Register types:
Action register

Atomic register
```
add_logic(lock_bit);
csl_bool get_lock_bit();
```
DFF register

Event register

Interrupt register
```
set_mask(numeric_expression);
```

LFSR register
```
set_feedback_mask(numeric_expression);
set_feedback_element(feedback_element);
```

Shift register
```
add_logic(shift_direction, left | right);
add_logic(shift_amount,numeric_expression);
set_final_shift_value(numeric_expression);
```

Semaphore register

Statistic register

Status register
```
set_mask(numeric_expression);
```

Table
```
csl_rom table_name;csl_rom table_name;
add_value(address,numeric_expression);
csl_column column_name(object_type,num_elements);
csl_row row_name(object_type,num_elements);
add_column(column_name);
add_row(row_name);
matrix(obj,x_dim,y_dim);
```

```
add_logic(serial_input);
add_logic(serial_output);
add_logic(init[,init_value]);
```

```
add_logic(clear[,clear_value]);
```

```
add_logic(gray_output);
```

```
add_logic(stop,stop_value);
```

```
add_logic(direction_control);
```

csl_list *object_name*.**get_attributes()**;

DESCRIPTION :

Returns a list of attributes that are applied to *object_name* eithin the memory map.

[CSL Register Syntax and Command Summary]

EXAMPLE :

Create two memory maps that have register file elements.

FIGURE 1.1 Two memory map page with a register file elements

CSL CODE

```
csl_register regf{
  regf(){
    set_type(register);
    set_width(8);
    set_depth(32);
      }
};
csl_memory_map_page mpage1{
regf regf1;
   mpage1(){
   add_address_range(0,300);
   regf1.set_attributes(read);
   }
};
csl_memory_map_page mpage2{
regf regf2;
   mpage2(){
     add_address_range(0,255);
     regf2.set_attributes(regf1.get_attributes())
   }
};
```

VERILOG CODE

```
`define <MMN>_OBJ_NAME_ATTRIBUTES attribute_list;
```

signal_object_name reg_name.**get_cnt_dir_signal();**

DESCRIPTION :

Return the name for the signal that drive the count direction.

[CSL Register Syntax and Command Summary]

EXAMPLE :

Create two counter that will count up or down from an initial value. The count direction is driven by a signal.

CSL CODE

```
csl_register reg_cnt1{
  reg_cnt1(){
    set_type(counter);
    set_width(8);
    add_logic(direction_control);
  }
};
csl_register reg_cnt2{
  reg_cnt2(){
    set_type(counter);
    set_width(8);
    add_logic(direction_control);
  }
};
csl_unit top{
csl_signal cnt_dir_signal;
reg_cnt1 reg_cnt1(.cnt_port(cnt_dir_signal));
reg_cnt2 reg_cnt2(.cnt_port(reg_cnt1.get_cnt_dir_signal());
}
};
```

VERILOG CODE

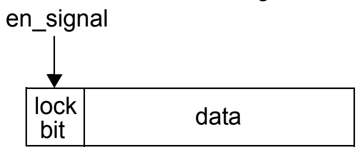
```
module reg_cnt1(reset, en, clk1, cnt_port, count_out);
    parameter init_val = 8'b0;
    parameter step = 2;
    input reset, en, clk1,cnt_port;
    output [7:0] count_out;
    reg [7:0] count_out;
    always @(posedge clk1 or negedge reset) begin
        if(!reset) begin count_out = init_val; end
        else if(en) begin
            case (cnt_signal)
            0:count_out = count_out - step;
            1:count_out = count_out + step;
            default:$display("Wrong value for direction");
            endcase
         end
    end
endmodule
module reg_cnt2(reset, en, clk1, cnt_port, count_out);
    parameter init_val = 8'b0;
    parameter step = 2;
    input reset, en, clk1,cnt_port;
    output [7:0] count_out;
    reg [7:0] count_out;
    always @(posedge clk1 or negedge reset) begin
        if(!reset) begin count_out = init_val; end
        else if(en) begin
            case (cnt_signal)
            0:count_out = count_out - step;
            1:count_out = count_out + step;
            default:$display("Wrong value for direction");
            endcase
         end
    end
endmodule

module top ;
wire cnt_dir_signal;
reg_cnt1 reg_cnt1(.cnt_port(cnt_dir_signal));
reg_cnt2 reg_cnt2(.cnt_port(cnt_dir_signal));
endmodule
```

add_logic(lock_bit**);**

DESCRIPTION :

