CHAPTER 2 CSL Memory Map

TABLE 2.1 Chapter Outline

2.1 CSL Memory Map Command Summary

2.1.1 Memory map rules

1. Every unit has its own local address map in the conceptual model.

2. An object from a unit can only be added to a memory map once.

3. Multiple instances of the same unit all have the same local address map

4. An unit can contains multiple instances of unit's mulitple instance.

5. The parent map is formed by the concatenation of the address maps of the child units

6. memory map ranges are declared as free or reserved.

7. The address ranges which an object is added to is no longer free.

8. It is illegal to add an object to an address range which is reserved.

9. It is illegal to add an object outside of a declared address range

10. It is illegal to add an object to a used/not free address range

11. It is illegal to have a memory map page which is not added to a memory map.

12. It is illegal to have a memory map with out memory map page instances.

13. It is illegal to have flat memory map with multiple page instances..

14. It is illegal for a flat memory map to have hierarchical memory pages.

15. It is illegal for a virtual memory map to have hierarchical memory pages.

16. It is legal to have Hierarchical memory pages for hierarchical memory map.

17. What can be added to a memory map page? Instances of an addressable object which include:
csl_memory
csl_register
csl_register_file
csl_fifo

18. Before adding an object ot a memory map page the following are required:
add all address ranges (legal, reserved)

19. After the memory map page is built it is then added to the memory map.

20. It is legal to add the memory map page to the memory map and then add objects to the memory page.

21. mandatory cmds for a memory map are set_type

22. memory map must have memory map pages instantiated

23. mandatory cmds for a memory map page are add_address_range

24. a memory map page must have at least one adressable object added to it. A memory map page cannot be empty

25. The design address limits for the memory map is optional
    set_address_limits(lower, upper) -this is the valid address range for the chip
    set_address_width(numeric_expression)-this is the width of the address bus on the chip
    the limits and the width can be set. This is related to the address bus on the chip.

26. If the design address limits for the memory map is set then check all of the pages to see that the memory map page address range is wth in the design address limits and that the width of the memory map page...

27. A memory map page has a set_unit

28. A memory map page has to have a set_unit before adding addressable objects to the memory map page.

29. Every addressable object added to the memory map page has to be instantiated in the unit specified by set_unit in the memory map page.

30. Only one set_unit can be called per memory map page

```
csl_unit a {
 rf rf0;
 a() {}
};
csl_memory_map_page mpa {
  mpa(){
    set_unit(a);
    add(rf0);
  }
};
```

29. A unit can only be added to one memory map page

30. It is illegal to add a unit can only be added to more than one memory map page

31. The unit instances must have different base addresses

32. Multiple instances of the same page need to have different base addresses

33. If no base address is specified for a unit instance then the base address of the unit instance in the design memory map is calculted by the order of the instantiation of the pages in the memory map csl file(s).

34. The base address for the unit instances can be set using the following code example

```
csl_unit a { a(){} };
csl_unit b { a a0; a a1;  b(){} };
csl_unit top  { b b0; b b1; top(){} };

csl_memory_map_page apage{
  apage(){
    set_unit(a);
    add_address_range(0,511);
    set_address_increment(2);
  }
};

csl_memory_map mm{
  mm(){
    set_type(hier);
    top.b0.a0.set_base_address(  500);
    top.b0.a1.set_base_address( 1000);
    top.b1.a0.set_base_address( 1500);
    top.b1.a1.set_base_address( 2000);
  }
};
```

35. Memory map pages are automatically added to the memory map.

36. Memory map pages will be automatically self registering with the design's single memory map class. The memory map page will NOT be instantiated in the memory map. This is consistent with testbench and vc's.

37. the ***add_reserved_address_range*** can NOT be called on a memory_map_page instance.
38. A range may only be set once in a memory page. Setting any part of the range again is illegal
39. Memory map page methods can only be called in memory map page constructors.

2.1.2 Usage tables

CSL Memory map page
can be defined and instantiated
- can be defined in

TABLE 2.2 CSL memory map page definition in other CSL classes

CSL class	Can be defined in
global scope	YES
CSL Unit	-
CSL Signal Group	-
CSL Interface	-
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	-

- can be instantiated in

TABLE 2.3 CSL memory map page instantiation in other CSL classes

CSL class	Can be instantiated in
global scope	-
CSL Unit	-
CSL Signal Group	-
CSL Interface	-
CSL Testbench	-

TABLE 2.3 CSL memory map page instantiation in other CSL classes

CSL class	Can be instantiated in
CSL Vector	-
CSL State Data	-
CSL Register	-
CSL Register File	-
CSL Fifo	-
CSL Memory Map	YES
CSL Memory Map Page	YES
CSL Isa Element	-
CSL Isa Field	-
CSL Field	-

The manadatory commands for the memory map page are:

Memory map type	Manadatory commands
hierarchical	set_unit_name(unit_name);
virtual_with_base_address	set_unit_name(unit_name);
virtual_with_page_number_and_address	set_unit_name(unit_name);

f

CSL Memory map

can be defined and NOT instantiated

- can be defined in

TABLE 2.4 CSL memory map definition in other CSL classes

CSL class	Can be defined in
global scope	YES
CSL Unit	-
CSL Signal Group	-
CSL Interface	-
CSL Testbench	-
CSL Vector	-
CSL State Data	-
CSL Register	-
CSL Register File	-

TABLE 2.4 CSL memory map definition in other CSL classes

CSL class	Can be defined in
CSL Fifo	-
CSL Memory Map	-
CSL Memory Map Page	-
CSL Isa Element	-
CSL Isa Field	-
CSL Field	-

The manadatory commands for the memory map page are:

Manadatory commandas for memory map page

~~set_unit_name(unit_name);~~

~~memory_map_page_name.add_address_range(lower_bound,upper_bound);~~

Memory Map methods to be called in units:

```
set_address_range(num_expr,num_expr);
set_access_rights( addressable_object | address_range, access_right);
add(addressable_object, start_address [,access_rights] [,symbol]);
set_address_increment(numeric_expression);
[instance_name.]set_id(num_expr);
//setting id for use w/ address bus
auto_mapper(off);
//turning off automapper for the current unit so that the page would
//span across multiple units until it's turned back on
```

Naming

```
set_symbol_max_length(numeric_expression);
```

Word width

```
set_data_word_width(numeric_expression);
```

NOTE:to be explained

```
set_access_rights_enum( enum ); //i need more details on this one
set_next_address(numeric_expression); //same as above
```

Alignment

```
set_alignment(numeric_expression);
```

NOTE:propsed for removal

```
~~memory_map_page_name.add_reserved_address_range(lower_bound,upper_bound);~~ //use access rights on address range?
~~addr_obj.add_to_memory_map();~~ //leave to automapper or manually add
~~csl_memory_map_page memory_map_page_name;~~
~~memory_map_page_name.add(memory_map_page_object_name);~~
```

Memory map commands

Manadatory commands for memory map

set_type(memory_map_type);

Memory Map: Declaration

```
csl_memory_map memory_map_name;
```

Memory Map

```
set_base_address(num_expr);
set_top_unit(unit_object_name);
set_endianess(endianess_type);
```

Memory Map: Word width

```
set_data_word_width(numeric_expression);
```

Memory Map: Prefix

```
set_prefix(string);//needed now ?
```

Memory Map: Suffix

```
set_sufix(string);//needed now ?
```

NOTE:proposed for removal

~~auto_gen_memory_map();~~ //default

~~object_name.add_to_memory_map([address],[group,access_right]);~~

TBD:
set_addr_abs(constant_numeric); //new address
set_addr_rel(constant_numeric); //new address = offset from current address
//add set max number of words command

2.2 CSL Memory Map Commands

```
set_unit_name(unit_name);
```

DESCRIPTION :

//

[CSL Memory Map Command Summary]

EXAMPLE :

//

CSL CODE

```
//
```

VERILOG CODE

```
//
```

csl_memory_map_page *memory_map_page_name*;

DESCRIPTION :

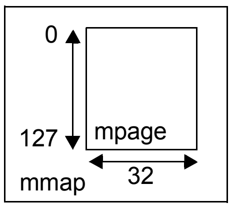
Creates a memory map page named *memory_map_page_name*. It is a scope delimited by curly braces. Each memory map page has an address range.

[CSL Memory Map Command Summary]

EXAMPLE :

In this example we have one memory map *mmap* which contains a memory map page *mpage*.

