# Veristruct: An IEEE1364.1995 (Verilog 1995) Preprocessor

Michael Cowell

October 10, 2008

## 1 Overview

Veristruct is an an IEEE1364.1995 preprocessor that adds some C-style struct support to the Verilog language. It takes as input Veristruct files (with a `.vs` extension) and struct definition files (with a `.struct` extension). It outputs standard Verilog files (with a `.v` extension). Veristruct files are, for the most part, standard Verilog files—but, as well as the normal `nets` and `regs`, bundled, hierarchical variables can be declared and used.

Veristruct can process one veristruct file per invocation (which may include many `.struct` struct definition files).

Veristruct is written in object oriented Perl. Please feel free to fix any bugs you find![1]

## 2 Installation

First you will have to get Veristruct. A tarball of the latest release should be available at the Rachael SPARC website[2]. Alternatively, you can check out the source directly with a command like:

```
svn co https://www.rachaelsparc.org/repository/projects/tools/veristruct/
```

Note that code in SVN is not guaranteed to produce correct output or even run, but you will probably get more features than in the release tarball.

Next, you need to add the Veristruct classes to your Perl distribution. One day I might put Veristruct in CPAN and this process will be automated, but until then you'll need to do things manually. First work out where you want to place the classes. They need to be in your Perl include path. You can either add the folder that you've got Veristruct in to your Perl include path—or find out what's in your `@INC` (run something like `perl -e 'print join(", ",@INC);'`) and copy the `Verilog` folder from the Veristruct root into one of those directories.

Finally, you need to put the `veristruct.pl` script (the application) somewhere in your shell path. Once again you can either change your shell's `PATH` variable to point to your veristruct folder, or copy `veristruct.pl` to `/usr/local/bin` or something. If you're copying it, you might want to strip `.pl` off of the name (to have a nicer looking command).

## 3 Command-line Options

The following table describes each of the Veristruct application's command line arguments:

[1] That includes those in this documentation.

[2] `https://www.rachaelsparc.org`

| Argument | Action |
|---|---|
| `--help`, `-h` | Prints usage information. |
| `--debug`, `-d` | Prints debugging information while parsing the input files. This is useful if (for instance) Veristruct fails to parse a syntactically valid file, and you are interested in what mistake it made. Note: this option is intended for developers. |
| `--write`, `-w` | By default Veristruct will not overwrite an existing file with the same name as the output file. This options causes it to do so. |
| `--infile=file`, `-i file` | This compulsory argument should be used to specify the input `.vs` file. Note: the `.vs` extension is not compulsory, but is recommended for clarity. |
| `--outfilef=file`, `-o file` | This compulsory argument should be used to specify the output `.v` file. Note: the `.v` extension is not compulsory, but is recommended for clarity. |
| `--libpath=path`, `-L path` | This argument (which can be used multiple times) should be used to specify the search path used when trying to locate `.struct` files referenced in the input. It can be either a path, or a comma separated list of paths. |

The command line arguments are not case sensitive, and it is sufficient to specify just enough of the command as is unique (although this is not recommended, because more commands may be added in the future).

# 4 Input & Output Files

Veristruct takes two sorts of input files, `.vs` (or Veristruct) files and `.struct` (or struct definition) files. Following is a description of the two.

## 4.1 Veristruct files — .vs

Veristruct files are, for the most part, standard IEEE1364.1995 (Verilog) files. The following extra syntax is allowed:

- **`` `sinclude``**: This extra pre-processor directive can be used to instruct Veristruct to read in a struct definition file. The file to be read in should follow immediately after the token (with possible whitespace, other than line breaks). This directive may be used as many times as desired, to read in any number of `.struct` files. Note: the directive should not be used inside modules, and surrounding `` `ifdef `` directives will **not** be honoured.
- **`reg`, `wire` and port struct declarations**: If any of these Verilog declarations are followed by the name of a defined struct, the next word will interpreted as an instance of that struct. All uses of that variable, throughout the rest of the module (and, in the case of module port lists, earlier in the module as well) will be recognised by Veristruct as struct references, and will be re-written in the output file. Arrays of structs can be declared in the same way that normal vectors are declared (the range should precede the struct name).
- **Struct element references**: Any expression token (or l-value) in the input `.vs` file of the form `inst_name.element` (of arbitrary element depth, and with optional ranges) are recognised and processed by Veristruct. The semantics are the same as in C. Note that slices of struct elements, where the struct is declared as an array, are not allowed as l-values.