Connect the lock signal to the lock enable bit in the atomic register. A lock bit locks a register and sets a number associated with the processor that locked the register. Only that processor can unlock the register. The lock bit is toggled

FIGURE 1.2 An atomic register



EXAMPLE :

Create an atomic register and connect a signal to the lock bit

CSL CODE

```
csl_register regA{
  regA(){
    set_type(atom);
    set_width(8);
    add_logic(lock_bit);
  }
};
```

VERILOG CODE

```
//AV
module atom_reg(en,rst,clk,data_in,data_out);
    input en, rst, clk;
    reg lock_bit;
    input [7:0] data_in;
    output [7:0] data_out;
    reg [7:0] data_out;
    always @(en) begin lock_bit = en; end
    always @(posedge clk or negedge rst) begin
        if(!rst) begin data_out = 8'b0; end
        else if(!lock_bit) begin data_out = data_in; end
    end
endmodule
```

csl_bool **get_lock_bit();**

DESCRIPTION :

Return the value of the lock bit flag.

EXAMPLE :

Create two atomic registers and connect same signal to the lock bit

CSL CODE

```
csl_register regA{
  regA(){
    set_type(dff);
    set_width(8);
    add_logic(lock_bit);
  }
};
csl_register regB{
  regB(){
    set_type(dff);
    set_width(8);
    set_lock_enable_bit(regA.get_lock_enable_bit());
  }
};
```

VERILOG CODE

```
//AV
module atom_reg(en,rst,clk,data_in,data_out);
    input en, rst, clk;
    reg lock_bit;
    input [7:0] data_in;
    output [7:0] data_out;
    reg [7:0] data_out;
    always @(en) begin lock_bit = en; end
    always @(posedge clk or negedge rst) begin
        if(!rst) begin data_out = 8'b0; end
        else if(!lock_bit) begin data_out = data_in; end
    endendmodule
```

1.0.0.0.1 Control register

1.0.0.0.2 DFF register

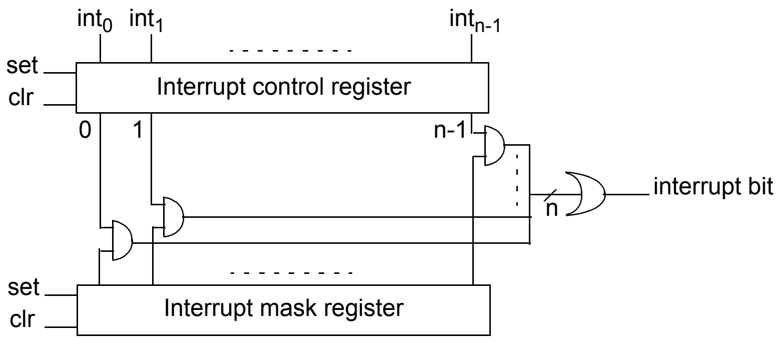
1.0.0.0.3 Event register

1.0.0.0.4 Interrupt register

set_mask(*numeric_expression***);**

DESCRIPTION :

The interrut mask register is used to select/enable the active interrupts in the interrupt register.