FIGURE 2.1



CSL CODE

```
csl_memory_map_page mpage{
  mpage(){
  add_address_range(0,128);
  }
};
csl_memory_map mmap{
  mpage mpage;
  mmap(){
  set_data_word_width(32);
  set_type(hierarchical);
  }
};
```

VERILOG CODE

```
set_address_range(num_expr,num_expr);
```

DESCRIPTION :

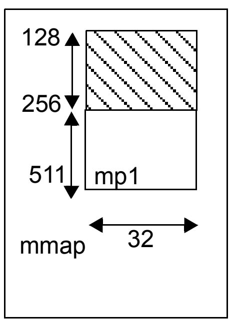
This command adds an addres range to a memory map page named *memory_map_page_name*
The address range is set using *lower_bound* and *upper_bound.*
The *add_reserved_address_range* command shows that a part of the initial address range is marked as reserved(and that add_address_range has to be called before add_reserved_address_range).
A flat memory map has one page .The address range for the one page is the address range for the entire memory map.

[CSL Memory Map Command Summary]

EXAMPLE :

In this example it was created a memory map named *mmap* with a memory map page named *mpage_1.*

FIGURE 2.2



CSL CODE

```
csl_memory_map_page mpage_1{
  mpage_1(){
  add_address_range(128,511);
  add_reserved_address_range(128, 256);
  }
};
csl_memory_map mmap{
  mpage_1 mp1;
  mmap(){
  set_data_word_width(32);
  set_type(hierarchical);
    }
```

```
};
```

VERILOG CODE

set_address_increment(*numeric_expression*);

DESCRIPTION :

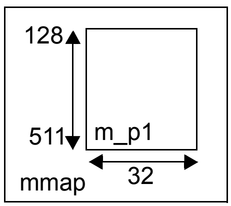
The address increment is the amount that the memory address increments from word to word for a memory map page named *memory_map_page_name* .The *numeric_expression* represent the increment of address.

[CSL Memory Map Command Summary]

EXAMPLE :

In this example it is set the address increment for a memory map page named *m_p1* from a memory map named *mmap .*

FIGURE 2.3



CSL CODE:

```
csl_memory_map_page memp1{
  memp1(){
   set_address_increment(4);
   add_address_range(128, 511);}
};
csl_memory_map mmap{
  memp1 m_p1;
  mmap(){
   set_data_word_width(32);
   set_type(hierarchical);
  }
};
```

VERILOG CODE:

set_next_address(*numeric_expression*)**;**

DESCRIPTION :

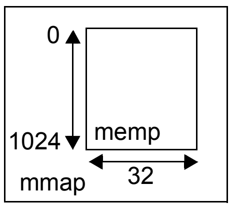
Sets the next address for a memory map page named *memory_map_page_name* . The *numeric_expression* represents the next address.

[CSL Memory Map Command Summary]

EXAMPLE :

In this example it is set the next address in a memory map page named *memp.*

FIGURE 2.4



CSL CODE

```
csl_memory_map_page mpage{
  mpage(){
   add_address_range(0, 1024);
   set_next_address(64);
   }
};
csl_memory_map mmap{
  mpage memp;
  mmap(){
   set_data_word_width(32);
   set_type(hierarchical);
}
};
```

VERILOG CODE

```
set_access_rights( addressable_object | address_range,
access_right);
```

DESCRIPTION :

With this command we can set the access rights for an *addressable_object* (eg. an instance of a register) or an address range `address_range` from a memory map page `memory_map_page_name`. The access rights can be the following:

TABLE 2.5

group
sw
hw
test
driver
user

TABLE 2.6

acc_right	Description
access_none	without access rights
access_read	access rights only for read
access_write	access rights only for write
access_read_write	access rights for read-write

[CSL Memory Map Command Summary]

EXAMPLE :

Sets the access rights for the memory map page named *mp.*

CSL CODE

```
csl_memory_map_page mp{
  mp(){
   set_access_rights(200: 251, HWR, access_read_write);
   set_access_rights( 100, SWR, access_read);
   }
};
csl_memory_map mmap{
  mp mp;
  mmap(){
   set_data_word_width(32);
   set_type(hierarchical);
```

```
    }
    };
```

VERILOG CODE

set_access_rights_enum(enum);

DESCRIPTION :

Sets the access rights for the enum named *enum*.
Can only be called in the mem_map scope. The category is an enum item from the enum set by set_access_rights_enum.If no enum has been set, the category can have one of the following default values: swr, hwr, driver, test, user.

[CSL Memory Map Command Summary]

EXAMPLE :

//

CSL CODE

```
csl_enum alu{
ADD,
SUB,
MUL
    };
csl_memory_map_page mpag{
  mpag(){
    set_data_word_width(32);
    add_address_range(0,512);
    }
};
csl_memory_map mmap{
  mpag mpag;
  mmap(){
  set_access_rights_enum(alu);
  set_type(hierarchical);
}
};
```

VERILOG CODE

```
//
```

memory_map_page_name.**add_reserved_address_range(**lower_bound,upper_bound**);**

DESCRIPTION :

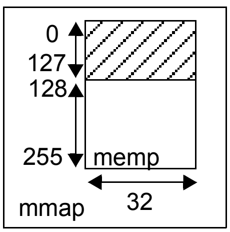
Adds a reserved address range in a memory map page *memory_map_page_name*. The reserved address range is adds by lower_bound and upper_bound.
This command shows that a part of the initial address range is marked as reserved(and that add_address_range has to be called before add_reserved_address_range).

[CSL Memory Map Command Summary]

EXAMPLE :

In a memory map page *mpage* is added a reserved address range.

FIGURE 2.5



CSL CODE

```
csl_memory_map_page mpage{
  mpage(){
   add_address_range(0,255);
   add_reserved_address_range(0,127);
   set_address_increment(2);
   }
};
csl_memory_map mmap{
  mpage mpage;
  mmap(){
  set_data_word_width(32);
  set_type(hierarchical);
}
};
```

VERILOG CODE

add(addressable_object, start_address [,access_rights] [,symbol]);

DESCRIPTION :

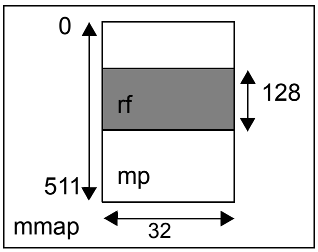
Adds to a memory map page named `memory_map_page_name` an *addressable_object* which can be a fifo , register, register file, memory that are instantiate in a unit.

[CSL Memory Map Command Summary]

EXAMPLE :

In this example it is added a register file named *rf* in a memory map page *mp.* The *symbol* for register file is “rf” and the *base_address* is 64.

FIGURE 2.6



CSL CODE

```
csl_register_file rf{
rf(){
   set_width(32);
   set_depth(128);
   }
};
csl_unit a {
   rf r1;
 };
csl_memory_map_page mpage{
mpage(){
   add_address_range(0,511);
   set_address_increment(2);
   add(a.r1, "rf", 64);
     }
};
csl_memory_map mmap{
mpage mpage1;
mmap(){
 set_type(hierarchical);}
};
```

addr_obj.**add_to_memory_map**();

DESCRIPTION :

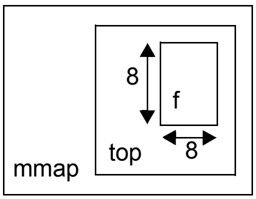
Adds to a memory map an addressable object *addr_obj* which can be a fifo , register, register file, memory that are instantiate in a unit.