Whole structs can only be used as l-values if the expression to the right is also a whole struct, of the same type, and nothing else.

Wire declarations can include assignments, as usual.

## 4.2 Struct declaration files — .struct

The EBNF for struct declaration files is:

```
struct_file := struct_block
struct_block := struct_defn { struct_block }
struct_defn := struct struct_name { elem_block };
elem_block := elem ; { elem_block }
elem := wire_element | signed_element | struct_element
wire_element := wire((elem_name [ max :  min ]) | (elem_name) | (elem_name ` define_string))
signed_element := signed((elem_name [ max :  min ]) | (elem_name) | (elem_name ` define_string))
struct_element := struct_name elem_name
```

The atomic tokens are:

- `struct_name`: A valid Verilog variable name. Trying to use "wire" (which is illegal anyway) is doubly bad. Double underscores are not allowed (this is not actually enforced, but is a very bad idea).
- `elem_name`: Similar to above.
- `max`: An integer greater than `min`.
- `min`: An integer less than `max`.
- `define_string`: Any string without whitespace.

The semantics are similar to C. Note the main syntactical difference is that a name does not need to be specified twice (as in C, after the closing brace). C-style comments are allowed wherever (they are stripped out prior to parsing). Line comments count as a line break, block comments count as nothing.

### 4.3 IEEE1364.1995 (Verilog) file — .v

Veristruct outputs IEEE1364.1995 compliant `.v` files. Structs declarations are expanded into their elements, and struct element references are expanded into formally compliant variable names. Double underscores are used to indicate depth.

When entire structs are referenced (instance names are used without element references) a guess is made as to what is desired. In sensitivity lists and module port lists, the name is exploded into its constituent elements (and appropriate separator tokens are added). In module instance port lists, it is assumed that the module being instantiated has multiple ports with the instance name as the base (that will be the case if the module was written in Veristruct).

## 5 Limitations

The following limitations currently exist within Veristruct:

- **Namespace collisions not detected**: Veristruct currently does no checking for namespace collisions. If you don't use double underscores in your variables names, though, you should be fine.
- **Limited lvalue support**: Slices of some struct elements (and whole structs, in most contexts) cannot be lvalues. Veristruct will return an error when it encounters these.
- **Structs can only be declared in `.struct` files**: This is a deliberate limitation. It avoids polluting the Verilog syntax even more.
- **`` `ifdefs `` ignored**: Because Veristruct is not designed to replace the normal Verilog preprocessor, it does not do its job. As such, you will need to ensure that do not rely on your structs being conditionally included using normal pre-processor directives. There are many ways around this. Ranges in structs can be tick defined, and variable declarations can be surrounded in `` `ifdef ``s, which will get honoured later.

# 6 Changes

**Changes since the 0.1 release**[3]:

- Signed support added (as per the request made by luked in RCHL10).
- Bugfixes discussed in RCHL12.

# 7 Examples

## 7.1 Example .struct file

Here is an example .struct file:

```
/*****************************************************************************
 *
 * This file is a part of the Rachael SPARC project accessible at
 * https://www.rachaelsparc.org. Unless otherwise noted code is released
 * under the Lesser GPL (LGPL) available at http://www.gnu.org.
 *
 * Copyright (c) 2005:
 *   Michael Cowell
 *
 * Rachael SPARC is based heavily upon the LEON SPARC microprocessor
 * released by Gaisler Research, at http://www.gaisler.com, under the
 * LGPL. Much of the architectural work on Rachael was done by g2
 * Microsystems. Contact michael.cowell@g2microsystems.com for more
 * information.
 *
 *****************************************************************************
 * $Id: $
 * $URL: $
 * $Rev: $
 * $Author: $
 *****************************************************************************
 *
 * Test struct definition file
 *
 *****************************************************************************/


struct memory_bus {
  wire address [31:2];
  wire data [31:0];
  wire req; // This is a line comment in a struct!!
  signed pie;
  signed pie2 [3:1];
};


/* block block */

struct dual_bus {
  memory_bus primary;
  memory_bus secondary;
```

[3]References to issue IDs refer to the Rachael SPARC Scarab bug tracking system. Go to `https://www.rachaelsparc.org/scarab/issues` to see details.

```
  /* block
  block */
};

struct complex {
  signed reall [7:0];
  signed imag [7:0];
};
```