EXAMPLE :

Create an interrupt register and set a mask for is

CSL CODE

```
    csl_register reg_int{
     reg_int(){
        set_type(int);
        set_width(8);
        set_mask(8'b01101110);
      }
    };
```

VERILOG CODE

```
    //AV
    `define MASK 8'b01101110
    module int_reg(rst,clk,en,int_in,int_out);
    integer i;
    input rst, clk, en;
    input [7:0] int_in;
    output int_out;
    reg int_out;
    reg out;
    reg [7:0]var;
    always @(posedge clk or negedge rst) begin
        if(!rst) begin var = 8'b0; end
        else if(en) begin
            var = `MASK & int_in;
            for (i=0;i<8;i=i+1) begin out = out | var[i]; end
        end
        int_out = out;
    end
endmodule
```

1.0.0.0.5 LFSR register

set_feedback_mask(*numeric_expression***);**

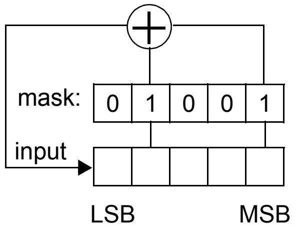
DESCRIPTION :

The feedback mask decides which bits from the LFSR are connected to the loop.

EXAMPLE :

Create a 5-bit LFSR register with 2 bits connected to the loop.

FIGURE 1.3 A LFSR register with the feedback mask



CSL CODE

```
csl_register reg_lfsr{
  reg_lfsr(){
    set_type(lfsr);
    set_width(5);
    set_feedback_mask(5'b10010);
  }
};
```

VERILOG CODE

```
//AV
module LFSR_5bit(clk,rst,out);
    input clk,rst;
    output [4:0] out;
    reg [4:0] out;
    reg [4:0] lfsr_reg;
    reg fdb_out;
    always @(posedge clk or negedge rst) begin
        if(!rst) begin lfsr_reg = 8'b00001; end
        else begin
            fdb_out = lfsr_reg [4] ^ lfsr_reg [1];
            lfsr_reg = lfsr_reg << 1;
            lfsr_reg [0] = fdb_out;
        end
        out = lfsr_reg;
    end
endmodule
```

set_feedback_element(*feedback_element***);**

DESCRIPTION :

Sets the elements that are used in the feedback of the LFSR. Feedback comes from a selection of bits from the register and constitutes either XORing or XNORing these bits and the output of the feedback is connected with the input of the register (with the least semnificant bit).
If is not specified, the feedback elements will be with XOR by default.

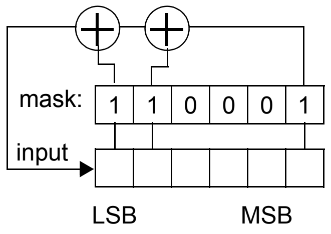
TABLE 1.1 Feedback elements

Feedback element	Description
xor	the bits that create the feedback are XORed toghether
xnor	the bits that create the feedback are XNORed toghether

EXAMPLE :

Create a 6-bit LFSR register with 3 bits connected to the loop.

FIGURE 1.4 A LFSR register with the feedback mask



CSL CODE

```
csl_register reg_lfsr{
  reg_lfsr(){
    set_type(lfsr);
    set_width(5);
    set_feedback_element(xnor);
  }
};
```

VERILOG CODE

```
//AV
module LFSR_6bit(clk,rst,out);
    input clk,rst;
    output [5:0] out;
    reg [5:0] out;
    reg [5:0] lfsr_reg;
    reg fdb_out;
    always @(posedge clk or negedge rst) begin
        if(!rst) begin lfsr_reg = 8'b000001; end
        else begin
```

```
            fdb_out = lfsr_reg [5] ^ lfsr_reg [1] ^ lfsr_reg [0];
            lfsr_reg = lfsr_reg << 1;
            lfsr_reg [0] = ~fdb_out;
        end
        out = lfsr_reg;
    end
endmodule
```

1.0.0.0.6 Shift register

add_logic(shift_direction, left | right**);**

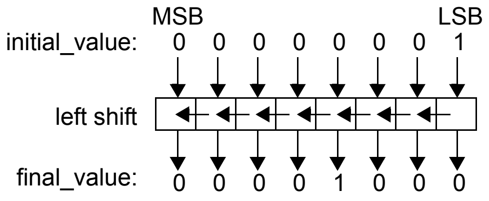
DESCRIPTION :

Sets the type of the *register_shifter_object_name* register. The shifter types are showed in the Table 7.12

EXAMPLE :

Create a logical shift register with 8-bit wide.