[CSL Memory Map Command Summary]

EXAMPLE :

We added in a memory map *mmap* a fifo named *f* which is instantiated in a unit *top.*

FIGURE 2.7



CSL CODE

```
csl_fifo f1{
  f1(){
  set_width(8);
  set_depth(8);
  add_to_memory_map();
  }
};
csl_unit top{
  f1 f;
  top(){}
};
csl_memory_map mmap{
  mmap(){
set_type(hierarchical);
}
};
```

VERILOG CODE

```
`define DEFAULT_GROUP_hwr 2
`define DEFAULT_GROUP_test 3
`define DEFAULT_GROUP_driver 4
module f1(push,
          pop,
          full,
          empty,
          data_out,
```

```
        data_in,
        reset_,
        clock,
        valid);
parameter ADDR_WIDTH = 3'd4;
parameter DATA_WIDTH = 4'd8;
input push;
input pop;
input [7:0] data_in;
input reset_;
input clock;
output reg full;
output reg empty;
output reg [7:0] data_out;
output reg valid;
reg [ADDR_WIDTH - 1:0] wr_addr;
reg [ADDR_WIDTH - 1:0] rd_addr;
reg wr_en;
reg rd_en;
assign   full = wr_addr + 1 == rd_addr;
assign   empty = wr_addr == rd_addr;
assign   wr_en = !full && push;
assign   rd_en = !empty && pop;
f1_fifo_memory fifo_memory_instance(.clock(clock),
                                    .data_in(data_in),
                                    .data_out(data_out),
                                    .rd_addr(rd_addr),
                                    .rd_en(rd_en),
                                    .reset_(reset_),
                                    .valid(valid),
                                    .wr_addr(wr_addr),
                                    .wr_en(wr_en));

always @( posedge clock or negedge reset_ )  begin
  if ( ~reset_ )  begin
    rd_addr <= 1'd0;
  end
  else        if ( pop )  begin
      rd_addr <= rd_addr + 1'd1;
    end
```

```
  end

  always @( posedge clock or negedge reset_ )  begin
    if ( ~reset_ )  begin
      wr_addr <= 1'd0;
    end
    else       if ( push )  begin
        wr_addr <= wr_addr + 1'd1;
      end
  end
endmodule

module f1_fifo_memory(clock,
                      reset_,
                      data_in,
                      data_out,
                      valid,
                      wr_addr,
                      rd_addr,
                      wr_en,
                      rd_en);
  parameter ADDR_WIDTH = 3'd4;
  parameter DATA_WIDTH = 4'd8;
  parameter NUM_WORDS = (1'd1 << DATA_WIDTH);
  input clock;
  input reset_;
  input [DATA_WIDTH - 1:0] data_in;
  input [ADDR_WIDTH - 1:0] wr_addr;
  input [ADDR_WIDTH - 1:0] rd_addr;
  input wr_en;
  input rd_en;
  output reg [DATA_WIDTH - 1:0] data_out;
  output reg valid;
  reg [DATA_WIDTH - 1:0] internal_memory[1'd0:NUM_WORDS - 1'd1] ;

  always @( posedge clock or negedge reset_ )  begin
    if ( ~reset_ )  begin
      valid <= 1'd1;
    end
    else  begin
```

```
      valid <= rd_en;
      data_out <= internal_memory[rd_addr];
      if ( wr_en )  begin
        internal_memory[wr_addr] <= data_in;
      end
    end
  end
endmodule

module top();
  f1 f();
endmodule
```

memory_map_page_name.**add**(*memory_map_page_object_name*);

DESCRIPTION :

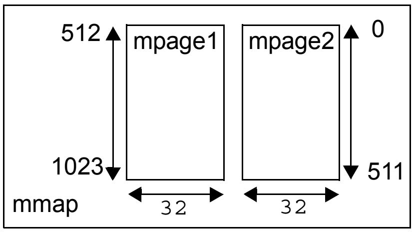
This command adds a memory map page object named *memory_map_page_object_name* to a memory map page *memory_map_page_name*.

[CSL Memory Map Command Summary]

EXAMPLE :

In this example two memory map pages *mpage1* and *mpage2* are added.

FIGURE 2.8



CSL CODE

```
csl_memory_map_page mpage1{
  mpage1(){
  add_address_range(512,1023);
  add(mpage1);
  }
};
csl_memory_map_page mpage2{
  mpage2(){
  add_address_range(0,511);
  add(mpage2);
  }
};
csl_memory_map mmap{
  mpage1 mpage1;
  mpage2 mpage2;
  mmap(){
  set_data_word_width(32);
  set_type(hierarchical);
}
};
```

VERILOG CODE

```
//
```

set_data_word_width(*numeric_expression*);

DESCRIPTION :

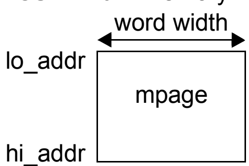
Sets the width of the words in the memory map page. If the memory map page width is specified, it is not necessary to declare the width of each individual word. The elements that will be added to the memory map page must have the width less or equal with the word with of memory map page.

[CSL Memory Map Command Summary]

EXAMPLE :

Create a memory map page named *mpage* with the word width 32.

FIGURE 2.9 A memory map page with word width 32



CSL CODE

```
csl_memory_map_page mpage{
  mpage(){
  set_data_word_width(32);
  }
};
csl_memory_map mmap{
  mpage mpage;
  mmap(){
   set_data_word_width(32);
   set_type(hierarchical);
}
```

VERILOG CODE

C++ CODE

```
const int WORD_WIDTH = 16;
```

`set_alignment(numeric_expression);`

DESCRIPTION :

Address alignment - addresses are byte, half-word (16-bit), word (32-bit), double-word (64-bit), quad-word (128-bit) aligned. The width of the memory elements in a particular memory element range.

TABLE 2.7 Byte aligned

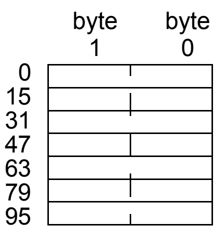
Type	Dimension
byte	8
half word	16
word	32
double word	64
long word	128
quad word	256

[CSL Memory Map Command Summary]

EXAMPLE :

Create two memory map pages named *mpage_0* and *mpage_1* set alignment as byte.

FIGURE 2.10 A memory map with byte alignment



CSL CODE

```
csl_memory_map_page mpage_0{
  mpage_0(){
  add_address_range(0,63);
  set_address_increment(2);
  set_alignment(16);
  }
};
csl_memory_map mmap{
  mpage_0 mpage_0;
  mmap(){
  set_data_word_width(16);
  set_type(hierarchical);
}
};
```

VERILOG CODE

```
`define <MMN>_ALIGN 16
```

set_endianess(*endianess_type*);

DESCRIPTION :

The endianess of the memory map can be specified with the **endianess** keword. The *endianess_type* can be either little endian or big endian respectively.

TABLE 2.8 Endianess Type

Endianess	Mnemonic
Little Endian	little_endian
Big Endian	big_endian

[CSL Memory Map Command Summary]

EXAMPLE :

Create a memory map page named *mpage0* and set endianess to big_endian.

FIGURE 2.11 Little Endian

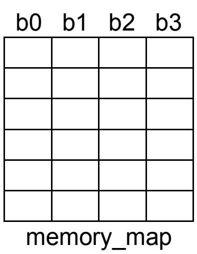
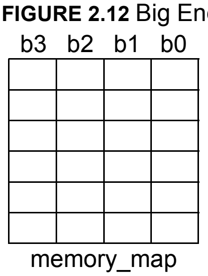


FIGURE 2.12 Big Endian



CSL CODE

```
csl_memory_map_page mpage0{
  mpage0(){
  add_address_range(0,128);
  set_endianess(big_endian);
  }
};
csl_memory_map mmap{
  mpage0 mpage0;
  mmap(){
  set_data_word_width(32);
```

```
set_type(hierarchical);
}};
```

set_symbol_max_length(*numeric_expression*)**;**

DESCRIPTION :

Sets the maximum number of characters for each word (name) in the memory map name. The number of characters for each word can be less or equal with the maximum length.

[CSL Memory Map Command Summary]

EXAMPLE :

Create a memory map page with the name *mpage_0* and set the maximum number of characters for the name of elements to 10.

CSL CODE

```
csl_memory_map_page mpage_0 {
  mpage_0(){
  add_address_range(0, 63);
  set_symbol_max_length(10);
  }
};
csl_memory_map mmap{
  mpage_0 mpage_0;
  mmap(){
set_type(hierarchical);
}
};
```

VERILOG CODE

```
`define MEM_MAP_MAX_LENGTH 10;
```

csl_memory_map memory_map_name;

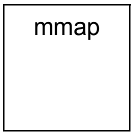
This command declares an object of type memory map, named *memory_map_name*.

[CSL Memory Map Command Summary]

EXAMPLE :

Create a memory_map named *mmap.*

FIGURE 2.13 A memory map named mmap



CSL CODE

```
//AV
//creates a memory map with the name mmap;
csl_memory_map mmap{
   mmap(){
    set_type(hierarchical);
}
};
```

VERILOG CODE

```
//AV
```

`auto_gen_memory_map();`

DESCRIPTION :

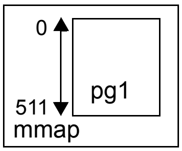
This command is used to generate automatic the memory map for the all memory elements and unit instances.

