CHAPTER 1 CSL Register File

TABLE 1.1 Chapter Overview

1.1 CSL Register File Overview
1.2 CSL Register File Concepts
1.3 CSL Register File Command summary (FIX)
1.4 CSL Register File Commands
1.5 CSL Register File Examples
1.6 CSL Register File Checker

1.1 CSL Register File Overview

CSL Register File Specification Description

Register files are declared using the CSL memory map specification and the CSL register file constructs.The CSL memory map specification is used to create named registers_fields and fields inside of the register file. All memory map operations are supported inside of the register file. The register file's registers can be connected to the inputs and the outputs of the register file. Fields within the registers can also be connected to the inputs or outputs of the register file.

All CSL register operations are supported inside of the register file including clear, set, enable, and soft reset. We use the term register_field to refer to either a register or field inside of a register. Note that the group operation can group registers and fields within registers together so that common operations such as clear or set can be performed on the registers. Read and write operations can trigger events. Read operations can generate valid bits. The register file decodes an address and if the wr_en is set then writes a value to a register. The argument all expands to all registers or words in the register file.

Register files are either instantiated inside of an RTL module using CSL commands or the register file is a stand alone module which we connect to the design using the CSL interconnect commands Typical Register File configuration options:

- Single read port

- Single write port
- Multiple read ports
- Multiple write ports
- Connect individual registers to an output
- read/write registers from more than one source
- num_rd_ports
- num_wr_ports

1.1.1 Abreviations description

TABLE 1.2 Register file abreviations used to registers/fields in diagrams

ar	address range
ext	the register is external to the register file
o	output - the register/field is an output of the block
rn	register name
fn	field name
v	valid - generate a valid bit
ws	word size
nw	number of words

1.2 CSL Register File Concepts

A register file(**rf**) block is used to store values in registers. The registers are addressable by the read and write address lines. The read enable signal (rd_en) is used to access the memory array and return the contents of the addressed word. The register file contains a memory array. The memory array can be constructed from flip flops or an SRAM array. The choice is based on the size of the memory array and the available technology options.

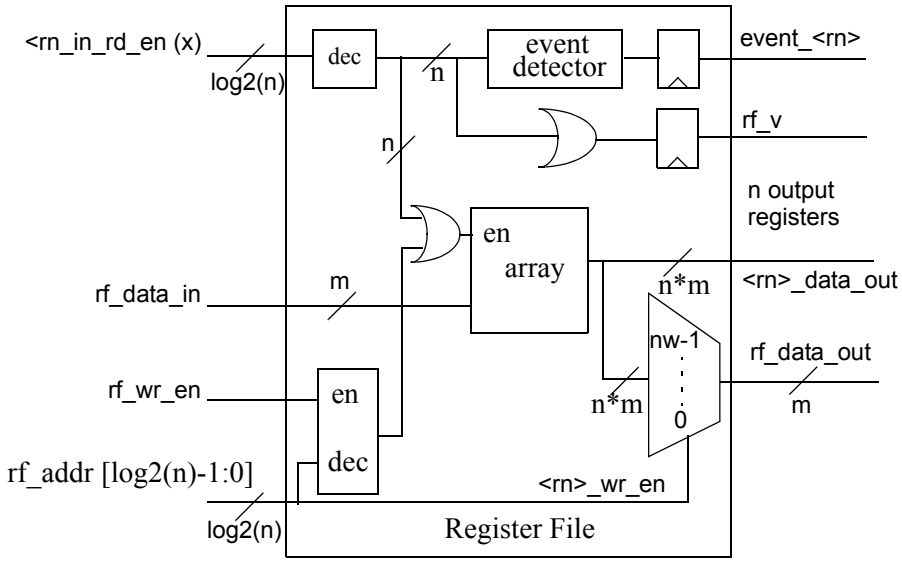
TABLE 1.3

implementation type	valid required	rd_en required
sram	0	1
sram	1	1
ff	0	0
ff	1	1

1.2.1 Address decoders

The register file contains read and write address decoders. The decoded output of the write address decoder is anded (qualified) with the write enable signal. The decoded read address output may also be qualified with a read enable signal. But this is not necessary. An event detector detects when a certain register or range of registers has been read or written.

FIGURE 1.1 Register file architecture



1.2.2 Register File clock inputs

The register file is a clocked device. It has a clock input. The clock input does not need to be specified in the CSL register file specification if the register file is instantiated in a module with only one clock. The module's clock is automatically connected to the register file in this case. If there are multiple clocks in the module in which the register file is instantiated then the CSL clock command has to be used to specify the clock name which is connected to the register file.

1.2.3 Register file address space

The Register file address space (i.e. starting and ending address) is defined with CSL memory map operations. Note that the register file memory map can have discontinuities or gaps in the address space. Named registers or registers with fields can be defined using CSL register construction.

1.2.4 Register files for processor architectures

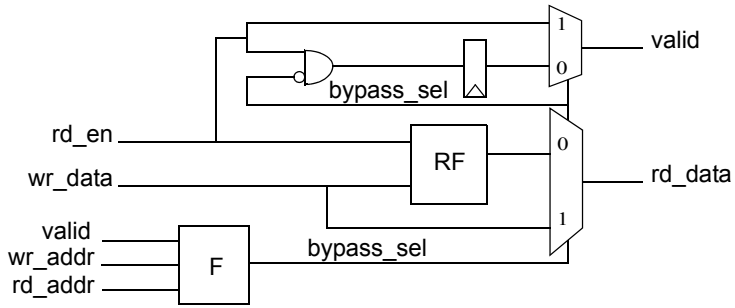
Register files for processor architectures have special requirements.

- valid bits
- operand bypassing

1.2.5 Register Bypassing

Some implementations of register files are optimized so that the read request bypasses the register file to save one clock cycle if the register being read has the same address as the value being written back into the register file.

FIGURE 1.2



The equation that implements the bypass_sel logic is

```
bypass_sel = (wr_addr == rd_addr) ? ~valid : 1'b0;
```

If the previous cycle did not have a valid read and the same address is being used to write and read bypass logic is designed to forward writes to the output of the of register file. If there is valid data on the output then the bypass is dsabled. A truth table for the F block is given in the next table :

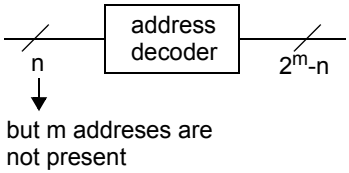
TABLE 1.4 Truth table for F

rd_addr == wr_addr	valid	bypass_sel
0	0	0
1	0	1
0	1	0
1	1	0

```
<register_file_name>.bypass( ) ;
```

The bypass function creates a bypass which corrects the write bus and the register file output to the read output.

FIGURE 1.3 Incomplete address space



- n is the width of the bus
- m are the number of unused addresses in the address spaces

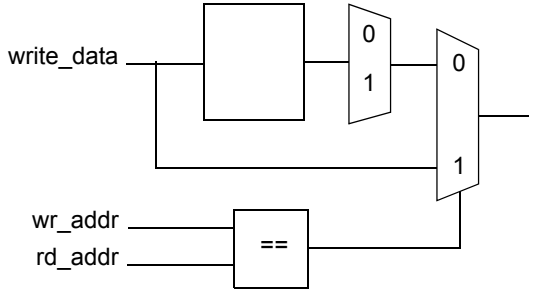
1.2.6 Register File addressing

base address + register offset
If the base address is 0x32000000 and the address range is 0-63 then the mask for the address range is 0x3200_0000. The address range is 0x3200_0000 to 0x3200_0063. If example addresses 5 -10 are included in the address range then 4 is missing.
6 bits are used as the offset into the register file
A sparse address range may be specified : 0-3 and 10-5.