### 7.2 Example .vs file

And an example .vs file that uses those structs:

```
/*********************************************************************
 *
 * This file is a part of the Rachael SPARC project accessible at
 * https://www.rachaelsparc.org. Unless otherwise noted code is released
 * under the Lesser GPL (LGPL) available at http://www.gnu.org.
 *
 * Copyright (c) 2005:
 *   Michael Cowell
 *
 * Rachael SPARC is based heavily upon the LEON SPARC microprocessor
 * released by Gaisler Research, at http://www.gaisler.com, under the
 * LGPL. Much of the architectural work on Rachael was done by g2
 * Microsystems. Contact michael.cowell@g2microsystems.com for more
 * information.
 *
 *********************************************************************
 * $Id: $
 * $URL: $
 * $Rev: $
 * $Author: $
 *********************************************************************
 *
 * Test Verilog file
 *
 *********************************************************************/

`sinclude "test.struct"

module test (
  // Inputs
  clk, reset, pie, pie2
  );
  input clk;
  input reset;
  input memory_bus pie2;
  input dual_bus pie;

  wire  dual_bus db1;

  reg   complex a;

  reg   memory_bus mb1;
  wire  dual_bus db2 = db1;
  reg [4:1] memory_bus mb2;
  reg [4:1] memory_bus mb5;
  reg [4:1] memory_bus mb6;
```

```
47  wire      memory_bus mb3 = mb2[1];
48
49  assign    db2.primary.req = 1'b1, db2.primary.address[31:28] = 4'h4;
50  assign    db2.primary.address[31:28] = 4'h4;
51
52  always @(posedge clk or posedge db2.primary.req
53           or db2.secondary.address or mb2[1]) begin
54    mb1.address <= 32'h12348765;
55    begin assign mb2[1].req = 1'b1;
56      force mb2[2].req = 1'b1;
57      assign thing = mb6[2].address[16:0];
58      release mb2[2].req;
59    end
60  end
61
62  wire [3:0] memory_bus mb2;
63
64  fake_module fake_inst2 (mb2[1], db1, db1.primary.req,
65                          db1.primary.address[1], mb2[1].req);
66
67  fake_module fake_inst ( . port1 ( mb2[1] ) ,
68                          .port2 (db1),
69                          .port3 (mb2[2]),
70                          .port4 (db1.primary.req),
71                          .port5 (db1.primary.address[1]),
72                          .port6 (mb2[1].req));
73
74  assign mb2[3].req = 1'b1;
75
76  wire [15:0] temp;
77  wire [1:0]  dual_bus db3;
78
79  assign      temp = mb2[2].address[15:0], temp = db2[2].primary.data[4:3];
80
81  assign      p=q, db3[2] =db2 , q=p;
82
83  //assign      mb2[2].address[15:0] = 16'h00;
84
85  always @(posedge clk) begin
86    if (mb2[3].primary == 1'b1)
87      temp = mb5[3].address[15:0];
88    while (mb2[1].address[1] == 1'b1)
89      #1;
90    case (mb2[1].req)
91      1'b1: temp <= 4'h1;
92      1'b0: begin
93        temp <= 4'h1;
94        case (mb2[2].req)
95          1'b1: temp <= 4'h2;
96        endcase // case(mb2[2].req)
97      end
98      mb2[3].req, mb2[4].req: temp <= 4'hx;
99    endcase // case(mb2[1].req)
100   for (mb1.req = 1'b1; mb1.req < db1[1].primary.address[15]; mb2[1].req++)
101     temp <= 1'b1;
102   sometask(mb1.req);
103 end
104
105
106 endmodule
107
```

```
/* */
```