[CSL Memory Map Command Summary]

EXAMPLE :

Generates automatic a memory map for a memory map page named *pg1.*

FIGURE 2.14



CSL CODE

```
csl_memory_map_page pg1{
  pg1(){
  add_address_range(0, 511);
  }
};
csl_memory_map mmap{
  pg1 pg1;
  mmap(){
  auto_gen_memory_map();
  set_type(hierarchical);
  }
};
```

VERILOG CODE

`set_top_unit(unit_object_name);`

DESCRIPTION :

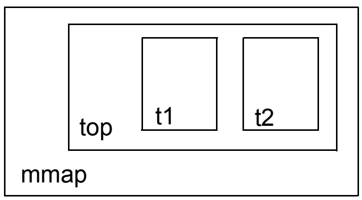
This command is used to set the top unit that has instances of other units.

[CSL Memory Map Command Summary]

EXAMPLE :

In this example we have three units named *top1* , *top2* and *top*. The unit *top* has instances of the *top1* and *top2* units named *t1* and *t2*. In a memory map named mmap is set the top unit using *set_top_unit* method.

FIGURE 2.15



CSL CODE

```
csl_unit top1{
  top1(){}
};
csl_unit top2{
  top2(){}
};
csl_unit top{
  top1 t1;
  top2 t2;
  top(){}
};
csl_memory_map mmap{
  mmap(){
  set_top_unit(top);
  set_type(hierarchical);
  }
};
```

VERILOG CODE

```
object_name.add_to_memory_map([address],[group,access_right]);
```

DESCRIPTION :

This method will add one object called *object_name* to the memory map. Optionaly can be set the address in the memory map where the object will be mapped and the access right for this object. At least one parameter should exist.

TABLE 2.9 Acces Rights

ACCES RIGHTS	Description
access_none	None
access_read	Read
access_write	Write
access_read_write	Read/Write

[CSL Memory Map Command Summary]

EXAMPLE :

Adds a *fifo* named *f1* to a memory map named *mmap*.Fifo will be added to the address 32 and with access write.

CSL CODE

```
csl_fifo f1{
  f1(){
  set_width(32);
  set_depth(128);
  add_to_memory_map(32, SWR, access_write);
  }
};
csl_memory_map mmap{
  mmap(){
 set_type(hierarchical); }
};
```

VERILOG CODE

```
`define DEFAULT_GROUP_user 0
`define DEFAULT_GROUP_swr 1
`define DEFAULT_GROUP_hwr 2
`define DEFAULT_GROUP_test 3
`define DEFAULT_GROUP_driver 4
module f1(push,
          pop,
          full,
          empty,
          data_out,
          data_in,
```

```
        reset_,
        clock,
        valid);
parameter ADDR_WIDTH = 4'd8;
parameter DATA_WIDTH = 6'd32;
input push;
input pop;
input [31:0] data_in;
input reset_;
input clock;
output reg full;
output reg empty;
output reg [31:0] data_out;
output reg valid;
reg [ADDR_WIDTH - 1:0] wr_addr;
reg [ADDR_WIDTH - 1:0] rd_addr;
reg wr_en;
reg rd_en;
assign   full = wr_addr + 1 == rd_addr;
assign   empty = wr_addr == rd_addr;
assign   wr_en = !full && push;
assign   rd_en = !empty && pop;
f1_fifo_memory fifo_memory_instance(.clock(clock),
                                    .data_in(data_in),
                                    .data_out(data_out),
                                    .rd_addr(rd_addr),
                                    .rd_en(rd_en),
                                    .reset_(reset_),
                                    .valid(valid),
                                    .wr_addr(wr_addr),
                                    .wr_en(wr_en));
always @( posedge clock or negedge reset_ )  begin
   if ( ~reset_ )  begin
     rd_addr <= 1'd0;
   end
   else       if ( pop )  begin
       rd_addr <= rd_addr + 1'd1;
     end
 end
 always @( posedge clock or negedge reset_ )  begin
```

```
    if ( ~reset_ )  begin
      wr_addr <= 1'd0;
    end
    else       if ( push )  begin
        wr_addr <= wr_addr + 1'd1;
      end
  end
endmodule
module f1_fifo_memory(clock,
                      reset_,
                      data_in,
                      data_out,
                      valid,
                      wr_addr,
                      rd_addr,
                      wr_en,
                      rd_en);
  parameter ADDR_WIDTH = 4'd8;
  parameter DATA_WIDTH = 6'd32;
  parameter NUM_WORDS = (1'd1 << DATA_WIDTH);
  input clock;
  input reset_;
  input [DATA_WIDTH - 1:0] data_in;
  input [ADDR_WIDTH - 1:0] wr_addr;
  input [ADDR_WIDTH - 1:0] rd_addr;
  input wr_en;
  input rd_en;
  output reg [DATA_WIDTH - 1:0] data_out;
  output reg valid;
  reg [DATA_WIDTH - 1:0] internal_memory[1'd0:NUM_WORDS - 1'd1] ;
always @( posedge clock or negedge reset_ )  begin
    if ( ~reset_ )  begin
      valid <= 1'd1;
    end
    else  begin
      valid <= rd_en;
      data_out <= internal_memory[rd_addr];
      if ( wr_en )  begin
        internal_memory[wr_addr] <= data_in;
      end
```

```
    end
  end
endmodule
```

```
set_type(memory_map_type);
```

DESCRIPTION :

Sets the type for a memory map. The `memory_map_type` can be one of the following:

TABLE 2.10

Memory map type
hierarchical
virtual_with_page_number_and_address
virtual_with_base_address

[CSL Memory Map Command Summary]

EXAMPLE :

In this example is set the type of a memory map *named* mmap wihch contains the instances of two memory map pages named *map0* and *map1*.

CSL CODE

```
csl_memory_map_page mpage_0{
  mpage_0(){
  add_address_range(64,511);
  set_address_increment(2);
  }
};
csl_memory_map_page mpage_1{
  mpage_1(){
  add_address_range(0,63);
  set_address_increment(1);
  }
};
csl_memory_map mmap{
  mpage_0 map0;
  mpage_1 map1;
  mmap(){
  set_data_word_width(32);
  set_type(hierarchical);
  }
};
```

VERILOG CODE

set_prefix(*string*);

DESCRIPTION :

Applies *string* as a prefix to all names in the memory map. Acts as a global prefix to the current scope.

[CSL Memory Map Command Summary]

EXAMPLE :

In a memory map *mmap* which contains a memory map page *mpage* is set the prefix "*mem*".

CSL CODE

```
csl_memory_map_page mpage{
  mpage(){
  add_address_range(0,1023);
  set_address_increment(1);
  }
};
csl_memory_map mmap{
  mpage mpage0;
  mmap(){
  set_data_word_width(32);
  set_prefix("mem");
  set_type(hierarchical);
  }
};
```

VERILOG CODE

```
`define <MMN>_PREFIX name;
```

set_sufix(string);

Description :

Applies *string* as a sufix to all names in the memory map. Acts as a global sufix to the current scope.

[CSL Memory Map Command Summary]

Example :

In a memory map *mmap* which contains a memory map page *mpage* is set the sufix “*mem*”.