1.2.7 Read address bypass logic

Writes to the RF during cycle n and reads from the same address during cycle n+1 will result in the write data being forwarded/bypassed.

FIGURE 1.4 Read address bypass logic



1.2.8 Addr_collision Detection logic

Register files with more than one write port will detect multiple writes to the same address location during the same cycle. This condition generates an error.

A register can be set to a constant value by setting the const attribute to a value.

1.2.9 Connecting register file inputs and outputs to registers and fields

Specifying that a register or field is connected to an individual input will not disconnect that register from data_in. The register can still be written using the global register file address, write enable, and data_in signals. The default write action uses the global write signals. There are multiplexers connected to each of the register/fields which override the data_in and write enable signals for each register when the signal <register_field_name>_wr_en is asserted. <register_field_name>_wr_en is the mux select for the inputs to the register/field.

1.2.10 Register file decoder

mask is used to extract the address offset

```
wire [3:0] mask = 4'b1111;
wire [3:0] addr = addr_in & mask;
wire [15:0] decoder = 1 << addr;
```

1.2.11 Address options

The first address for the register file can be specified to enable read and writes.

- Read address bypass logic

During cycle n writes to the RF and during cycle n+1 reads from the same address will result in the write data being forwarded.

- Address collision detection

Multiple writes to the same address location during the same cycle will generate an error.

- Constant value in adress register

The offset_mask is used to get only the address offset.

```
wire [3:0] addr = addr_in & mask;
wire [15:0] decoder = 1 << addr;
```

1.2.12 Register File interrupts

There is an option for a bad address checker which sets an error bit, captures the bad address in a register and generates an interrupt.

1.2.13 Register File flags specifying registers/fields

TABLE 1.5 Register file abreviations used to registers/fields in diagrams

v	valid - generate a valid bit at the register field
e	event - generate an event when the register field is written
x	external - the register field data is stored external to the block
n	normal - the register field is inside of the RF

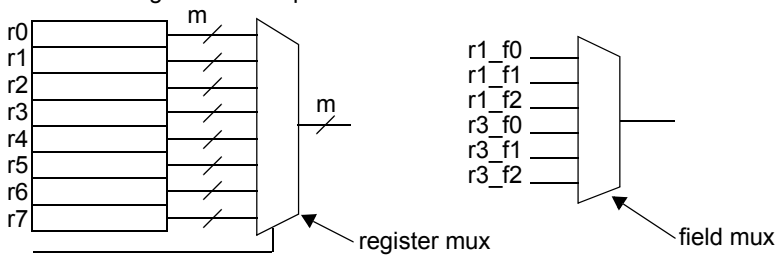
1.2.14 Accesing the register file in the global memory map

The register file is accessed with the base address + offset .
If the base address is 0x32000000 and the address range is 0-63 than the mask for the address is 0x3200_0000.
The address range in the global map is 0x3200_0000 to 0x3200_0063.
6 bits are used as the address offset into the register file.

1.2.15 Register files are constructed hierarchically using elements

Individual fields in registers in a register file may be accessed with a mux just as the individual registers in a register file are accessed with a mux.

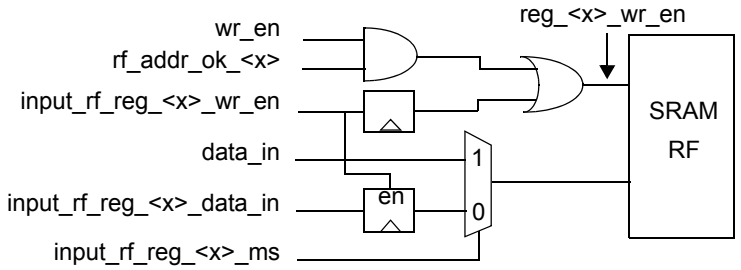
FIGURE 1.5 Register File output mux



1.2.16 Reg file Inputs

A register in an RF can be written using either the data_in and the wr_addr, wr_en or a special input which is tied to a specific register along with a special wr_en.

FIGURE 1.6 Wrapper around SRAM based r.f. to shadow external registers

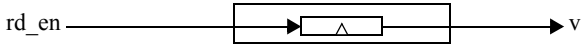


1.2.17 Output valid bits

The register files have an output valid which is the delayed version of the enable read bit. The delay is equal to the delay of the read request of the output. If the read is a bypass then the valid bit needs to be bypassed. Else if the read is the output of the register file then the delay is equal to the normal delay through the register file.

1.2.18 Regfile file valid bit

FIGURE 1.7 Register File valid bit



The valid bit (v) is the pipelined rd_en which corresponds to the data which will be read of the register file due to the read enable. If the rd_en is only associated with the register file then the read enable does not need to be qualified with the “or” of the read address decoder’s outputs.

1.2.19 Register file dataflow architecture

Valid transactions have a valid bit associated with them in the same pipestage: “data announces its arrival to the next pipestage”.
Read transactions can have a valid bit associated with the output data . The valid bit is ligned with the data in the same pipeline output data announces its arrival to the next pipestage.
Individual registers can be read from a register file by connecting the register to the output of the register file.

FIGURE 1.8 Dataflow register file unqualified valid bit

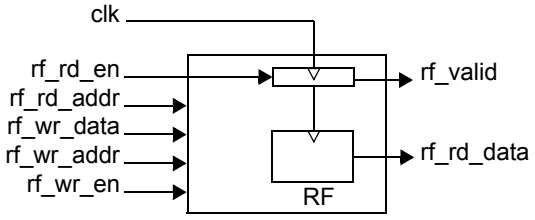
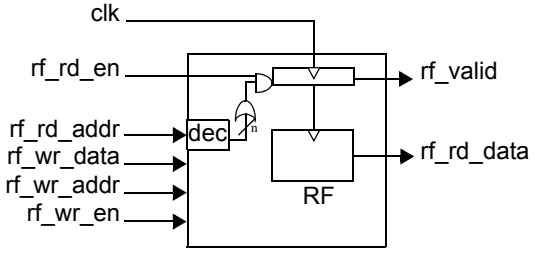


FIGURE 1.9 Dataflow register file unqualified valid bit

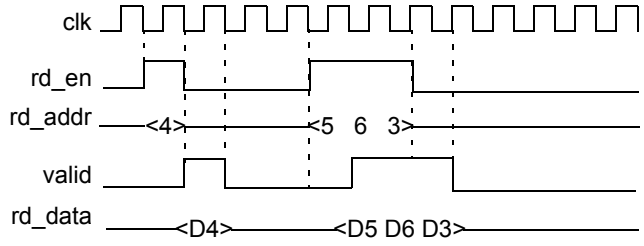


1.2.20 Valid bit generation

A valid bit may be optionally added to the register file. The register file will generate a valid bit whenever there is a read operation (i.e. rd_en is asserted) . The valid bit may be qualified. The valid bit may be qualified with the or of the read address decoder outputs to check the address range.

