** ]

**FIGURE 1.1**



**EXAMPLE :**

**csl_rd_interface** *ifc_name;*
The following ports are created for a register file object:

```
port: output - <ifc_name>_rd_data
```

**DESCRIPTION :**
Creates the *<ifc_name>_rd_data* port for a register file when when **csl_rd_interface** csl command is used

```
port: input - <ifc_name>_rd_addr
```

**DESCRIPTION :**
Creates the *<ifc_name>_rd_addr* port for a register file when **csl_rd_interface** csl command is used

```
port: input - <ifc_name>_rd_en
```

**DESCRIPTION :**
Creates the *<ifc_name>_rd_en* port for a register file when **csl_rd_interface** csl command is used

For a fifo:

**DESCRIPTION :**
This adds a user defined read interface to the register file/fifo. The user defined interface replaces the default ports for read operation. More than one read interfaces can be added to a register file. A rd_interface contains a read address port, a read data port a read enable port and optionally a valid port. This will be generated in the verilog code.

**EXAMPLE :**
For example you want to write data in 2 fields for the reg file in same time and read from 1

CSL CODE

```
csl_register_file regf1{
  csl_wr_interface ch1;
  csl_wr_interface ch2;
  csl_rd_interface ch3;
  regf1() {
    set_width(8);
    set_depth(32);
  }
};
```

VERILOG CODE

```
module regf1(clk, rst_, ch1_wr_en, ch1_wr_data, ch1_wr_addr,
ch2_wr_en, ch2_wr_data, ch2_wr_addr, ch3_rd_en, ch3_rd_data,
ch3_rd_addr);
  parameter ADDR_WIDTH = 5;
  parameter DEPTH = (1<<ADDR_WIDTH)-1;
  parameter DATA_WIDTH = 8;
  input clk, rst, ch1_wr_en, ch2_wr_en, ch3_rd_en;
  input [ADDR_WIDTH-1:0] ch1_wr_addr, ch2_wr_addr, ch3_rd_addr;
  input [DATA_WIDTH-1:0] ch1_wr_data, ch2_wr_data;
  output [DATA_WIDTH-1:0] ch3_rd_data;
  reg [DATA_WIDTH-1:0] ch3_rd_data;
  reg [DATA_WIDTH-1:0] mem [0:DEPTH-1];
  integer i;
  always @(posedge clk or negedge rst_) begin
    if (~rst_) begin
      for(i = 0; i < DEPTH; i=i+1)
         mem[i] = 0;
    end
    else begin
      if (ch3_rd_en) begin
        ch3_rd_data <= mem[ch3_rd_addr];
      end
      if (ch1_wr_en) begin
        mem[ch1_wr_addr] = ch1_wr_data;
      end
      if (ch2_wr_en) begin
        //here maybe we should have a check for wr_addr
        if (ch1_wr_addr == ch2_wr_addr) begin
          $display("Multiple writes in same memory lacation at same
time");
        end
        else begin
          mem[ch2_wr_addr] = ch2_wr_data;
        end
      end
    end
  end
endmodule
```

//

**FIGURE 1.2**

CSL CODE:

```
//
```

VERILOG CODE:

//Register file with 1 input ports and 2 output

```
module
RF_1X2(wr_data,rd_data_1,rd_data_2,wr_addr,rd_addr_1,rd_addr_2,wr_en,r
d_en_1,rd_en_2,clk,reset);
//list of signals
reg [15:0] register_file [3:0];
input clk,reset,rd_en_1,rd_en_2,wr_en;
input [1:0] wr_addr,rd_addr_1,rd_addr_2;
input [15:0] wr_data;
output [15:0] rd_data_1,rd_data_2;
reg [15:0] rd_data_1,rd_data_2;
integer i;

//register file logic
always@(posedge clk or negedge reset or rd_en_1 or rd_en_2)
begin

if(!reset)
begin

for(i=0;i<4;i=i+1)
begin
register_file[i]=16'b0;
end

end

else
begin

if(wr_en)
begin
register_file[wr_addr]=wr_data;
end

end
```