CSL code

```
csl_memory_map_page mpage{
  mpage(){
  add_address_range(0,1023);
  set_address_increment(1);
  }
};
csl_memory_map mmap{
  mpage mpage0;
  mmap(){
  set_data_word_width(32);
  set_sufix("mem");
  set_type(hierarchical);
  }
};
```

Verilog code

```
'define <MMN>_SUFIX name;
```

2.2.1 Generated Code

2.2.1.1 Generated C++ Code !!turn this to H2

```
#ifndef __csl_I_<NAME>_VH_
#define __csl_I_<NAME>_VH_

//
// DO NOT EDIT - automatically generated by <toolname>!
//
// --------------------------------------------------------------------
-------
//
// Copyright (c) <year>, <company name>
// All Rights Reserved.
//
// This is UNPUBLISHED PROPRIETARY SOURCE CODE of <company name>;
// the contents of this file may not be disclosed to third parties,
copied or
// duplicated in any form, in whole or in part, without the prior writ-
ten
// permission of <company name>.
//
// RESTRICTED RIGHTS LEGEND:
// Use, duplication or disclosure by the Government is subject to
restrictions
// as set forth in subdivision (c)(1)(ii) of the Rights in Technical
Data
// and Computer Software clause at DFARS 252.227-7013, and/or in simi-
lar or
// successor clauses in the FAR, DOD or NASA FAR Supplement. Unpub-
lished -
// rights reserved under the Copyright Laws of the United States.
//

// generated C++ section
// generated from toolname : <toolname>
// path to tool:           : <path>
// tool version:           : <version>
// time stamp for tool:    : <tool time stamp>
```

```
// generated from filename : <filename>
// source filename:         : <filename>
// source file timestamp:   : <source file time stamp>
// generated file timestamp: <current file time stamp>

// Register register_name
#define register_name_REGISTER_ADDRESS        0x<address>

// value to reset the entire register to

// the following two fields can be defined using the field reset and
set values.
#define register_name_REGISTER_RESET_VAL       0x<reset_value>
#define register_name_REGISTER_SET_VAL         0x<set_value>

// the shift value is equal to the LSB bit position of the field
#define register_name_field_name_SHIFT_AMOUNT   <shift_value>
#define register_name_field_name_MASK           <mask>

// use the following define to set the value of the field
#define register_name_field_name_SET_SHIFT_AND_MASK <mask> << <shift>

// use the following define to get the value of the field
#define register_name_field_name_GET_SHIFT_AND_MASK <mask> >> <shift>

#define register_name_field_name_BITRANGE
<msb_bit_position>:<lsb_bit_position>
#define register_name_field_name_INIT_VAL  0x<field_init_value>
#define register_name_field_name_SET_VAL   0x<field_set_value>
#define memory_map_name_END_ADDRESS          <address>
```

2.2.1.2 Generated Verilog Code

```
#ifndef __csl_I_<NAME>_VH_
#define __csl_I_<NAME>_VH_

//
// Generated by <toolname>
// DO NOT MODIFY
```

```
// -----------------------------------------------------------------
-------
//
// Copyright (c) <year>, <company name>
// All Rights Reserved.
//
// This is UNPUBLISHED PROPRIETARY SOURCE CODE of <company name>;
// the contents of this file may not be disclosed to third parties,
copied or
// duplicated in any form, in whole or in part, without the prior writ-
ten
// permission of <company name>.
//
// RESTRICTED RIGHTS LEGEND:
// Use, duplication or disclosure by the Government is subject to
restrictions
// as set forth in subdivision (c)(1)(ii) of the Rights in Technical
Data
// and Computer Software clause at DFARS 252.227-7013, and/or in simi-
lar or
// successor clauses in the FAR, DOD or NASA FAR Supplement. Unpub-
lished -
// rights reserved under the Copyright Laws of the United States.
//

// Generated verilog section
// generated from toolname : <toolname>
// path to tool:             <path>
// tool version:             <version>
// time stamp for tool:      <tool time stamp>
// generated from filename:  <filename>
// source filename:          <filename>
// source file timestamp:    <source file time stamp>
// generated file timestamp: <current file time stamp>

#define <name>_WIDTH16
#define <name>_RANGE15:0
#define <name>_ADDR0

// Register <reg_name>
```

```
#define <reg_name>_WIDTH8
#define <reg_name>_RANGE7:0
#define <reg_name> 32'h0
#define <reg_name>_RESET_NUM8'bxxxxxxxx
#define <reg_name>_INIT_NUM8'h0

//fields belonging to the above register
// there are n fields which in total width can equal but not exceed the
width of the above //register definition
#define <reg_name>_field_name_WIDTH<field_width>
#define <reg_name>_field_name_RANGE<field_range>
#define <reg_name>_field_name_RW<r=2, rw=3> // 10 and 11
#define <reg_name>_field_name_NUM  <field_width>'h<value>  // devulat
for
//<value> is 0

Example:

// Register register_name
#define register_name_ADDRESS 32'h<address>
#define register_name_RESET_VALUE 2'b<value>
#define register_name_SET_VALUE 3'h<value>
#define register_name_BITRANGE [<msb_bit_position>:<lsb_bit_position>]
#define register_name_REGISTER_WIDTH <width>
#define register_name_field_name_BITRANGE
#define register_name_field_name_field_WIDTH 1
#define register_name_field_name_ATTR
#define register_name_field_name_DEFAULT 1'h0
#define BASE_ADDRESS_<module_name>              <address>


class register_name : public register {
  register_name_ADDRESS 32'h<address>
  register_name_RESET_VALUE 2'b<value>
  register_name_SET_VALUE 3'h<value>
  register_name_BITRANGE [<msb_bit_position>:<lsb_bit_position>]
 field_name register_name_REGISTER_WIDTH <width>
  register_name_field_name_BITRANGE
  register_name_field_name_field_WIDTH 1
  register_name_field_name_ATTR
```

```
  register_name_field_name_DEFAULT 1'h0
  BASE_ADDRESS_<module_name>              <address>
}


#endif __csl_I_<NAME>_VH_
```

2.2.2 Generated code

2.2.2.1 Generated C++ Code

```
#ifndef_csl_1_<NAME>_VH_
#define_csl_1_<NAME>_VH_
// DO NOT EDIT =automatically generated by <toolname>!
//
// -------------------------------------------------------------------
---
//
//Copyright (c) <year><company name>
// All Rights Reserved
//
//This is UNPUBLISHED PROPRIETARY SOURCE CODE of <company name>;
//the contents of this file may not be disclosed to third parties, cop-
ied or duplicated in any form, in whole or in part, without the prior
written permission of <company name>
//
//RESTRICTED RIGHTS LEGEND:
// Use, dulpication or disclosure by the Government is subject to
restrictions as se
//forth in subdivision (c)(1)(ii) of the Rights in Technical Data and
Computer Soft
//ware clause at DFARS 252.227-7013, and/or in similar or succesor
clauses in the
//FAR, DOD or NASA FAR Supplement. Unpublished  rights reserved under
the
//Copyright Laws of the United States
//
#Generated C++ section
#toolname: <toolname>
#path to tool: <path>
```

```
#tool version: <version>
#time stamp for tool: <tool time stamp>
#generated from filename: <filename>
##file timestamp <source file timestamp>
#generated timestamp <current file time stamp>
//Register STATUS_0
#define STATUS_0            .0xc
#define STATUS_0_RESET_NUM 0x0
#define STATUS_0_BSY_SHIFT 7
#define STATUS_0_BSY_FIELD (0x1<<STATUS_0_BSY_SHIFT)
#define STATUS_0_BSY_RANGE 7:7
#define STATUS_0_BSY_DEFAULT 0x0
#define STATUS_0_DRDY_SHIFT 6
#define STATUS_0_DRDY_FIELD (0x1<<STATUS_0_DRDY_SHIFT)
#define STATUS_0_DRDY_RANGE 6:6
#define STATUS_0_DRDY_DEFAULT 0x0
#define STATUS_0_DRQ_SHIFT 3
#define STATUS_0_DRQ_FIELD (0x1<<STATUS_0_DRQ_SHIFT)
#define STATUS_0_DRQ_RANGE 3:3
#define STATUS_0_DRQ_DEFAULT 0x0
#define STATUS_0_ERR_SHIFT 0
#define STATUS_0_ERR_FIELD (0x1<<STATUS_0_ERR_SHIFT)
#define STATUS_0_ERR_RANGE 0:0
#define STATUS_0_ERR_DEFAULT 0x0
#define CEATA0_LAST_REG STATUS_0//0x000d
```