1.2.21 Dataflow register file read logic

FIGURE 1.10

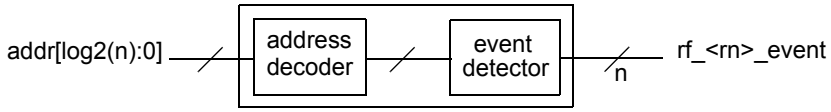


1.2.22 Register Files event detectors

- n is the width of the bus
- m is the number of unused addresses in the addresses spaces

Note: the verilog implementation of the register file address decoder is 1 << addr

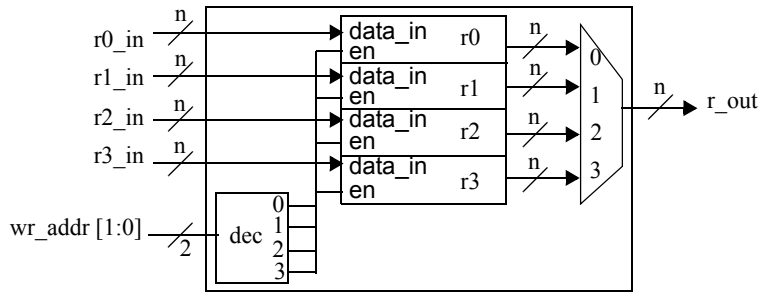
FIGURE 1.11 Register File with an event detector



1.2.23 Connect inputs to individual registers

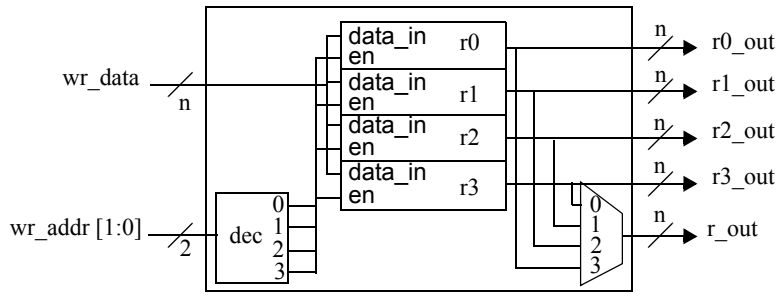
Individual inputs may be conected directly to registers in the register file.

FIGURE 1.12 Connect individual inputs to all registers



1.2.24 Connect individual registers to outputs

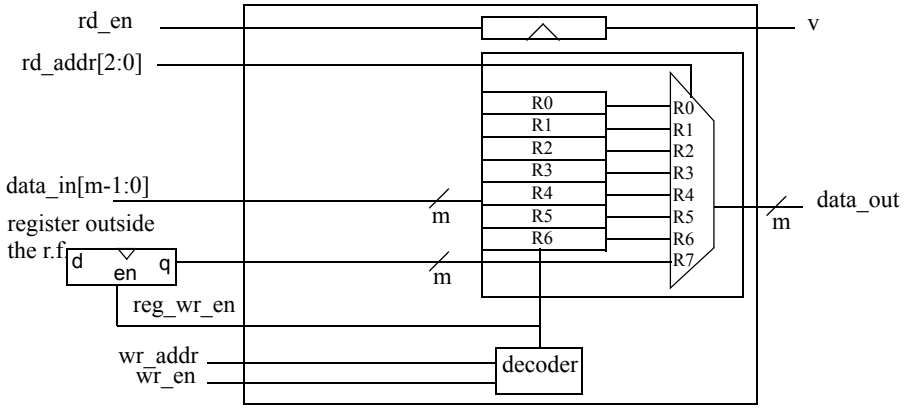
FIGURE 1.13 Connect all registers to outputs



1.2.25 Register File with external register

The external register is an input to the output mux.

FIGURE 1.14 Outside of Register File



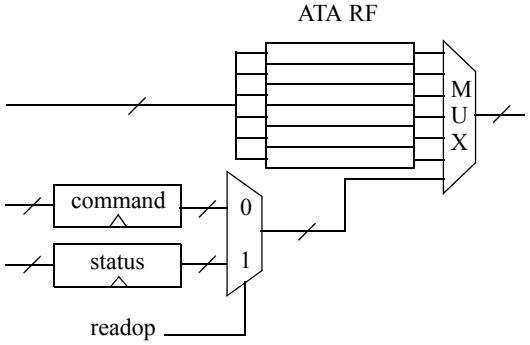
1.2.26 Register file address errors

A checker will validate the addresses presented to the rf. If an address does not fall in the rf address range and the rd_en or wr_en bits are set then and error will be generated.

1.2.27 Register Aliases within Register Files

The same register may have different contexts depending on the address. For example in ATA the task of address space is 0-15 logical addresses but the regfile is only 8 physical addresses. The registers have different contexts depending on whether the current operation is read or is write. Register 7 is a status register and register 15 is a command register but they are both the same physical register. Register file outputs can be either a register or a field. Register file inputs can be either internal to the units which contains the register file and a register external to the register file but input to RF. R7 is not an internal register instead R7 is an input into the RF.

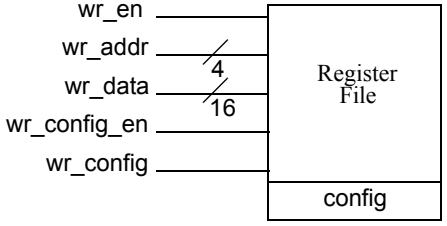
FIGURE 1.15 Aliasing physical register to different addresses



1.2.28 Write registers

In Figure 1.16 on page 12 we should see the write side of a rf which contains a named register. The named register can be written either using the wr_addr, wr_en, and wr_data signals or the wr_config_en and wr_data signals.

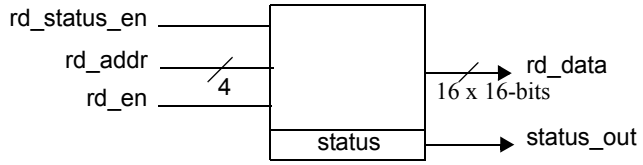
FIGURE 1.16 Write registers



1.2.29 Read registers

The status register can be read via data_out or status_out. In figure 2.21 we should see the read side of a rf which contains a named register. The named register can be read either using the rd_addr, rd_en signals.

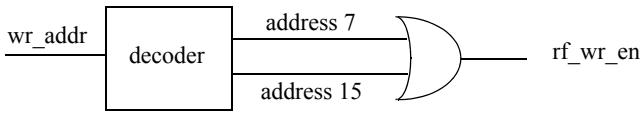
FIGURE 1.17 Read registers



The register with the symbolic name status can be read by data_out or status_out.
The external write enable is generated by the register file's address decoder.

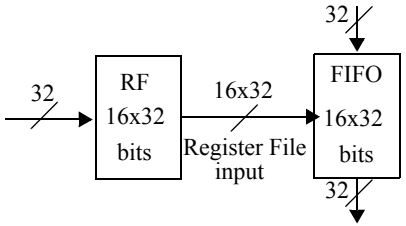
1.2.30

FIGURE 1.18 Detecting write to an address range



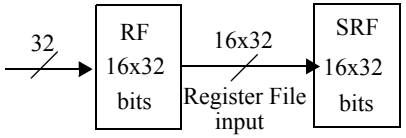
1.2.31

FIGURE 1.19 Pre-loading a register file and shifting its contents into a FIFO in one cycle



The contents of a register file may be written and the entire register file can be shifted into a FIFO in one cycle.

FIGURE 1.20 Pre-loading a register file and shifting its contents into a SRF in one cycle



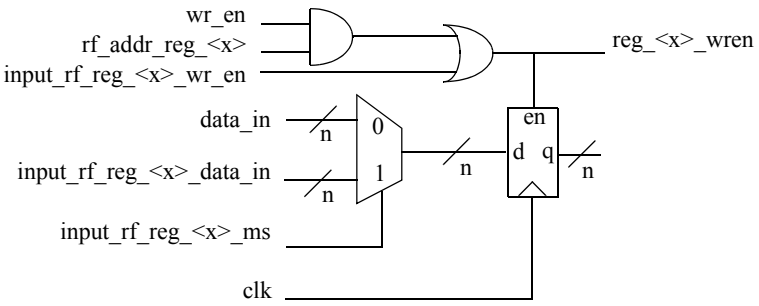
The entire contents of the Shadow Register File is moved in one cycle into the register file.
The register file can serve many purposes. A simultaneous write in one cycle of all configuration registers in an address space guarantees that the bits in the c.r.f. will not conflict with each other if they

were programmed correctly by the software layer. Each register in the SRF can be updated in any order.

1.2.31.1 Register file inputs

A register in an register File can be written using either the data_in and the wr_addr, wr_en or a special input which is tied to a specific register along with a special wr_en.