### 7.3 Example .v file

Finally, here the the .v file that Veristruct outputs:

```
/*****************************************************************************
 *
 * This file is a part of the Rachael SPARC project accessible at
 * https://www.rachaelsparc.org. Unless otherwise noted code is released
 * under the Lesser GPL (LGPL) available at http://www.gnu.org.
 *
 * Copyright (c) 2005:
 *   Michael Cowell
 *
 * Rachael SPARC is based heavily upon the LEON SPARC microprocessor
 * released by Gaisler Research, at http://www.gaisler.com, under the
 * LGPL. Much of the architectural work on Rachael was done by g2
 * Microsystems. Contact michael.cowell@g2microsystems.com for more
 * information.
 *
 *****************************************************************************
 * $Id: $
 * $URL: $
 * $Rev: $
 * $Author: $
 *****************************************************************************
 *
 * Test Verilog file
 *
 *****************************************************************************/

//`sinclude "test.struct"

module test (
  // Inputs
  clk, reset, pie__secondary__pie2, pie__secondary__pie, pie__secondary__req,
      pie__secondary__data, pie__secondary__address, pie__primary__pie2,
      pie__primary__pie, pie__primary__req, pie__primary__data,
      pie__primary__address, pie2__pie2, pie2__pie, pie2__req, pie2__data,
      pie2__address
  );
  input clk;
  input reset;
  input signed [3:1] pie2__pie2; input signed pie2__pie; input pie2__req; input
      [31:0] pie2__data; input [31:2] pie2__address;
  input signed [3:1] pie__secondary__pie2; input signed pie__secondary__pie;
      input pie__secondary__req; input [31:0] pie__secondary__data; input [31:2]
      pie__secondary__address; input signed [3:1] pie__primary__pie2; input
      signed pie__primary__pie; input pie__primary__req; input [31:0]
      pie__primary__data; input [31:2] pie__primary__address;

  wire signed [3:1] db1__secondary__pie2; wire signed db1__secondary__pie; wire
      db1__secondary__req; wire [31:0] db1__secondary__data; wire [31:2]
      db1__secondary__address; wire signed [3:1] db1__primary__pie2; wire signed
      db1__primary__pie; wire db1__primary__req; wire [31:0] db1__primary__data;
      wire [31:2] db1__primary__address;

  reg signed [7:0] a__reall; reg signed [7:0] a__imag;
```

```
41
42  reg signed [3:1] mb1__pie2; reg signed mb1__pie; reg mb1__req; reg [31:0]
       mb1__data; reg [31:2] mb1__address;
43  wire signed [3:1] db2__secondary__pie2; wire signed db2__secondary__pie; wire
       db2__secondary__req; wire [31:0] db2__secondary__data; wire [31:2]
       db2__secondary__address; wire signed [3:1] db2__primary__pie2; wire signed
       db2__primary__pie; wire db2__primary__req; wire [31:0] db2__primary__data;
       wire [31:2] db2__primary__address; assign db2__secondary__pie2 =
       db1__secondary__pie2; assign db2__secondary__pie = db1__secondary__pie;
       assign db2__secondary__req = db1__secondary__req; assign
       db2__secondary__data = db1__secondary__data; assign db2__secondary__address
        = db1__secondary__address; assign db2__primary__pie2 = db1__primary__pie2;
        assign db2__primary__pie = db1__primary__pie; assign db2__primary__req =
       db1__primary__req; assign db2__primary__data = db1__primary__data; assign
       db2__primary__address = db1__primary__address;
44  reg signed [3:1]mb2__pie2; reg signed [4:1]mb2__pie; reg [4:1]mb2__req; reg
       [31:0]mb2__data[4:1]; reg [31:2]mb2__address[4:1];
45  reg signed [3:1]mb5__pie2; reg signed [4:1]mb5__pie; reg [4:1]mb5__req; reg
       [31:0]mb5__data[4:1]; reg [31:2]mb5__address[4:1];
46  reg signed [3:1]mb6__pie2; reg signed [4:1]mb6__pie; reg [4:1]mb6__req; reg
       [31:0]mb6__data[4:1]; reg [31:2]mb6__address[4:1];
47  wire signed [3:1] mb3__pie2; wire signed mb3__pie; wire mb3__req; wire [31:0]
       mb3__data; wire [31:2] mb3__address; assign mb3__pie2 = mb2__pie2[1];
       assign mb3__pie = mb2__pie[1]; assign mb3__req = mb2__req[1]; assign
       mb3__data = mb2__data[1]; assign mb3__address = mb2__address[1];
48
49  assign    db2__primary__req = 1'b1, db2__primary__address[31:28] = 4'h4;