2.2.2.2 Generated Verilog Code

```
#ifndef_csl_|_<NAME>_VH_
#define_csl_|_<NAME>_VH
//DO  NOT EDIT -automatically generated by <toolname>!
// -----------------------------------------------------------------
--
///
//Copyright (c) <year><company name>
//All Rights Reserved
//
//this is UNPUBLISHED PROPRIETARY SOURCE CODE pf <company name>;
//the contents of this file may not be disclosed to third parties, cop-
ied or duplicated in any form, in whole or in part, without the prior
written permission of <company name>
```

```
//RESTRICTED RIGHTS LEGEND:
// Use, dulpication or disclosure by the Government is subject to restrictions as se
//forth in subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Soft
//ware clause at DFARS 252.227-7013, and/or in similar or succesor clauses in the
//FAR, DOD or NASA FAR Supplement. Unpublished  rights reserved under the
//Copyright Laws of the United States
#Generated verilog section
#toolname : <toolname>
#path to tool <path>
#tool version : <version>
#time stamp for tool: <tool time stamp>
#generated from filename : <filename>
#file timestamp <source file time stamp>
#generated timestamp <current file time stamp>
#define <name>_WIDTH16
#define <name>_RANGE 15:0
#define <name>_ADDR0
//register <reg_name>_0
#define <reg_name>_0_WIDTH8
#define <reg_name>_0_RANGE 7:0
#define <reg_name>_0 32'h0
#define <reg_name>_0_RESET_NUM8'bxxxxxxxx
#define <reg_name>_0_INIT_NUM8'h0
//fields belonging to the above register
//there are n fields which in total can equal ubt not exceeded the width of the above register definition
#define <reg_name>_0_field_name_WIDTH<field_width>
#define <reg_name>_0_field_name_RANGE<field_range>
#define <reg_name>_0_field_name_RW<r=2,rw=3>//10 and 11
#define <reg_name>_0_field_name_NUM <field_width>'h<value>//devulat for <value>
```

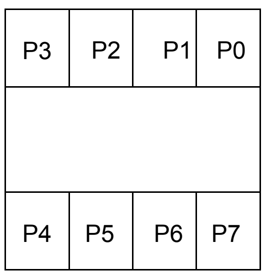
Example:

```
//register register_name_0
#define register_name_032'h5
#define register_name_0_RESET_NUM2'bxx
#define register_name_0_INIT_NUM3'h0
```

```
#define register_name_0_RANGE2:1
#define register_name_0_WIDTH2
#define register_name_0_field_name_RANGE2
#define register_name_0_field_name_WIDTH1
#define register_name_0_field_name_RW3
#define register_name_0_field_name_DEFAULT'h0
#define register_name_0_field_name_RANGE1
#define register_name_0_field_name_WIDTH1
#define register_name_0_field_name_RW3
#define register_name_0_field_name_DEFAULT1'h0
#deine BASE_ADDRESS_MODULE32'h00000000
#endif_csl_I_<NAME>_VH_
```

NOTE:<Move this to commands examples>

FIGURE 2.16

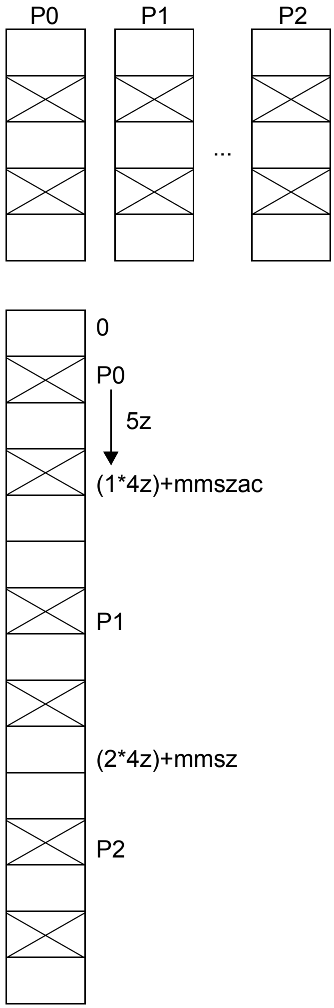



```
csl_memory_map mmn;
csl_unit p[0-7];
p[0-7].set_range(mmn, 0, 29);
set_address(mmn, \1.getadde_size()*\2);
```

- \1=p[0-7]
- \2=[0-7]

```
default address= lastobject.baseaddress()+lastob-
ject.addr_size()+mmn.inc_amounts
p[0-7].add_to_memory_map(mmn);
user next address which is equal to mmn.inc_amounts
```

FIGURE 2.17 Individual processors memory spaces and the combined memory map shrink figure



each processor address space starts at an offset
</Move this to commands examples>
<ADD>
move this ADD to sw components
Consumer Electronic Chips

FIGURE 2.18

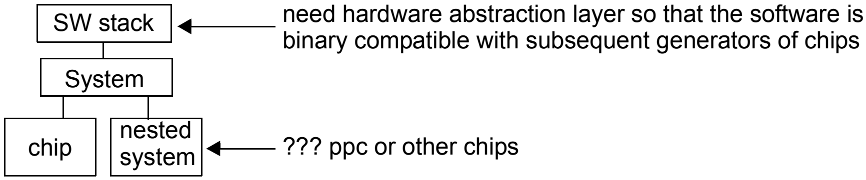
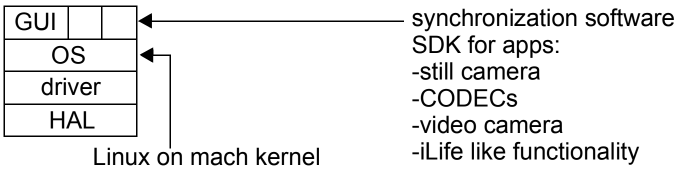


FIGURE 2.19



</ADD>

<ADD>

Code generation - move to code gen doc
!also move the generated code abvoe

<memory_map_class_name>::enum<register_name>_<field_name_in_register>_<enum_name_in_field>

All letters are capitilized except the enum.

Register output abreviations

mme = memory map element
fld = field in a register
enum = enumerated type value for a given field

C/C++ classes will be generated with a prefix letter "C".
C/C++ enumerated types will be generated with a prefix letter "c" (i.e. enum cenum<enumerated_type_name> {...}).

Verilog code will be generated with a prefix letter "v"

Verilog defines which are equivalent to the C/C++ enumerated will be generated with a prefix letter "cv" (i.e. `define venum<enumerated_type_name> <value>).

```
enumerated types should have an illegal field which can be returned
from C/C++ switch default cass and from Verilog cas statement
default cases. The illegal field can be "caught" by the "down-
stream" logic and can flag problems with switch and case statement
selector inputs.
```

</ADD>

Virtual Memory

SW address map is global.
HW address map is local.
Upper bits are the page ID.
Upper bits map to a unit ID.

TABLE 2.11 Virtual memory table

upper bits	global	local
0	m	0
0	n	(n-m)
1	p	0
1	q	(q-p)
2	b	0
2	c	(c-b)
3	d	0
3	e	e-d

FIGURE 2.20

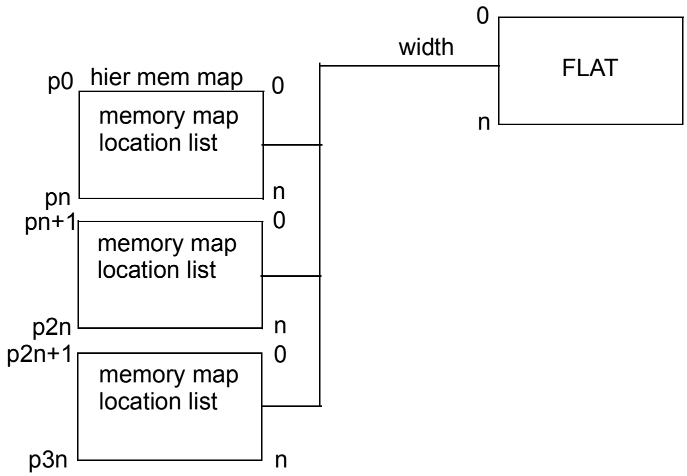


TABLE 2.12

	global	local
flat	x,y	x, y
	psa+m, psa+n address	m, n
hier	x, y	x, y
v m	sa=(pno<<amount) ea=(pno<<amount)	1.x 1.y x,y

VM base	VMPN in Addr
000000	0000000 = (0<<20) | 0
100000	0100000 = (1<<20) | 0000
200000	0200000 = (2<<20) | 0000
20 bits global	28 bit global

<ADDED_2007.05.12>

```
Different methods used for  programming chip registers

Chips contain registers which need to be configured with values
```

The register values also need to be read out to a different unit ont he chip or outside of the chip.

CSL provides a way to write a set of registers on a chip using one or 4 different
physical bus/network topologies. The

All buses contain essentially the same set of commands.

addr (address)

data

v (valid)

cmd (command)

All buses/networks are connected to the controller and all leaf level units.

In the caes of the tree network there may be intermediate nodes which are used to
gather information from a cluster of units and for timing reasons.