FIGURE 1.21 Register file inputs



1.3 CSL Register File Command summary (FIX)

```
csl_register_file ID ;
<register_file_name>.width(expression) ;
<register_file_name>.depth(expression) ;
<register_file_name>.output_reg ( register_name );
<register_file_name>.output_individual_reg_fields( register_name );
<register_file_name>.output_individual_reg_fields(
register_name.[field_name] );
<register_file_name>.output_ff;
<register_file_name>.input_ff;
<register_file_name>.valid();
<register_file_name>.event ( (rd|wr) (register_name | all_regs ) );
<register_file_name>.event ( register_name );
<register_file_name>.event( register_name.[field_name] );
<register_file_name>.out_prefix ( register_name );
<register_file_name>.out_prefix ( register_name.field );
 <register_file_name>.in_prefix ( prefix, (register_name | all_regs)
);
```

```
<register_file_name>.in_prefix (register_name);
<register_file_name>.in_prefix (prefix, (register_name | all_regs));
<register_file_name>.starting_address ( address_expression ) ;
<register_file_name>.register_group<group_name>(
[all|<register_field>] ) ;
<register_file_name>.valid_bit( or_all_regs|<register_field> ) ;
<register_file_name>.read_channel( [<register_field_name>|<prefix>] );
<register_file_name>.write_channel( [<register_field_name>|<prefix>]
);
<register_file_name>.do_not_connect_registers_fields_to_ios( [ all |
<register_field>] ) ;
<register_file_name>.connect_input_to_registers_fields(
[all|<register_field>] );
<register_file_name>.clock( <clock_name> );.
<register_file_name>.named_register(<register_field_name>, <address>)
;
<register_file_name>.connect_registers_fields_to_outputs(
[all|<register_field>] ) ;
<register_file_name>.wr_addr( ID );
<register_file_name>.rd_addr( IO );
<register_file_name>.in_prefix;
register_file_name.module();
register_file_name.inline();
register_file_name.library(library_file_name);
register_file_name.rd_en();
register_file_name.out_prefix( prefix, (register_name | all_regs) );
register_file_name.register_fields.explicitly_cleared( [[register name
| register address], <signal_name>] );
register_file_name.register_fields.create_wr_event_list( single_pulse,
[all | <register_field>] );
register_file_name.register_fields.create_rd_event_list( single_pulse,
[all | <register_field>]) ;
```

1.4 CSL Register File Commands

<register_file_name>.**output_reg (** *register_name* **);**

DESCRIPTION :

EXAMPLE :

CSL CODE

VERILOG CODE

<register_file_name>.**output_individual_reg_fields(** *register_name* **);**

DESCRIPTION :

//register the data_out bus

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**output_individual_reg_fields(** *register_name*.**[***field_name***]);**

DESCRIPTION :

//create outputs tied to each register

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**valid();**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**event (** (**rd**|**wr**) (*register_name* | **all_regs**) **);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**event (** *register_name* **);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**event(** *register_name*.**[***field_name***]);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**out_prefix (** *register_name* **);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**out_prefix (** *register_name*.field **);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**in_prefix (** *prefix*, (*register_name* | **all_regs**) **);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**in_prefix (***register_name***);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**in_prefix (***prefix*, (*register_name* | **all_regs**)**);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

csl_register_file *ID* **;**

DESCRIPTION :

Declares a new register file with the name <register_file_name>.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**width(***expression***) ;**

DESCRIPTION :

Declare the width of the register file words.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**depth(***expression***) ;**

DESCRIPTION :

Declare the number of words in the register file.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**address_range(***<address_range>***) ;**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**starting_address (** *address_expression* **) ;**

DESCRIPTION :
Declare the starting address of the register file. address_expression can be an absolute address or can be relative to another address.

EXAMPLE :
//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**register_group*<group_name>*(**
[all|*<register_field>***]) ;**

DESCRIPTION :
Group the registers_fields in the argument list and name the group. Operations listed below can be performed on named groups of registers_fields.

EXAMPLE :
//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**valid_bit(or_all_regs|***<register_field>***) ;**

DESCRIPTION :
A pipelined valid bit is generated from the logical OR of the read enable(s) or from a read of a named register. This is used to tell the downstream logic that an read has occurred and that the data at the output is valid.

EXAMPLE :
//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**read_channel([***<register_field_name>***|***<prefix>***]);**

DESCRIPTION :
Add a read channel to a register file. The read channel consists of the signals <n>_rd_data, <n>_rd_addr and optionally <n>_rd_en, where <n> is either the name of the register file (if no name is passed to read_channel) or <prefix> or <register_field_name>. Note that <register_field_name>

has to already be declared.

EXAMPLE :

//small descritption of the command

CSL CODE

VERILOG CODE

<register_file_name>.**write_channel([***<register_field_name>*|*<prefix>***]);**

DESCRIPTION :

Add a write channel to a register file. The write channel consists of the signals <n>_wr_data, <n>_wr_addr and optionally <n>_wr_en, where <n> is either the name of the register file (if no name is passed to write_channel) or <prefix> or <register_field_name>. Note that <register_field_name> has to already be declared.The construct implements the following operation:

```
if(<registe_file_name>_wr_en) begin
   <register_file_name>[<n.>_wr_addr] = <n>_wr_data;
end
always @ (posedge clk)
   <n>_rd_data = <register_file_name>[<n>_rd_addr];
or
if(<n>_rd_en) begin
   <n>_rd_data = <register_file_name>[<n>_rd_addr];
end
```

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**do_not_connect_registers_fields_to_ios([all | *<register_field>***]) ;**

DESCRIPTION :

This directive is the default for the register file. All registers_fields are written by the data_in, address, and wr_en signals. All registers_fields outputs are connected to the data_out, address, and are optionally read when the rd_en signal is asserted. This directive can be overriden by the connect directives below.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**connect_input_to_registers_fields(**
[all|*<register_field>***]);**

DESCRIPTION :

Each register_field in the argument list is connected to a separate register file input. A data input to the register file called <register_field_name>_input is connected to the data input of register_field. A write enable input to the register file called <register_field_name>_wr_en is connected to the write enable input of register_field. The register_field is written with the <register_field_name>_input when the <register_field_name>_wr_en is asserted.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

FIX :

connect_all_registers_to_inputs,
connect_inputs_to_all_registers,
connect_all_registers_to_outputs,
connect_input_to_reg/field,
connect_reg/field_to_output

<register_file_name>.**clock(** *<clock_name>* **);.**

DESCRIPTION :

The <register_file_name> will use the clock named <clock_name> for the clocked operations in the register file. Note that if the clock is not specified then the module clock will be used. This default only works when there is one module clock

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**named_register(***<register_field_name>***,**
*<address>***) ;**

DESCRIPTION :

The named_register operation will add the <register_field_name> to the <address>. This operation associates the <register_field_name> with the address <address>. Note that more than one logical register can be added to a physical address.

CSL generic/alternative may be used for adding a register to the memory map:

```
<csl_vector_object>.register(<register_name>, <address>);
<csl_scalar_object>.csl_address_range(start_address);
<csl_vector_object>.csl_address_range(start_address, end_address);
```

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**connect_registers_fields_to_outputs([all|***<register_field>***]) ;**

DESCRIPTION :

The output of each register in the argument list is connected to a separate output of the register file. A read enable input to the register file called <register_field_name>_rd_en is connected to the read enable input of register_field. The register_field is written with the <register_field_name>_input when the <register_field_name>_rd_en is asserted.

Figure 1.25 Register file with a read from an individual register named status// FIX

The register can still be read out to the data_out signal using the global register file address and optional global read enable signal. In order to save power we may want to add another option to only enable the address decoder for the global register file output mux if all of the <register_field_name>_rd_en signals are off (the NOR of all of the <register_field_name>_rd_en signals is off).