//read logic is not clocked

//read logic for port 1

```
if(!rd_en_1)
begin
    rd_data_1=16'b z;
    end

else
begin
rd_data_1= register_file[rd_addr_1];
end

//read logic for port 2
if(!rd_en_2)
begin
    rd_data_1=16'b z;
end

else
begin
    rd_data_2= register_file[rd_addr_2];
end

end

endmodule
```

**set_prefix(string** “prefix”**);**

**DESCRIPTION :**

Add prefix “prefix“ to the csl object. Is aplicable to all container objects/classes (units, interfaces, ports, signals, groups, lists, etc)

**EXAMPLE :**

`string` **`get_prefix();`**

**DESCRIPTION :**

Return the prefix “prefix“ from the csl object. Is aplicable to all container objects/classes (units, interfaces, ports, signals, groups, lists, etc)

**EXAMPLE :**

```
string str = string;
```

**DESCRIPTION :**

n/a

**EXAMPLE :**

//example description

CSL CODE

```
   //
```

VERILOG CODE

```
   //verilog code goes here
```

```
namespace.any_object;
```

**DESCRIPTION :**

//you can prefix any object name in a declaration or the usage of a any object

**EXAMPLE :**

//example description

CSL CODE

```
   //
```

VERILOG CODE

```
   //verilog code goes here
```

`csl_define`

**DESCRIPTION :**

n/a

**EXAMPLE :**

//example description

CSL CODE

```
    //
```

VERILOG CODE

//verilog code goes here

*bitrange_name*.**get_upper_index(***numeric_expression***);**

**DESCRIPTION :**

//

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

small description of the example.

CSL CODE

```
csl_bitrange br1(4);
csl_bitrange br2(4);
br1.set_upper_index(3);
br1.set_lower_index(0);
br2.set_upper_index(br1.get_upper_index());
br2.set_lower_index(br1.get_lower_index());
```

VERILOG CODE

*bitrange_name*.**get_lower_index(***numeric_expression***);**

**DESCRIPTION :**

//

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

small description of the example.

CSL CODE

```
csl_bitrange br1(4);
csl_bitrange br2(4);
br1.set_upper_index(3);
br1.set_lower_index(0);
br2.set_upper_index(br1.get_upper_index());
br2.set_lower_index(br1.get_lower_index());
```

VERILOG CODE

**int** *bitrange_name*.**get_width();**

## DESCRIPTION :

Returns the width of a bitrange object.

[ ***CSL Language Commands Summary*** ]

## EXAMPLE :

Sets the width of the bitrange *br2* using the width of bitrange *br1*.

CSL CODE

```
csl_bitrange br1(4);
csl_bitrange br2(br1.get_width());
csl_unit u{
  csl_port p1(input, br1), p2(output, br2);
  u(){}
};
```

VERILOG CODE

```
module u(p1,
         p2);
  input [3:0] p1;
  output [3:0] p2;
endmodule
```

*bitrange_name*.**get_offset();**

**DESCRIPTION :**

Returns the offset of the bitrange.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Sets the offset for *br2* using the offset of *br1*.

CSL CODE

```
csl_bitrange br1(4);
csl_bitrange br2(4);
br1.set_offset(8)
br2.set_offset(br1.get_offset());
csl_unit u{
  csl_port p1(input, br1), p2(output, br2);
  u(){}
};
```

VERILOG CODE

*bitrange_name*.**get_lower_index**();

**DESCRIPTION :**

Returns the lower index of the bitrange.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Sets the range for bitrange *br2* using lower index and upper index from bitrange *br1*.

CSL CODE

```
csl_bitrange br1(0,7);
csl_bitrange br2(br1.get_lower_index(), br1.get_upper_index());
csl_unit u{
  csl_port p1(input, br1), p2(output, br2);
  u(){}
};
```

VERILOG CODE

*bitrange_name*.**get_upper_index**();

**DESCRIPTION :**

Returns the upper index of the bitrange.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Sets the range for bitrange *br2* using lower index and upper index from bitrange *br1*.