1. In band SOC bus

2a. Out of band network tree

2b. Out of band network Ring

3. In band pipeline

==

1. In band SOC bus

Each bus master waits for a slot on the bus and then sends a bus command to
another unit on the bus. All units "listen" to the bus for bus commands
addressed to the unit.

2a. Out of band network tree

The Out of band network tree has both a send and a receive network

The send network contains the following of signals:

addr - data

data - address

v - valid

cmd - command

The send network is used to send data and commands to the leaf level units.

```
The leaf level units execute the commands and if requested send a reply
via the
reply tree to the controller.
The controller broadcasts messages to all units which match the uid in
the message and then
execute the command.

2b. Out of band network Ring
The Out of band network ring connects all units in a ring topology.
The ring contains the following of signals:
addr - data
data - address
v    - valid
cmd  - command

3. In band pipeline
Each pipestage can contain one or more registers which can be read/
written via packets sent down the
command pipeline. The command pipelne packets contain the following
signals.
addr - data
data - address
v    - valid
cmd  - command
When the address in the pipestage address signal matches an address in
the pipestage and the valid is
'1' then the command is exectued and a register is either read or writ-
ten.

========================================================================
==========

// note that in the memory map b elow we do not set the data word width
or the address word
width. The clsc will determine the address word width based on the
address range (start and end
addresses) for the memory map.

csl_memory_map mem_map {
  csl_memory_map_page unit_a;
  mem_map () {
```

```
    set_type(VM_WITH_ADDRESS);
    unit_a.set_range(0, 65767);
  }
}

csl_enum bus_cmd {
  BUS_CMD_RD,
  BUS_CMD_WR,
  BUS_CMD_PING,
  BUS_CMD_NOP
};

// create a bus with signal names that match the pin names on the reg-
isters that the bus
// is logically connected to. There are intermediate units whiuch the
bus is connected to
// for timing and distribution reasons. The units that the bus is con-
nected to have a bus
// interface unit (BIU) that the bus is connected to. The BIU detects
commands that are
// intended for the unit and converts the commands into local control,
address, and data
//
//
//
//

csl_interface reply_bus {
  csl_port data(input,32),
          addr(input, mem_map.get_address_word_width()), // 9 bits
since log2(512) = 9
          v   (input  ) ;
};

csl_interface cmd_bus : reply_bus {
  csl port cmd(input,2);
  ifc(){
    cmd.add_enum(bus_cmd);
  }
};
```

```
csl_unit controller {
  cmd_bus bus_out;
  reply_bus bus_in;
  controller(){
    bus_out.reverse();
  }
};


csl_register_group unit_a_rg {
  csl_register r[[0-31]](32); // create 32 32-bit registers

  unit_a_rg() {

  }
};

csl_unit a{
  cmd_bus   bus_in;
  reply_bus bus_out;
  unit_a_rg unit_a_rg0;
  int unit_a_mem_map_base_addr;

  a() {
    unit_a_mem_map_base_addr = 2048;
    bus_out.reverse();
    mem_map.unit_a.add(unit_a_rg0, "unit_regs",
unit_a_mem_map_base_addr);
    unit_a_rg0.use_biu_to_write();

// unit_a_rg0 has the same interface as bus_in so they can be connected
// each register has a set of pins that match the signal names and
directions
// in the bus.
// however the bus_in and the bus_out are not directly connected to the
registers
// instead intermediate logic is created to write the registers.
//
// The cslc detects that each register in the register group unit_a_rg0
are in the memory
```

```
// map. Since all registers in the memmory map they need to be con-
nected to the the unit_a
// BIU (bus interface unit ) which listens to the bus as described
above and generates the
// write enable (wr_en) signals for each individual register.
//
// The interface bus_out is no directly connected to the register out-
puts. Instead the register
// outputs are connected to a mux and the bus_in_addr selects the reg-
ister to send back to the
// controller which sent the read command to unit_a.
//
// If not all registers are in the memory map generate a compiler
error.

    unit_a_rg0.connect(bus_in);
    bus_in.connect(unit_a_rg0);

    csl_signal a_en =
    reg_0.d = data;
    mem_map.set_unit_address_signal(a,addr);
  }
};


csl_unit top{
  a a0;
  controller cntl0;
  top(){
    a0.set_instance_id(3);

  }
};
```

OLD CSL CODE

```
csl_memory_map mem_map;

csl_enum bus_cmd {
  BUS_CMD_RD,
  BUS_CMD_WR,
```

```
  BUS_CMD_PING,
  BUS_CMD_NOP
};

csl_interface reply_bus {
  csl_port data(input,32),
           addr(input,5) ,
           v(input)      ;
};

csl_interface cmd_bus : reply_bus {
  csl port cmd(input,2);
  ifc(){
    cmd.add_enum(bus_cmd);
  }
};

csl_unit controller {
  cmd_bus bus_out;
  reply_bus bus_in;
  controller(){
    bus_out.reverse();
  }
};

csl_unit a{
  cmd_bus   bus_in;
  reply_bus bus_out;
  csl_register reg_0(32);
  a(){
   bus_out.reverse();
   reg_0.
   csl_signal a_en =
   reg_0.d = data;
  }
};
mem_map.add_logic(object_wr_en, address)
mem_map.add_object(a.reg_0)
mem_map.set_unit_address_signal(a,addr);
```

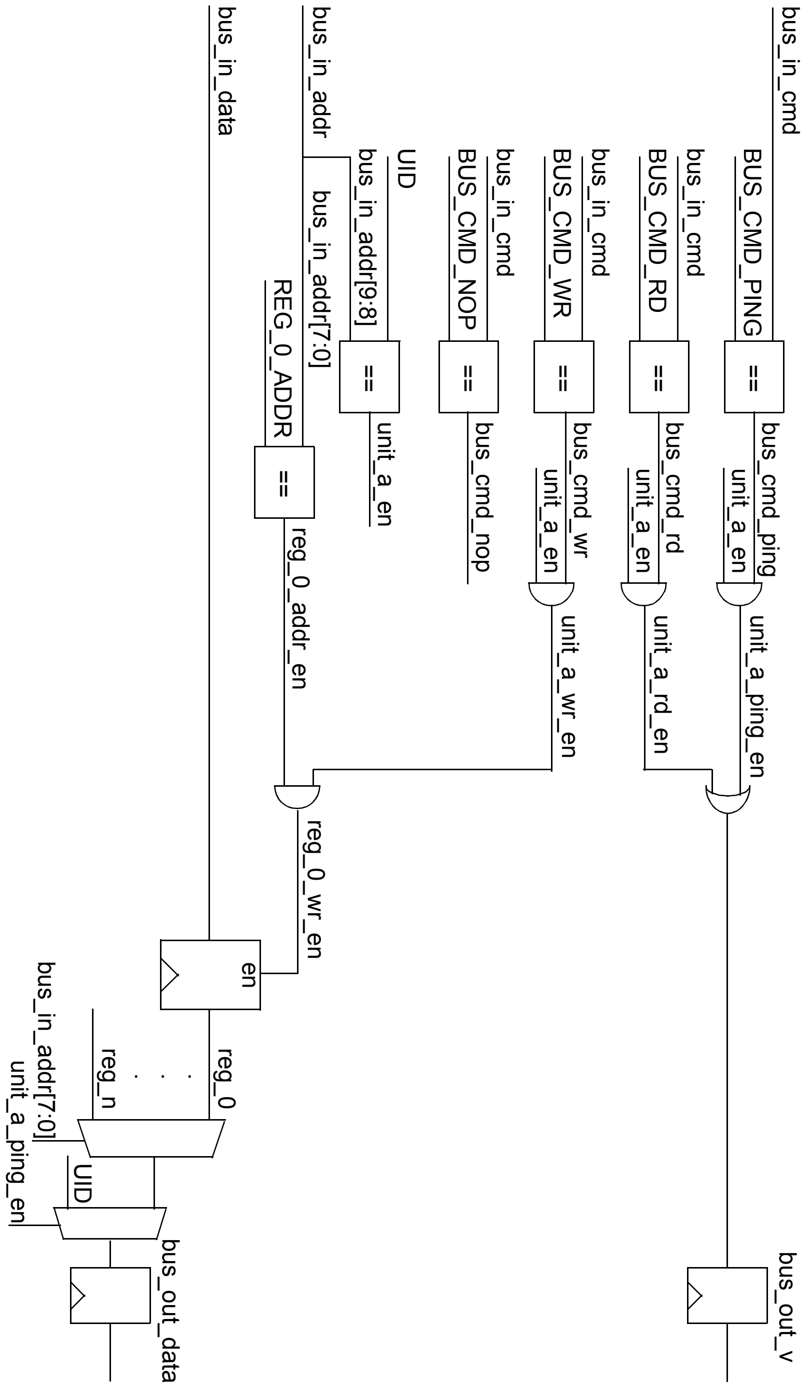
```
csl_unit top{
  a a0;
  top(){
    a0.set_instance_id();
  }
};
```