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**in_prefix;**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the command

CSL CODE

VERILOG CODE

<register_file_name>.*<register_fields>*.**explicitly_cleared(**
[[*register_name* **|** *register_address***],** *<signal_name>***]) ;**

DESCRIPTION :

Once written the register has to be explicitly cleared by a write to another address.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.*<register_fields>*.**create_wr_event_list(**
single_pulse, [all | *<register_field>***]) ;**

DESCRIPTION :

This creates a signal that will be asserted when there is a write to the corresponding register. For each signal name in the signal list a signal will be created. The name will be <signal_name>_wr_event. The event signal will be asserted when there is a write to the corresponding register. If single_pulse is true then when the register is written a single pulse is generated on a line called <register_fields_name>_event. Otherwise the event signal will be held high until the register is read or written. The event can be generated when there is a write to the register via the global wr_en/wr_data/wr_addr or the register specific wr_en/wr_data/wr_addr signals.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.*<register_fields>*.**create_rd_event_list(**
single_pulse, [all | *<register_field>***]) ;**

DESCRIPTION :

This creates a signal that will be asserted when there is a read from the corresponding register. For each signal name in the signal list a signal will be created. The name will be <signal_name>_rd_event. The event signal will be asserted when there is a read from the corresponding register. If single_pulse is true then when the register is written a single pulse is generated

on a line called <register_fields_name>_event. Otherwise the event signal will be held high until the register is read or written. The event can be generated when there is a read from the register via the global rd_en/rd_data/rd_addr or the register specific rd_en/rd_data/rd_addr signals. Note that the correspnding rd_en must be defined in the CSL register file specfication.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**output;**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**output_ff;**

DESCRIPTION :

add an output flip flop for all output signals. Register the output of the register file.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**input_ff;**

DESCRIPTION :

add an input flip flop for n input signals.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**out_prefix([***<all_regs>*** | *<regname>*.field]**
);

DESCRIPTION :

prefix the output names with the specified prefix. Prefix the input names with the specified prefix.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**wr_addr(** *ID* **);**

DESCRIPTION :

write the register file[address] with data when the write enable is asserted.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

<register_file_name>.**rd_addr(** *IO* **);**

DESCRIPTION :

read the contents of register file[addr] to data_out or <register_fields_name>_output when <expression> is asserted.

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

register_file_name.**module();**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

register_file_name.**inline();**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

register_file_name.**library(***library_file_name***);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

register_file_name.**rd_en();**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

register_file_name.**out_prefix(** *prefix*, (*register_name* | **all_regs**) **);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

register_file_name.*register_fields*.**explicitly_cleared(** [[register name | register address], <*signal_name*>] **);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

register_file_name.*register_fields*.**create_wr_event_list(** single_pulse, [all | <*register_field*>] **);**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

register_file_name.*register_fields*.**create_rd_event_list(** single_pulse, [all | <*register_field*>]**) ;**

DESCRIPTION :

//description of the command

EXAMPLE :

//small description of the example

CSL CODE

VERILOG CODE

1.5 CSL Register File Examples

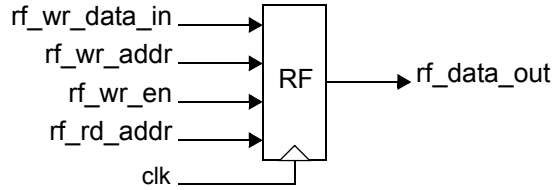
1.5.1 Register file with no special options

FIGURE 1.22 shows the inputs to a black box labeled RF. The black box is a register file (RF). The implementation of the black box can be inferred from the CSL code.

CSL CODE

```
csl_constant D_WIDTH = 32;
csl_constant A_WIDTH = 8;
csl_register_file rf;
rf.width( D_WIDTH );
rf.depth( A_WIDTH );
rf.clock( clk ) ; // clock name does not need to be declared if there
is only one clock in the module
```

FIGURE 1.22 Register file with no special options



VERILOG CODE

```
module register_file(clk, reset, rf_wr_data_in, rf_wr_addr, rf_wr_en,
rf_rd_addr, rf_data_out);
   parameter D_WIDTH=32;
   parameter A_WIDTH=8;
   input [D_WIDTH - 1: 0] rf_wr_data_in;
   input [A_WIDTH - 1: 0] rf_wr_addr;
   input                          rf_wr_en, clk, reset ;
   output [D_WIDTH - 1: 0] rf_data_out;
   input [A_WIDTH - 1: 0] rf_rd_addr;
   reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0], rf_data_out;
   always @ (posedge clk) begin
      if(rf_wr_en) begin
         rf[rf_wr_addr] <= rf_wr_data_in;
      end
      rf_data_out <= rf[rf_rd_addr];
```

```
    end
endmodule
```

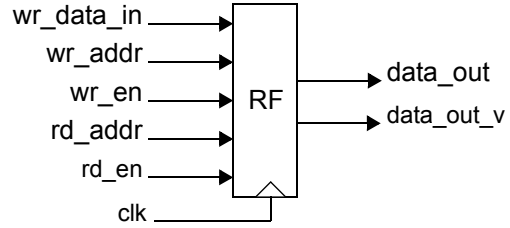
C++ CODE

1.5.2 Register file with read valid bit

CSL CODE

```
csl_register_file rf;
rf.width(D_WIDTH);
rf.depth(A_WIDTH);
rf.clock(clk) ;
rf.valid();
rf.rd_en();
```

FIGURE 1.23 Register file with read valid bit



VERILOG CODE

```
module register_file(clk, reset, rf_wr_data_in, rf_wr_addr, rf_wr_en,
rf_status_wr_data_in, rf_status_wr_addr, rf_status_wr_en, rf_data_out,
rf_rd_addr, rf_rd_en);
   parameter A_WIDTH =8;
   parameter D_WIDTH =8;
   input [D_WIDTH - 1: 0] rf_wr_data_in;
   input [A_WIDTH - 1: 0] rf_wr_addr;
   input  rf_wr_en, reset, clk,  rf_status_wr_en;
   input [D_WIDTH - 1: 0] rf_status_wr_data_in;
   input [A_WIDTH - 1: 0] rf_status_wr_addr;
   output [D_WIDTH - 1: 0] rf_data_out;
   input [A_WIDTH - 1: 0] rf_rd_addr;
   input rf_rd_en;
   reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0], rf_data_out;
  always @ (posedge clk) begin
```

```
        if(rf_wr_en) begin
           rf[rf_wr_addr] <= rf_wr_data_in;
        end
        if(rf_rd_en) begin
         rf_data_out <= rf[rf_rd_addr];
        end
     end
      always @ (posedge clk) begin
        if(rf_status_wr_en) begin
           rf[rf_status_wr_addr] <= rf_status_wr_data_in;
        end
      end
      endmodule
```

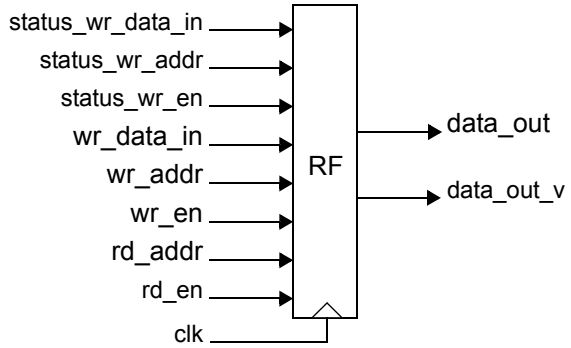
C++ CODE

1.5.2.1 Register file with write to an individual register named status

CSL CODE

```
csl_register_file rf;
rf.width(D_WIDTH);
rf.depth(A_WIDTH);
rf.clock(clk) ;
rf.valid();
rf.named_register(status, 12) ; // address 12 has the name status asso-
ciated with it
rf.connect_input_to_registers_fields(status ) ;
```