CSL CODE

```
csl_bitrange br1(0,7);
csl_bitrange br2(br1.get_lower_index(), br1.get_upper_index());
csl_unit u{
  csl_port p1(input, br1), p2(output, br2);
  u(){}
};
```

VERILOG CODE

```
bitrange_name.get_dim_width(numeric_expr);
```

**DESCRIPTION :**

Returns the numeric expresion that represents the width of the specified dimension of *object_multi_dim* :

param: dimension index (dimension index start from 0 and go up to num_of_dims-1);

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

In this example it is created two multidimensional bitrange *mbr1* and *mbr2* with 3 dimensions. We used the *get_dim_width()* method to set the width for the *mbr2* dimensions.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
mbr1.set_dim_width(0,2);
mbr1.set_dim_width(1,4);
mbr1.set_dim_width(2,8);
csl_multi_dim_bitrange mbr2(3);
mbr2.set_dim_width(0,mbr1.get_dim_width());
mbr2.set_dim_width(1,mbr1.get_dim_width());
mbr2.set_dim_width(2,mbr1.get_dim_width());
csl_unit u{
  csl_signal s1(mbr1), s2(mbr2);
  u(){}
};
```

VERILOG CODE

`object_multi_dim.get_dim_bitrange(num_expr);`

**DESCRIPTION :**

Returns : the simple dimension bitrange associated with the specified dimension of *object_multi_dim* :

param1: dimension index (dimension index start from 0 and go up to num_of_dims-1);

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

In this example it is created two multidimensional bitrange *mbr1* and *mbr2* with 3 dimensions. We used the *get_dim_bitrange()* method to set the width for the *mbr2* dimensions like *mbr1* dimensions.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
mbr1.set_dim_width(0,2);
mbr1.set_dim_width(1,4);
mbr1.set_dim_width(2,8);
csl_multi_dim_bitrange mbr2(3);
mbr2.set_dim_bitrange(0, mbr1.get_dim_bitrange(0));
mbr2.set_dim_bitrange(1, mbr1.get_dim_bitrange(1));
mbr2.set_dim_bitrange(2, mbr1.get_dim_bitrange(2));
csl_unit u{
  csl_signal s1(mbr1), s2(mbr2);
  u(){}
};
```

VERILOG CODE

`object_multi_dim.get_dim_lower(num_expr);`

**DESCRIPTION :**

Returns : numeric expresion that represents the lower index of the specified dimension of *object_multi_dim* :

param1: dimension index (dimension index start from 0 and go up to num_of_dims-1);

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

In this example it is created two multidimensional bitrange *mbr1* and *mbr2* with 3 dimensions. We used the *get_dim_lower() and get_dim_upper()* methods to set the range for the *mbr2* dimensions like *mbr1* dimensions.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
mbr1.set_dim_range(0,0,1);
mbr1.set_dim_range(1,0,3);
mbr1.set_dim_range(2,0,7);
csl_multi_dim_bitrange mbr2(3);
mbr2.set_dim_range(0, mbr1.get_dim_lower(0), mbr1.get_dim_upper(0));
mbr2.set_dim_range(1, mbr1.get_dim_lower(1), mbr1.get_dim_upper(1));
mbr2.set_dim_range(2, mbr1.get_dim_lower(2), mbr1.get_dim_upper(2));
csl_unit u{
  csl_signal s1(mbr1), s2(mbr2);
  u(){}
};
```

VERILOG CODE

*object_multi_dim*.**get_dim_upper**(num_expr);

**DESCRIPTION :**

Returns : numeric expresion that represents the upper index of the specified dimension of *object_multi_dim* :

param1: dimension index (dimension index start from 0 and go up to num_of_dims-1);

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

In this example it is created two multidimensional bitrange *mbr1* and *mbr2* with 3 dimensions. We used the *get_dim_lower() and get_dim_upper()* methods to set the range for the *mbr2* dimensions like *mbr1* dimensions.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
mbr1.set_dim_range(0,0,1);
mbr1.set_dim_range(1,0,3);
mbr1.set_dim_range(2,0,7);
csl_multi_dim_bitrange mbr2(3);
mbr2.set_dim_range(0, mbr1.get_dim_lower(0), mbr1.get_dim_upper(0));
mbr2.set_dim_range(1, mbr1.get_dim_lower(1), mbr1.get_dim_upper(1));
mbr2.set_dim_range(2, mbr1.get_dim_lower(2), mbr1.get_dim_upper(2));
csl_unit u{
  csl_signal s1(mbr1), s2(mbr2);
  u(){}
};
```