FIGURE 1.5 Logical left shifter will shift until the given value.



CSL CODE

```
csl_register shift1{
  shift1(){
    set_type(lfsr);
    set_width(8);
    set_init_val(8'b00000001);
    add_logic(shift_direction, left);
  }
};
```

VERILOG CODE

```
//AV
module shift_1X8(clk,sr_out,rst);
    parameter init_val = 8'b00000001;
    parameter final_val = 8'b00001000;
    input clk, rst;
    output [7:0] sr_out;
    reg [7:0] sr_out;
    always @(posedge clk or negedge rst) begin
        if (!rst) begin sr_out = init_val; end
        else if(sr_out != final_val) begin sr_out = sr_out << 1; end
    end
endmodule
```

add_logic(shift_amount,*numeric_expression***);**

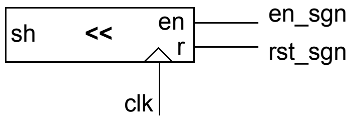
DESCRIPTION :

Set the number of bits to be shifted for a shifter.

EXAMPLE :

Create a shift register that will shift to the left. The number of bits that be shifted is set by *set_shift_amount* command.

FIGURE 1.6 A shift register



CSL CODE

```
csl_register shift1{
  shift1(){
    set_type(lfsr);
    set_width(8);
    set_shift_type(shl);
    set_shift_amount(2);
    set_reset_value(1);
    set_final_shift_value(8'b01000000);
  }
};
```

VERILOG CODE

```
//AV
module shifter_by_2(rst_sgn,clk,en_sgn,out);
    parameter reset_val = 8'b00000001;
    parameter final_val = 8'b01000000;
    parameter amount = 2;
    input rst_sgn, clk, en_sgn;
    output [7:0] out;
    reg [7:0] out;
    reg [7:0] o;
    integer i;
    always @(posedge clk or negedge rst_sgn) begin
    if(!rst_sgn) begin
        out = reset_val;
        var = reset_val;
    end
       if(en_sgn) begin
           for(i=0; i<amount; i=i+1) begin
```

```
                var = var << 1;
            end
        end
        out = var;
     end
endmodule
```

set_final_shift_value(*numeric_expression***);**

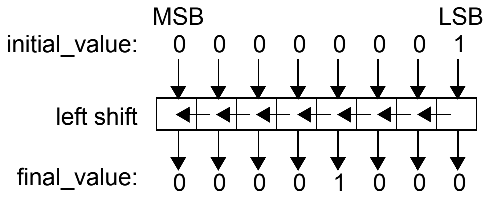
DESCRIPTION :

Set the stop shifting value.

EXAMPLE :

Create a logical shift register with 8-bit wide witch will shift until the final value.

FIGURE 1.7 Logical left shift until the given value.



CSL CODE

```
csl_register shift1{
  shift1(){
    set_type(lfsr);
    set_width(8);
    set_shift_type(shl);
    set_shift_amount(2);
    set_reset_value(1);
    set_final_shift_value(8'b01000000);
  }
};
```

VERILOG CODE

```
    //AV
    module shift_1X8(clk,sr_out);
        parameter init_val = 8'b00000001;
        parameter final_val = 8'b00001000;
        input clk;
        output [7:0] sr_out;
        reg [7:0] sr_out;
        always @(posedge clk) begin
            if(sr_out != final_val) begin sr_out = sr_out << 1; end
        end
endmodule
```

1.0.0.0.7 Semaphore register

1.0.0.0.8 Statistic register

1.0.0.0.9 Status register

set_mask(*numeric_expression***);**

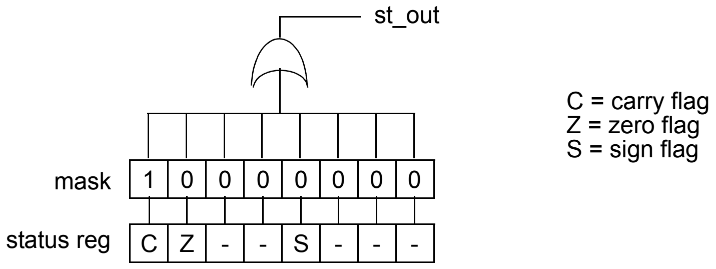
DESCRIPTION :

The mask decides which bits from the STATUS register can be read/written.