FIGURE 1.24 Register file with write to an individual register named status



VERILOG CODE

```
module register_file(clk, reset, rf_wr_data_in, rf_wr_addr, rf_wr_en,
rf_status_wr_data_in, rf_status_wr_addr, rf_status_wr_en, rf_data_out,
rf_rd_addr, rf_rd_en);
   parameter A_WIDTH =8;
   parameter D_WIDTH =8;

   input [D_WIDTH - 1: 0] rf_wr_data_in;
   input [A_WIDTH - 1: 0] rf_wr_addr;
   input clk, reset, rf_wr_en;
   input [D_WIDTH - 1: 0] rf_status_wr_data_in;
   input [A_WIDTH - 1: 0] rf_status_wr_addr;
   input                  rf_status_wr_en;
   output [D_WIDTH - 1: 0] rf_data_out;
   input [A_WIDTH - 1: 0] rf_rd_addr;
   input  rf_rd_en;
   reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0], rf_data_out;
   always @ (posedge clk) begin
      if(rf_wr_en) begin
         rf[rf_wr_addr] <= rf_wr_data_in;
      end
      if(rf_rd_en) begin
       rf_data_out <= rf[rf_rd_addr];
      end
  end
   always @ (posedge clk) begin
      if(rf_status_wr_en) begin
         rf[rf_status_wr_addr] <= rf_status_wr_data_in;
```

```
        end
      end
   endmodule
```

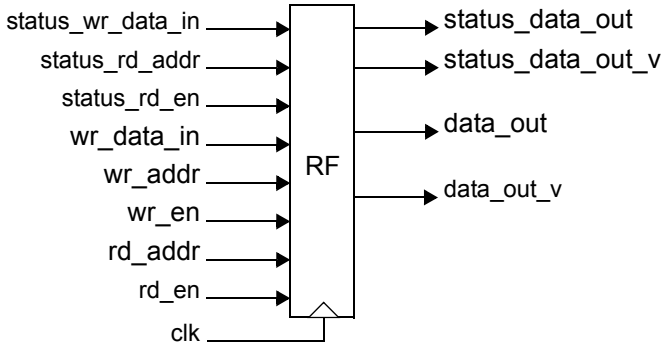
C++ CODE

1.5.2.2 Register file with a read from an individual register named status

CSL CODE

```
   csl_register_file rf;
   rf.width( D_WIDTH );
   rf.depth( A_WIDTH );
   rf.clock( clk );
   rf.valid();
   rf.named_register( status, 12 ) ; // address 12 has the name status
   associated with it
   rf.connect_output_to_registers_fields( status ) ;
```

FIGURE 1.25 Register file with a read from an individual register named status// FIX



VERILOG CODE

```
   module register_file(clk, reset, rf_wr_data_in, rf_wr_addr, rf_wr_en,
   rf_data_out, rf_rd_addr, rf_rd_en);
      parameter A_WIDTH =8;
      parameter D_WIDTH =8;
      input [D_WIDTH - 1: 0] rf_wr_data_in;
      input [A_WIDTH - 1: 0] rf_wr_addr;
      input  clk, reset, rf_wr_en;
      reg  rf_status_rd_data_in,rf_status_rd_addr,rf_status_rd_en;
      output [D_WIDTH - 1: 0] rf_data_out;
```

```
    input [A_WIDTH - 1: 0] rf_rd_addr;
    input  rf_rd_en;
    reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0];
    reg [D_WIDTH - 1: 0] rf_data_out;
    always @ (posedge clk) begin
       if(rf_wr_en) begin
          rf[rf_wr_addr] <= rf_wr_data_in;
       end
       if(rf_rd_en) begin
        rf_data_out <= rf[rf_rd_addr];
       end
  end
    always @ (posedge clk) begin
       if(rf_status_rd_en) begin
          rf[rf_status_rd_addr] <= rf_status_rd_data_in;
       end
    end
endmodule
```

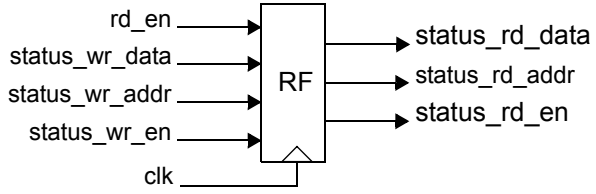
C++ CODE

1.5.3 Register file with read enable

CSL CODE

FIX

FIGURE 1.26



VERILOG CODE

```
module register_file(clk, reset, rf_rd_en, rf_status_wr_addr,
rf_status_wr_en, rf_status_rd_data,
rf_status_rd_addr,rf_status_wr_data, rf_status_rd_en);
   parameter A_WIDTH =8;
   parameter D_WIDTH =8;
```

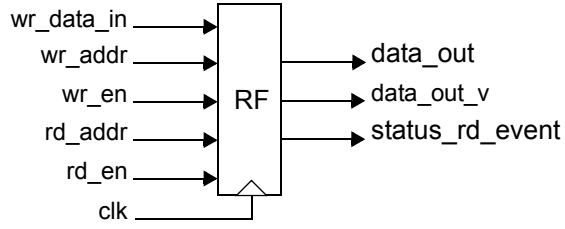
```
    input [D_WIDTH - 1: 0] rf_status_wr_data;
    input [A_WIDTH - 1: 0] rf_status_wr_addr;
    input  clk, reset, rf_rd_en, rf_status_wr_en;
    output [D_WIDTH - 1: 0] rf_status_rd_data;
    output [A_WIDTH - 1: 0] rf_status_rd_addr;
    output rf_status_rd_en;
    reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0];
    reg [D_WIDTH - 1: 0] rf_status_rd_data;
    always @ (posedge clk) begin
       if(rf_status_wr_en) begin
          rf[rf_status_wr_addr] <= rf_status_wr_data;
       end
       if(rf_rd_en) begin
        rf_status_rd_data <= rf[rf_status_wr_data];
       end
   end
endmodule
```

C++ CODE

1.5.4 Register file with an event generated from a read to an individual register named status

CSL CODE

FIGURE 1.27 Register file with an event generated from a read to an individual register named status



VERILOG CODE

```
 module register_file(clk, reset, rf_wr_data_in, rf_wr_addr, rf_wr_en,
rf_data_out, rf_rd_addr, rf_rd_en, rf_status_rd_event);
   parameter A_WIDTH =8;
   parameter D_WIDTH =8;
```

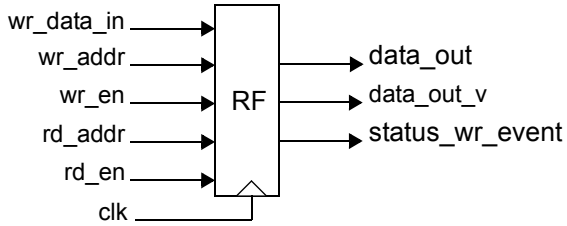
```
    input [D_WIDTH - 1: 0] rf_wr_data_in;
    input [A_WIDTH - 1: 0] rf_rd_addr;
    input [A_WIDTH - 1: 0] rf_wr_addr;
    input  clk, reset, rf_wr_en,rf_rd_en;
    reg  rf_status_rd_data_in, rf_status_rd_addr, rf_status_rd_en;
    output [D_WIDTH - 1: 0] rf_data_out;
    output rf_status_rd_event;
    reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0];
    reg [D_WIDTH - 1: 0] rf_data_out;
    always @ (posedge clk) begin
       if(rf_wr_en) begin
          rf[rf_wr_addr] <= rf_wr_data_in;
       end
       if(rf_rd_en) begin
        rf_data_out <= rf[rf_rd_addr];
       end
  end
    always @ (posedge clk) begin
       if(rf_rd_en) begin
           rf_status_rd_event <= rf[rf_rd_addr] ;
       end
    end
endmodule
```