VERILOG CODE

```
`define BUS_CMD_RD     0
`define BUS_CMD_WR     1
`define BUS_CMD_PING   2
`define BUS_CMD_NOP    3
`define UID            3 //unit id
`define REG_0_ADDR   128 //reg_0 address

module a(bus_in_data,
         bus_in_addr,
         bus_in_cmd ,
         clk,
         bus_out_data,
         bus_out_v
         );

 input [31:0] bus_in_data;
 input [9:0]  bus_in_addr;
 input [1:0]  bus_in_cmd;
 input clk;

 output bus_out_v;
 reg bus_out_v;
 output [31:0] bus_out_data;
 reg [31:0] bus_out_data;

 //local signals
 reg [31:0] reg_0;

 wire bus_cmd_rd   = (`BUS_CMD_RD   == bus_in_cmd)  ;
 wire bus_cmd_wr   = (`BUS_CMD_WR   == bus_in_cmd)  ;
 wire bus_cmd_ping = (`BUS_CMD_PING == bus_in_cmd);
 wire bus_cmd_nop  = (`BUS_CMD_NOP  == bus_in_cmd) ;
```

```
 wire unit_a_en      = bus_in_addr[9:8] == `UID;
 wire unit_a_wr_en   = unit_a_en && bus_cmd_wr;
 wire unit_a_rd_en   = unit_a_en && bus_cmd_rd;
 wire unit_a_ping_en= unit_a_en && bus_cmd_ping;

 wire reg_0_addr_en = bus_in_addr[7:0] == `REG_0_ADDR;
 wire reg_0_wr_en   = unit_a_wr_en && reg_0_addr_en;

 always @(posedge clk) begin
   if(reg_0_wr_en) begin
     reg_0 <= bus_in_data;
   end
 end

always @(posedge clk) begin
    bus_out_v <= unit_a_rd_en & unit_a_ping_en;
end


always @(posedge clk) begin
    if(unit_a_rd_en) begin
  case (bus_in_addr)
  `REG_0_ADDR: bus_out_data <= reg_0;
  endcase
  end
  else if(unit_a_ping_en) begin
      bus_out_data = `UID;
  end
end
endmodule
```

FIGURE 2.21 Bus Interface Unit Command Decoder


bus_in_cmd
BUS_CMD_PING
bus_in_cmd
BUS_CMD_RD
bus_in_cmd
BUS_CMD_WR
bus_in_cmd
BUS_CMD_NOP
UID
bus_in_addr[9:8]
bus_in_addr
bus_in_addr[7:0]
REG_0_ADDR
bus_in_data
==
==
==
==
==
==
bus_cmd_ping
unit_a_en
bus_cmd_rd
unit_a_en
bus_cmd_wr
unit_a_en
bus_cmd_nop
unit_a_en
reg_0_addr_en
unit_a_ping_en
unit_a_rd_en
unit_a_wr_en
reg_0_wr_en
en
reg_0
. . .
reg_n
bus_in_addr[7:0]
UID
unit_a_ping_en
bus_out_data
bus_out_v

</ADDED_2007.05.12>

<ADDED ON 2007.05.16>

NOTE:UPDATE COMMAND SUMMARY ACCORDING TO THIS

Note: There should be 8 examples from :

TABLE 2.13

Type	User defined mem map	automatic mem map
hierarchical	x	x
virtual with page number and address	x	x
virtual with base address	x	x

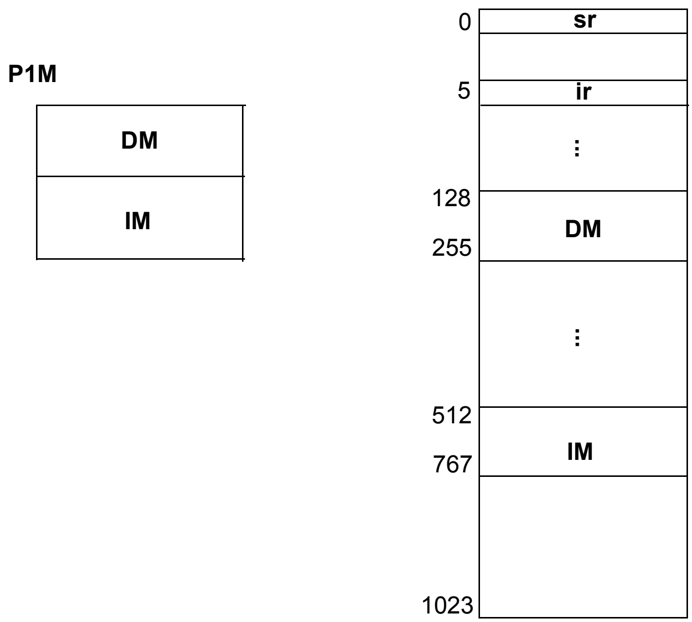
User defined example:

```
csl_unit processor {
    ...
};

csl_unit cluster {
  processor p[[0-7]];
};

csl_unit chip {
  cluster c[[0-7]];
};

csl_memory_map_page mproc {
  mproc(){
    set_unit(processor);
  }
};

csl_memory_map_page mcluster {
  mproc mp[[0-7]](p[[0-7]]); //user specified
  mcluster(){
    set_unit(cluster);
  }
};

csl_memory_map_page mchip {
  mcluster mc[0-7]](c[[0-7]]); //user specified
  mchip(){
    set_unit(chip);
  }
};

csl_memory_map mmap {
  mchip mchip;
  mmap(){
  set_type(hierarchical);
  }
}
```

Automatic example:

```
csl_unit processor {
   ...
};

csl_unit cluster {
  processor p[[0-7]];
};

csl_unit chip {
  cluster c[[0-7]];
};

csl_memory_map_page mproc {
  mproc(){
    set_unit(processor);
  }
};

csl_memory_map mmap {
   mmap(){
  set_top_unit(chip);
  set_type(hierarchical);
  use_instance_decl_order();
  }
}

//This generates the same code as the user defined version above
```

EXAMPLE :




```
csl_register sr;
csl_register ir;

csl_memory im(16,128);
csl_memory dm(16,256);

csl_unit p{
  im im();
  dm dm();

  sr sr();
  ir ir();
  p(){
     sr.add_to_mem_map();
     ir.add_to_mem_map(5);         // insert at address 5
     dm.add_to_mem_map(128);       // insert at address 128
```

```
      im.add_to_mem_map(512);      // insert at address 512
    }

  csl_memory_page mp{
    mp(){
      set_unit(p);
    }

  csl_memory_map mm{
    mp mp;
    mm(){
      set_top_unit(chip);
      set_type(hierarchical);
      autogen_mem_map;
    }
  }

  csl_unit cl{
    p p[[0-7]];
  }


  csl_unit chip{
    cl cl[[0-7]];
    chip(){
    }
  }
```

Generated header file:

```
#define sr             0x000
#define ir             0x005
#define mp_start_addr  0x000
#define mp_end_addr    0x3FF
#define dm_start_addr  0x080
#define dm_end_addr    0x0FF
#define im_start_addr  0x200
#define im_end_addr    0x2FF
```

When generating the memory map acces rights and visibility will be specified as parameters to generate only those defines that correspond to that specific options.

The adaptor needs to know how to connect the pins objects in the memory map to the network which will read/write the objects in the memory map.
Flat memory will need - data, address, command (W/R) and valid.
All units will listen to the address bus and they will need an address range checker (optional).

For hierarchical memory maps there will be a tree of enable signals to select the unit .
ex: chip enable + cluster enable + processor enable

Virtual with unit ID and address
The upper bits of the address bus will be used to identify the unit (unit ID), the lower bits will be used to address local memory in the selected unit.

Virtual memory with base address ?

0

1

2