EXAMPLE :

Create a status register and set a mask to read the carry flag value.

FIGURE 1.8 A status register with mask



CSL CODE

```
csl_register shift1{
  shift1(){
    set_type(status);
    set_width(8);
    set_mask(8'b10000000);
  }
};
```

VERILOG CODE

```
//AV
'define ST_REG_MASK 8'b10000000
module st_reg(st_out);
    output [7:0] st_reg;
endmodule
```

```
csl_rom table_name;
```

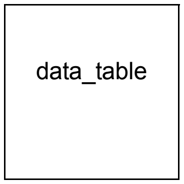
DESCRIPTION :

Create a new csl_rom called A table can be used to group some informations in it.

EXAMPLE :

Create the *data_table* table.

FIGURE 1.9 A csl_rom



CSL CODE

```
    //AV
    csl_rom data_table;
```

VERILOG CODE

```
    //AV
```

`add_value``(`*`address`*`,`*`numeric_expression`*`);`

DESCRIPTION :

Adds a value to a specified address in the rom table.

EXAMPLE :

csl_table *table_name;*

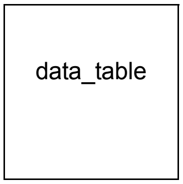
DESCRIPTION :

Create a new csl_table called A table can be used to group some informations in it.

EXAMPLE :

Create the *data_table* table.

FIGURE 1.10 A csl_table



CSL CODE

```
//AV
csl_table data_table;
```

VERILOG CODE

```
//AV
```

csl_column *column_name*(*object_type*,*num_elements*);

DESCRIPTION :

Create a new csl_column called *first_column*. Multiple columns can be used to create a table.

EXAMPLE :

Create the *first_column* column.

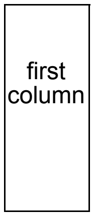


FIGURE 1.11 A csl_column

CSL CODE

```
//AV
csl_column first_column;
```

VERILOG CODE

```
//AV
```

csl_row *row_name***(***object_type*,*num_elements***)**;

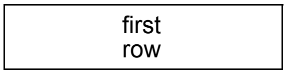
DESCRIPTION :

Create a new csl_row called *first_row*. Multiple rows can be used to create a table.

EXAMPLE :

Create the *first_row* row.

FIGURE 1.12 A csl_row



CSL CODE

```
//AV
csl_column first_column;
```

VERILOG CODE

```
//AV
```

add_column(*column_name***);**

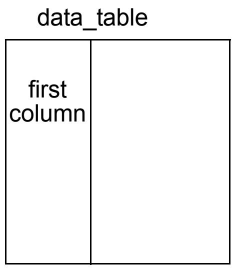
DESCRIPTION :

Adds a column called *column_name* to the table called

EXAMPLE :

Create a csl_table called *data_table* and a csl_column called *first_column* and then add the column to the table.

FIGURE 1.13 Add a column to a table



CSL CODE

```
//AV
csl_table data_table;
csl_column first_column;
data_table.add_column(first_column);
```

add_row(*row_name***);**

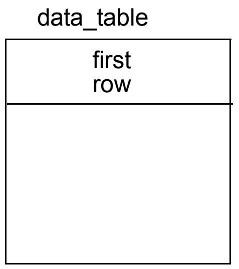
DESCRIPTION :

Adds a row called *row_name* to the table called

EXAMPLE :

Create a csl_table called *data_table* and a csl_row called *first_row* and then add the row to the table.