C++ CODE

1.5.5 Register file with an event generated from a write to an individual register named status

CSL CODE

FIGURE 1.28 Register file with an event generated from a write to an individual register named status



VERILOG CODE

```
 module register_file(clk, reset, rf_wr_data_in, rf_wr_addr, rf_wr_en,
rf_data_out, rf_rd_addr, rf_rd_en, rf_status_wr_event);
   parameter A_WIDTH =8;
   parameter D_WIDTH =8;
   input [D_WIDTH - 1: 0] rf_wr_data_in;
   input [A_WIDTH - 1: 0] rf_rd_addr;
   input [A_WIDTH - 1: 0] rf_wr_addr;
   input  clk, reset, rf_wr_en,rf_rd_en;
   reg  rf_status_rd_data_in, rf_status_rd_addr, rf_status_rd_en;
   output [D_WIDTH - 1: 0] rf_data_out;
   output rf_status_wr_event;
   reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0];
   reg [D_WIDTH - 1: 0] rf_data_out;
   always @ (posedge clk) begin
      if(rf_wr_en) begin
         rf[rf_wr_addr] <= rf_wr_data_in;
      end
      if(rf_rd_en) begin
       rf_data_out <= rf[rf_rd_addr];
      end
  end
   always @ (posedge clk) begin
      if(rf_rd_en) begin
        rf_status_wr_event <= rf[rf_rd_addr] ;
      end
   end
endmodule
```

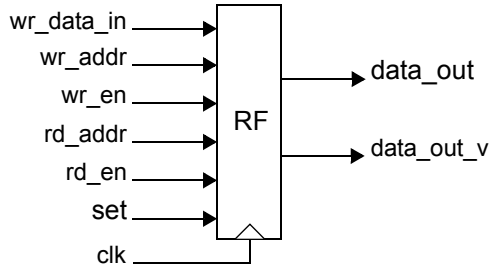
C++ CODE

1.5.6 Register file with an global set operation which sets all registers to a known value

CSL CODE

Register file with a global set operation which sets all registers to a known value.

FIGURE 1.29 Register file with an global clear operation which clears all registers to zero



VERILOG CODE

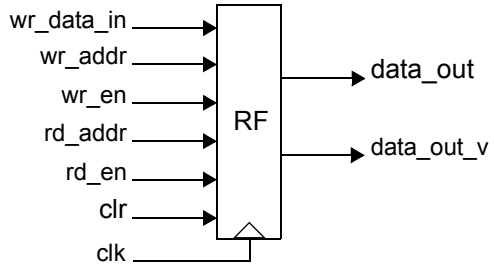
```
module register_file(clk, init, rf_rd_en, rf_wr_addr, rf_wr_data_in,
rf_wr_en, rf_rd_addr, rf_data_out,rf_data_out_v);
   parameter A_WIDTH =8;
   parameter D_WIDTH =8;
   integer i;
   input [D_WIDTH - 1: 0] rf_wr_data_in;
   input [A_WIDTH - 1: 0] rf_wr_addr;
   input [A_WIDTH - 1: 0] rf_rd_addr;
   input  clk,init rf_rd_en, rf_wr_en;
   output [D_WIDTH - 1: 0] rf_data_out;
   output [D_WIDTH - 1: 0] rf_data_out_v;
    reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0];
   reg [D_WIDTH - 1: 0] rf_data_out;
   always @ (posedge clk) begin
      if(rf_wr_en) begin
         rf[rf_wr_addr] <= rf_wr_data_in;
      end
      if(rf_rd_en) begin
      rf_data_out <= rf_wr_data_in;
      end
  end
   always @ (posedge clk) begin
    if (init) begin
     for (i = 0; i < A_WIDTH-1; i = i + 1) begin
```

```
    rf[i] <= D_WIDTH-1'b0;
    end
    end
   end
  endmodule
```

1.5.6.1 Register file with an global clear operation which clears all registers to zero

CSL CODE

FIGURE 1.30 Register file with an global clear operation which clears all registers to zero



VERILOG CODE

```
module register_file(clk, clr, rf_rd_en, rf_wr_addr, rf_wr_data_in,
rf_wr_en, rf_rd_addr, rf_data_out,rf_data_out_v);
   parameter A_WIDTH =8;
   parameter D_WIDTH =8;
   integer i;
   input [D_WIDTH - 1: 0] rf_wr_data_in;
   input [A_WIDTH - 1: 0] rf_wr_addr;
   input [A_WIDTH - 1: 0] rf_rd_addr;
   input  clk,clr, rf_rd_en, rf_wr_en;
   output [D_WIDTH - 1: 0] rf_data_out;
   output [D_WIDTH - 1: 0] rf_data_out_v;
    reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0];
   reg [D_WIDTH - 1: 0] rf_data_out;
   always @ (posedge clk) begin
      if(rf_wr_en) begin
         rf[rf_wr_addr] <= rf_wr_data_in;
      end
      if(rf_rd_en) begin
      rf_data_out <= rf_wr_data_in;
      end
```

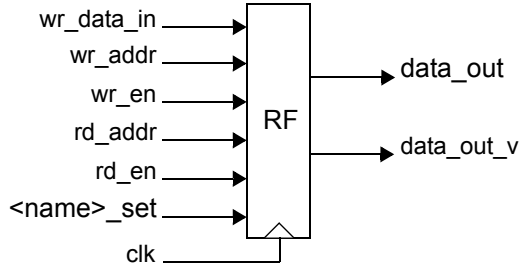
```
    end
    always @ (posedge clk) begin
     if (clr) begin
      for (i = 0; i < A_WIDTH-1; i = i + 1) begin
    rf[i] <= D_WIDTH-1'b0;
    end
    end
   end
  endmodule
```

C++ CODE

1.5.7 Register file with a set operation on a specific register/field or register/field group which sets the registers/fields to a known value

CSL CODE

FIGURE 1.31 Register file with a set operation on a specific register/field or register/field group which sets the registers/fields to a known value

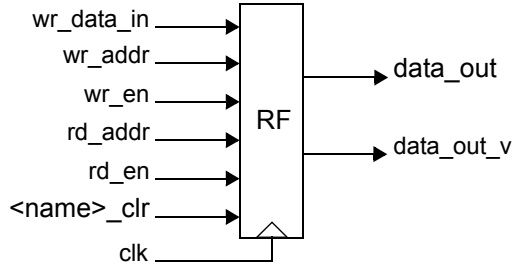


VERILOG CODE

1.5.8 Register file with a clear operation on a specific register/field or register/field group which clears the registers/fields to zero

CSL CODE

FIGURE 1.32 Register file with a clear operation on a specific register/field or register/field group which clears the registers/fields to zero



VERILOG CODE

1.5.8.1

CSL CODE

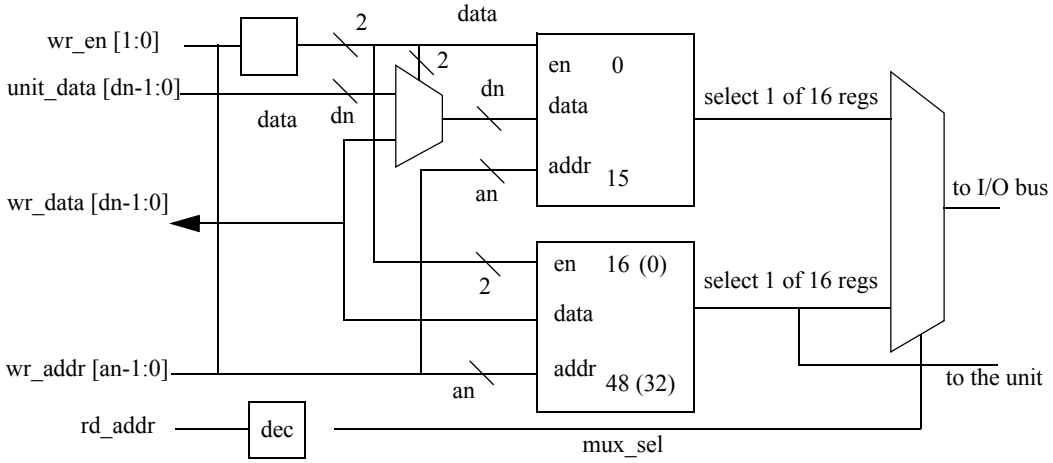
VERILOG CODE

1.5.8.2 Building trees of Register Files

Multiple Register Files may be used to create a single address space within a unit.