50  assign    db2__primary__address[31:28] = 4'h4;
51
52  wire [31:2] temp__mb6__address=mb6__address[2]; always @(posedge clk or
       posedge db2__primary__req
53            or db2__secondary__addressmb2__pie2[1] or mb2__pie[1] or mb2__req[1]
                 or mb2__data[1] or mb2__address[1]2[1]) begin
54    mb1__address <= 32'h12348765;
55    begin assign mb2__req[1] = 1'b1;
56      force mb2__req[2] = 1'b1;
57      assign thing = temp__mb6__address[16:0];
58      release mb2__req[2];
59    end
60  end
61
62  wire signed [3:1]mb2__pie2; wire signed [3:0]mb2__pie; wire [3:0]mb2__req;
       wire [31:0]mb2__data[3:0]; wire [31:2]mb2__address[3:0];
63
64  fake_module fake_inst2 (mb2__pie2[1], mb2__pie[1], mb2__req[1], mb2__data[1],
       mb2__address[1], db1__secondary__pie2, db1__secondary__pie,
       db1__secondary__req, db1__secondary__data, db1__secondary__address,
       db1__primary__pie2, db1__primary__pie, db1__primary__req,
       db1__primary__data, db1__primary__address, db1__primary__req,
65                          db1__primary__address[1], mb2__req[1]);
66
67  fake_module fake_inst ( .port1__pie2 (mb2__pie2[1]), .port1__pie (mb2__pie[1])
       , .port1__req (mb2__req[1]), .port1__data (mb2__data[1]), .port1__address (
       mb2__address[1]) ,
68                          .port2__secondary__pie2 (db1__secondary__pie2), .
                               port2__secondary__pie (db1__secondary__pie), .
                               port2__secondary__req (db1__secondary__req), .
                               port2__secondary__data (db1__secondary__data), .
                               port2__secondary__address (db1__secondary__address)
                               , .port2__primary__pie2 (db1__primary__pie2), .
                               port2__primary__pie (db1__primary__pie), .
```

```
                              port2__primary__req (db1__primary__req), .
                              port2__primary__data (db1__primary__data), .
                              port2__primary__address (db1__primary__address),
                            .port3__pie2 (mb2__pie2[2]), .port3__pie (mb2__pie[2])
                              , .port3__req (mb2__req[2]), .port3__data (
                              mb2__data[2]), .port3__address (mb2__address[2]),
                            .port4 (db1__primary__req),
                            .port5 (db1__primary__address[1]),
                            .port6 (mb2__req[1]));

  assign mb2__req[3] = 1'b1;

  wire [15:0] temp;
  wire signed [3:1] db3__secondary__pie2; wire signed db3__secondary__pie; wire
     db3__secondary__req; wire [31:0] db3__secondary__data; wire [31:2]
     db3__secondary__address; wire signed [3:1] db3__primary__pie2; wire signed
     db3__primary__pie; wire db3__primary__req; wire [31:0] db3__primary__data;
     wire [31:2] db3__primary__address;

  wire [31:0] temp__db2__primary__data=db2__primary__data[2]; wire [31:2]
     temp__mb2__address=mb2__address[2]; assign      temp = temp__mb2__address
     [15:0], temp = temp__db2__primary__data[4:3];

  assign      p=q, db3__secondary__req[2]=db2__secondary__req,
     db3__secondary__data[2]=db2__secondary__data,db3__secondary__address[2]=
     db2__secondary__address,db3__primary__req[2]=db2__primary__req,
     db3__primary__data[2]=db2__primary__data,db3__primary__address[2]=
     db2__primary__address, q=p;

  //assign      mb2[2].address[15:0] = 16'h00;

  wire [31:2] temp__db1__primary__address=db1__primary__address[1]; wire [31:2]
     temp__mb2__address=mb2__address[1]; wire [31:2] temp__mb5__address=
     mb5__address[3]; always @(posedge clk) begin
    if (mb2__primary[3] == 1'b1)
      temp = temp__mb5__address[15:0];
    while (temp__mb2__address[1] == 1'b1)
      #1;
    case (mb2__req[1])
      1'b1: temp <= 4'h1;
      1'b0: begin
        temp <= 4'h1;
        case (mb2__req[2])
          1'b1: temp <= 4'h2;
        endcase // case(mb2[2].req)
      end
      mb2__req[3], mb2__req[4]: temp <= 4'hx;
    endcase // case(mb2[1].req)
    for (mb1__req = 1'b1; mb1__req < temp__db1__primary__address[15]; mb2__req
       [1]++)
      temp <= 1'b1;
    sometask(mb1__req);
  end


endmodule

/* */
```