VERILOG CODE

*object_multi_dim*.**get_dim_offset**(num_expr);

**DESCRIPTION :**

Returns : numeric expresion that represents the offset of thespecified dimension of *object_multi_dim* :

param1: dimension index (dimension index start from 0 and go up to num_of_dims-1);

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

In this example it is created two multidimensional bitrange *mbr1* and *mbr2* with 3 dimensions. We used the *get_dim_offset()* methods to make the same changes like *mbr1* for *mbr2* dimensions.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
mbr1.set_dim_width(0,2);
mbr1.set_dim_width(1,4);
mbr1.set_dim_width(2,8);

mbr1.set_dim_offset(0,4);
mbr1.set_dim_offset(1,8);
mbr1.set_dim_offset(2,2);

csl_multi_dim_bitrange mbr2(3);
mbr2.set_dim_width(0,2);
mbr2.set_dim_width(1,4);
mbr2.set_dim_width(2,8);

mbr2.set_dim_offset(0,mbr1.get_dim_offset(0));
mbr2.set_dim_offset(1,mbr1.get_dim_offset(1));
mbr2.set_dim_offset(2,mbr1.get_dim_offset(2));

csl_unit u{
  csl_signal s1(mbr1);
  u(){}
};
```

VERILOG CODE

`[field_name.]`**`get_offset()`**`;`

**DESCRIPTION :**

Returns the offset of the field.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

set the offset for the field *fd2* using the offset from *fd1.*

CSL CODE

```
csl_field fd1{
  fd1(){
fd1.set_offset(8);
  }};
csl_field fd2{
  fd1 fd1;
  fd2(){
set_offset(fd1.get_offset());
  }};
```

```
int field_name.get_lower_index();
```

**DESCRIPTION :**

Returns the lower index of the field.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Creates a field *fd2* with the *fd1* range , using *get_lower()* and *get_upper_index()* methods .

CSL CODE

```
    csl_field fd1(4);
    csl_field fd2(fd1.get_lower_index(), fd1.get_upper_index());
```

```
int field_name.get_upper_index();
```

**DESCRIPTION :**

Returns the upper index of the field.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Creates a field *fd2* with the *fd1* range , using *get_lower()* and *get_upper_index()* methods .

CSL CODE

```
   csl_field fd1(4);
   csl_field fd2(fd1.get_lower_index(), fd1.get_upper_index());
```

```
int field_name.get_width();
```

**DESCRIPTION :**

Returns the width of the field.

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

Sets the width for the field *fd2* , using the *fd1* width.

CSL CODE

```
   csl_field fd1(4);
csl_field fd2(fd1.get_width());
```

`field_name.get_enum();`

**DESCRIPTION :**

Returns the associated enum item of the field object;

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

In this example it is associated an enum item to the field *fd0* using *get_enum()* method.

CSL CODE

```
csl_enum enum1{fd1, fd2, fd3};
csl_field fd{
  fd(){
    set_enum(enum1);
  }};
csl_field fd0{
fd fd;
  fd0(){
set_enum(fd.get_enum());
  }};
```

```
field_name.get_value();
```

**DESCRIPTION :**

Returns the associated numeric value of the field object;

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

In this example it is associated a value to the field fd2 using the *get_value()* method.

CSL CODE

```
csl_field fd1{
  fd1(){
set_value(16);
  }};
csl_field fd2{
fd1 fd1;
fd2(){
set_value(fd1.get_value());
}};
```

**csl_multi_dim_bitrange** *obj_name*(num_expr);

**DESCRIPTION :**

Creates a multidimensional bitrange object called *obj_name*: param number of dimensions;

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

A multidimensional bitrange *mbr1* is created with 3 dimensions used to set the width of a signal *s1*.

CSL CODE