FIGURE 1.14 Add a row to a table



CSL CODE

```
//AV
csl_table data_table;
csl_column first_column;
data_table.add_column(first_column);
```

matrix(*obj*,*x_dim*,*y_dim***);**

DESCRIPTION :

Obj can be a table

EXAMPLE :

CSL CODE

```
    //csl code goes here
```

***add_logic(**serial_input);*

```
   port: input - serial_input
```

DESCRIPTION :

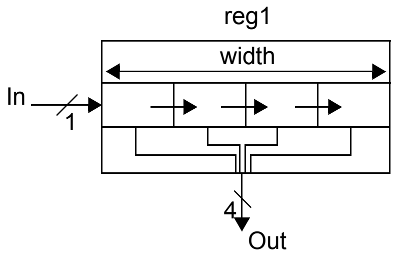
The input signal is connected in serial mode with the register.

[CSL Register Syntax and Command Summary]

EXAMPLE :

Create a shift register that has serial input and paralel output.

FIGURE 1.15 A register with serial input and paralel output



CSL CODE

```
csl_register reg1{
  reg1(){
    set_type(counter);
    set_width(4);
    add_logic(serial_input);
  }
};
```

VERILOG CODE

```
//AV
module shift_serial_input(rst,clk,in,out);
    input in, clk, rst;
    output [3:0] out;
    reg [3:0] out;
    always @(posedge clk or negedge rst) begin
        if(! rst) begin out = in; end
        else begin out = {in,out} >> 1; end
    end
endmodule
```

```
add_logic(serial_output);
   port: output - serial_output
```

DESCRIPTION :

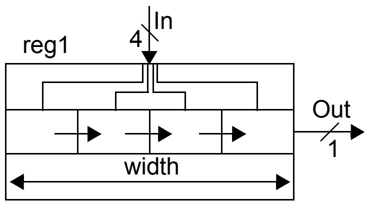
The output signal is connected in serial mode with the register.

[CSL Register Syntax and Command Summary]

EXAMPLE :

Create a shift register that has serial output and paralel input.

FIGURE 1.16 A register with paralel input and serial output



CSL CODE

```
   csl_register reg1{
     reg1(){
       set_type(counter);
       set_width(4);
       add_logic(serial_output);
     }
   };
```

VERILOG CODE

add_logic(init[,*init_value*]**);**

DESCRIPTION :

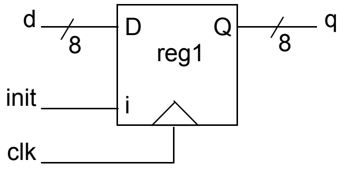
When the init signal is active the memory element is initalized with this value.

[CSL Register Syntax and Command Summary]

EXAMPLE :

Creates a register that have an init signal.

FIGURE 1.17 A register with init signal



CSL CODE

```
csl_register reg1{
  reg1(){
    set_type(register);
    set_width(8);
    add_logic(init,0);
  }
};
```

VERILOG CODE

```
//AV
module reg1(clk,d,q,init);
    parameter init_val = 8'b0;
    input clk;
    input [7:0] d;
    output [7:0] q;
    reg [7:0] q;
    always @(posedge clk) begin
       if(init_sgn) begin q = init_val; end
       else begin q = d; end
    end
endmodule
```

add_logic(clear[,*clear_value*]**);**

DESCRIPTION :

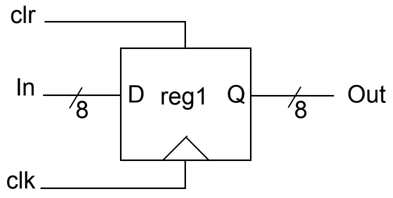
After a clear operation the memory element is set to this value.

[CSL Register Syntax and Command Summary]

EXAMPLE :

Create a registers. The registers will be cleared when the clear signal will be active.

FIGURE 1.18 A register with clear signal.