CSL CODE

FIGURE 1.33 Combining register Files



VERILOG CODE

1.5.8.3 Producer/consumer register file buffer

CSL CODE

FIGURE 1.34 Register File used as buffer between Producer Consumer modules

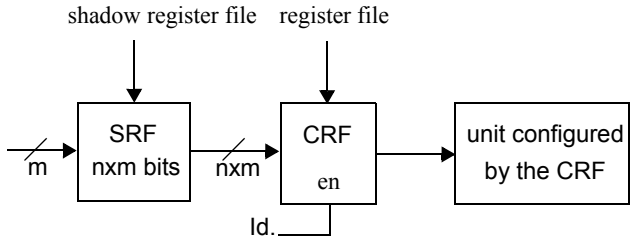
VERILOG CODE

1.5.8.4 Shadow register within R.f.'s

One register can shadow another register which is why a register can be an element and an element can be a register (register linking).

CSL CODE

FIGURE 1.35 Shadow register file configuration



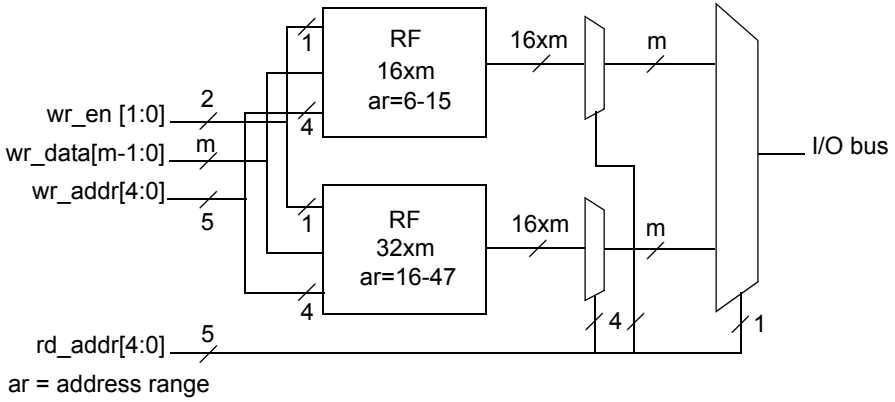
VERILOG CODE

1.5.8.5 Grouping Register Files into one address space

Multiple Register Files can be gromped into one address space using wrapper logic.

CSL CODE

FIGURE 1.36 Two Register Files grouped into one address range



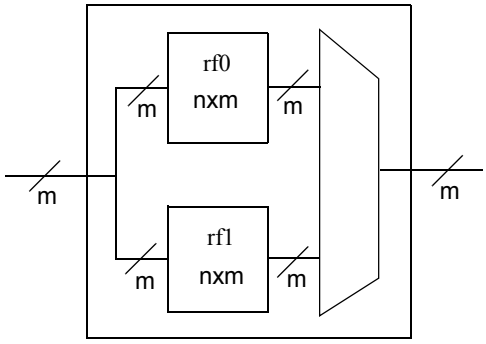
VERILOG CODE

1.5.8.6 “Ping Pong” register file

A “Ping Pong” register file architecture can be used to switch between two identical register files at any clock edge.

CSL CODE

FIGURE 1.37 “Ping Pong”



VERILOG CODE

1.5.8.7

CSL CODE

```
csl_register_file rf;
rf.width(D_WIDTH);
rf.depth(A_WIDTH);
rf.awc();//bitrange defaults to width
rf.arc();//bitrange defaults to log2(depth)
```

FIGURE 1.38

VERILOG CODE

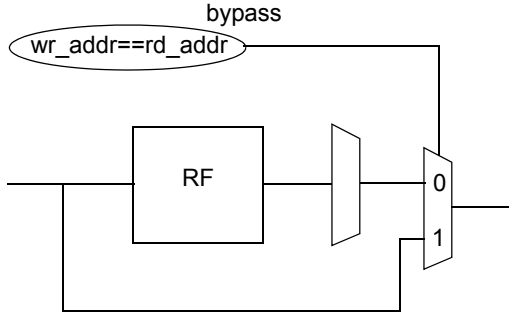
```
module register_file(clk, reset, wr_data_in, wr_addr, wr_en, rd_addr,
data_out);
```

```
    input [D_WIDTH - 1: 0] wr_data_in;
    input [A_WIDTH - 1: 0] wr_addr;
    input [A_WIDTH - 1: 0] rd_addr;
    input [A_WIDTH - 1: 0] ?!?
    output [D_WIDTH - 1: 0] data_out;
    reg [D_WIDTH - 1: 0] rf [A_WIDTH - 1: 0];
    always @ (posedge clk) begin
       if(wr_en) begin
          rf[wr_addr] = wr_data_in;
       end
       data_out = rf[rd_addr];
    end
end module
-------------------------------------------------------
```

1.5.8.8 Register file with bypass

CSL CODE

FIGURE 1.39



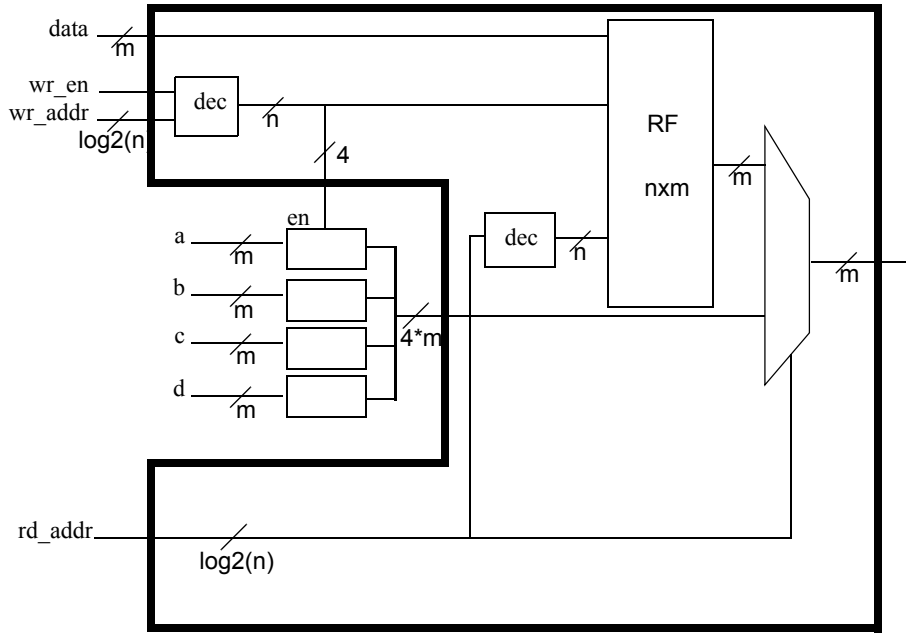
VERILOG CODE

C++ CODE

1.5.8.9 Register file with external registers

CSL CODE

FIGURE 1.40 External registers connected to register file



VERILOG CODE

C++ CODE

1.6 CSL Register File Checker

1.6.1 CSL Register File Reports