```
    csl_multi_dim_bitrange mbr1(3);
    csl_unit u{
      csl_signal s1(mbr1);
      u(){}
    };
```

VERILOG CODE

//

**csl_multi_dim_bitrange** *obj_name*(*multi_dim_br_obj*);

## DESCRIPTION :

Creates a multidimensional bitrange object called *obj_name*: param multi dimensional br object - copy constructor;

[ ***CSL Language Commands Summary*** ]

## EXAMPLE :

In this example *mbr2* will has the same number of dimensions like *mbr1*.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
csl_multi_dim_bitrange mbr2(mbr1);
csl_unit u{
  csl_signal s1(mbr1), s2(mbr2);
  u(){}
};
```

VERILOG CODE

*object_multi_dim*.**set_dim_width**(*numeric_expr*,*numeric_expr*);

## DESCRIPTION :

Set the with of *object_multi_dim* :

param1: dimension index (dimension index start from 0 and go up to num_of_dims-1);
param2: width of the dimension

[ ***CSL Language Commands Summary*** ]

## EXAMPLE :

In this example a multidimensional bitrange *mbr1* is created with 3 dimensions. The width of each dimension is set using the *set_dim_width* method.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
mbr1.set_dim_width(0,2);
mbr1.set_dim_width(1,4);
mbr1.set_dim_width(2,8);
csl_unit u{
  csl_signal s1(mbr1);
  u(){}
};
```

VERILOG CODE

*object_multi_dim*.**set_dim_bitrange**(num_expr,br_obj_name);

## DESCRIPTION :

Set the bitrange of *object_multi_dim* :

param1: dimension index (dimension index start from 0 and go up to num_of_dims-1);
param2: imple a csl_bitrange to be set as a dimension to the multi dim bitrange;

[ ***CSL Language Commands Summary*** ]

## EXAMPLE :

In this example two multidimensional bitranges *mbr1* and *mbr2* with 3 dimensions are created. The *set_dim_bitrange()* method is used to set the width for the *mbr2* dimensions like *mbr1* dimensions.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
mbr1.set_dim_width(0,2);
mbr1.set_dim_width(1,4);
mbr1.set_dim_width(2,8);
csl_multi_dim_bitrange mbr2(3);
mbr2.set_dim_bitrange(0, mbr1);
mbr2.set_dim_bitrange(1, mbr1);
mbr2.set_dim_bitrange(2, mbr1);
csl_unit u{
  csl_signal s1(mbr1), s2(mbr2);
  u(){}
};
```

VERILOG CODE

*object_multi_dim*.**set_dim_range**(num_expr,num_expr,num_expr);

**DESCRIPTION :**

Set the range of *object_multi_dim* :

param1: dimension index (dimension index start from 0 and go up to num_of_dims-1);
param 2 : lower_index of the dimension
param 3 : upper_index of the dimension

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

In this example a multidimensional bitrange *mbr1* with 3 dimensions is created. The *set_dim_range()* method is used to set the range for each dimension.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
mbr1.set_dim_range(0,0,1);
mbr1.set_dim_range(1,0,3);
mbr1.set_dim_range(2,0,7);
csl_unit u{
  csl_signal s1(mbr1);
  u(){}
};
```

VERILOG CODE

*object_multi_dim*.**set_dim_offset**(num_expr,num_expr);

**DESCRIPTION :**

Set the range of *object_multi_dim* :

param1: dimension index (dimension index start from 0 and go up to num_of_dims-1);
param 2 : offset of the dimension

[ ***CSL Language Commands Summary*** ]

**EXAMPLE :**

In the multidimensional bitrange *mbr1*, the width of the dimensions are changed, using *set_dim_offset* method.

CSL CODE

```
csl_multi_dim_bitrange mbr1(3);
mbr1.set_dim_width(0,2);
mbr1.set_dim_width(1,4);
mbr1.set_dim_width(2,8);

mbr1.set_dim_offset(0,4);
mbr1.set_dim_offset(1,8);
mbr1.set_dim_offset(2,2);

csl_unit u{
  csl_signal s1(mbr1);
  u(){}
};
```

VERILOG CODE