CSL CODE

```
csl_register reg1{
  reg1(){
    set_type(register);
    set_width(8);
    add_logic(clear,0);
  }
};
```

VERILOG CODE

```
//AV
module reg1(clr,clk,in,out);
    parameter clr_val = 8'b0;
    input clr,clk;
    input [7:0] in;
    output [7:0] out;
    reg [7:0] out;
    always @(posedge clk) begin
        if(clr) begin out = clr_val; end
        else begin out = in; end
    end
endmodule
```

```
add_logic(gray_output);
   port: output - gray_output
```

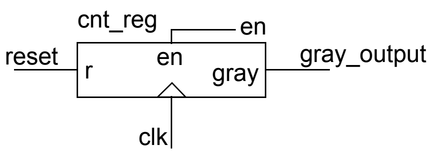
DESCRIPTION :

Automatical create the counter output port signal with the name *gray_output* for the *counter_reg_name.* The *gray_output* signal is in Gray code and the width is equal with the width of counter register.

[CSL Register Syntax and Command Summary]

EXAMPLE :

FIGURE 1.19



CSL CODE

```
   csl_register cnt_reg{
     cnt_reg(){
       set_type(counter);
       set_width(8);
       add_logic(gray_output);
     }
   };
```

VERILOG CODE

```
   module register(reset, clk, en, reg_out);
   input reset;
   input clk;
   input en;
   output reg [7:0] gray_out;

   always @(posedge clk )begin
     if(!reset)
        gray_out =8'b0;
        else
        if(en)
        gray_out=gray_out+8'b00000001;
        end
    endmodule
```

add_logic(stop,*stop_value***);**

DESCRIPTION :

When the counter reaches this value then the counter stops counting and does not reset to the start value until the init signal is asserted.

[CSL Register Syntax and Command Summary]

EXAMPLE :

Create a counter that count up to 128, and stop_value is 50.

CSL CODE

```
csl_register reg_cnt{
  reg_cnt(){
    set_type(counter);
    set_width(8);
    add_logic(count_direction,up);
    add_logic(start_value,1);
    add_logic(end_value,128);
    add_logic(count_amount,1);
    add_logic(stop,50);
  }
};
```

VERILOG CODE

```
//AV
module cnt_up(rst,en1,clk,count_out1);
    parameter start_val = 8'b0;
    parameter end_val = 8'b10000000;
    parameter stop_value =8’b00110010;
    input rst, en1, clk;
    output [7:0] count_out1;
    reg [7:0] count_out1;
    always @(posedge clk or negedge rst) begin
        if(!rst) begin count_out1 = start_val; end
        else if(en1) begin
            if ((count_out1 < end_val) && (count_out1 == stop_value))
            count_out1 = count_out1 + 1;

         end
    end
endmodule
```

```
add_logic(direction_control);
   port: input - cnt_dir
```

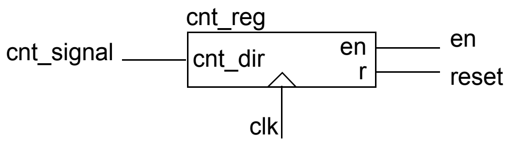
DESCRIPTION :

Create the count direction port with the name *cnt_dir* for the *counter_reg_name* counter register. ***[CSL Register Syntax and Command Summary]***

EXAMPLE :

Create a counter that will count up or down from an initial value. The count direction is driven by a signal.

FIGURE 1.20 A counter with a signal that will control the count direction



CSL CODE

```
csl_register reg_cnt{
  reg_cnt(){
    set_type(counter);
    set_width(8);
    add_logic(direction_control);
  }
};
csl_unit top{
csl_signal cnt_dir_signal;
reg_cnt reg_cnt(.cnt_dir(cnt_dir_signal));
top(){}
};
```

VERILOG CODE

```
//AV
module reg_cnt(reset, en, clk, cnt_dir, count_out);
    parameter init_val = 8'b0;
    parameter step = 2;
    input reset, en, clk, cnt_dir;
    output [7:0] count_out;
    reg [7:0] count_out;
    always @(posedge clk or negedge reset) begin
        if(!reset) begin count_out = init_val; end
        else if(en) begin
            case (cnt_dir)
            0:count_out = count_out - step;
            1:count_out = count_out + step;
            default:$display("Wrong value for direction");
            endcase
         end
    end
endmodule

module top;
wire cnt_dir_signal;
reg_cnt reg_cnt(.cnt_dir(cnt_dir_signal));
endmodule
```