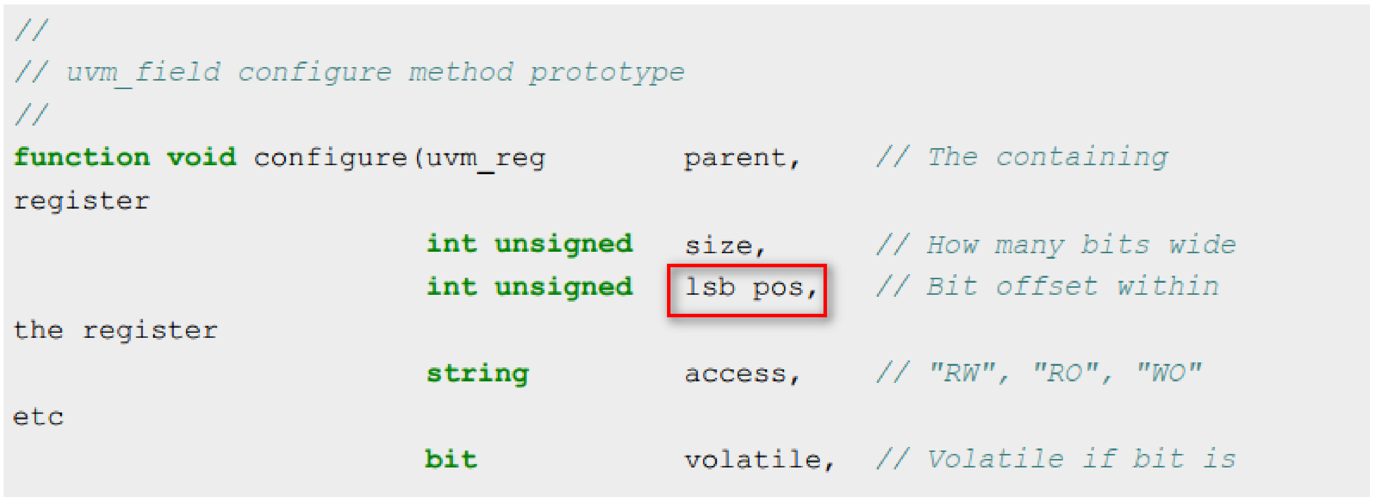
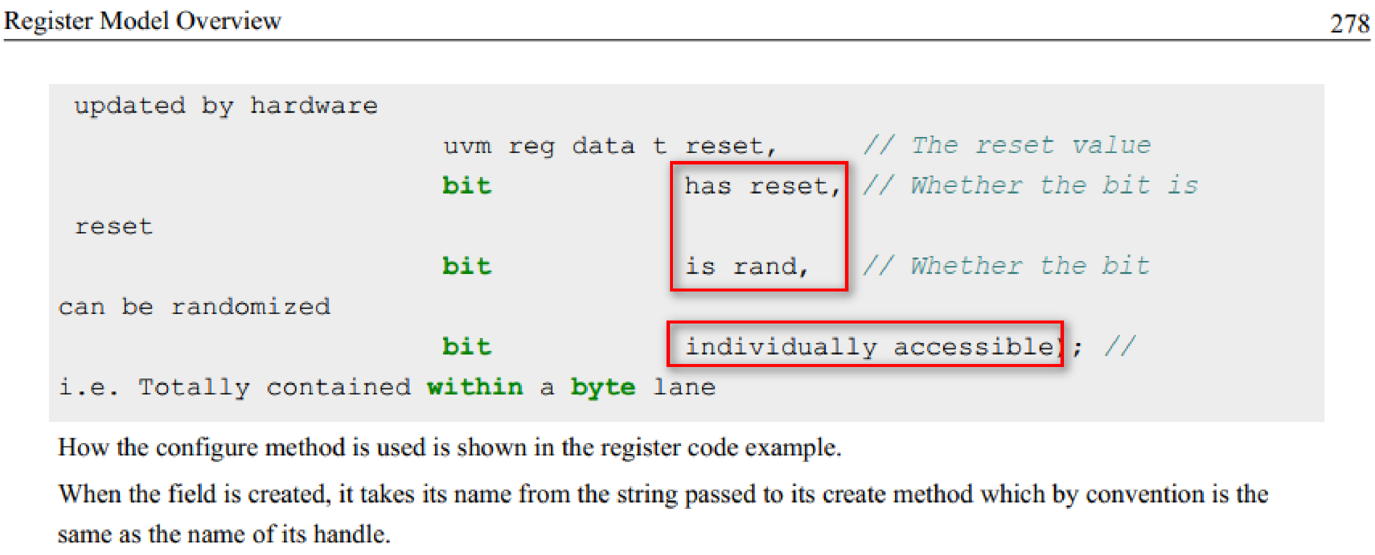
Some format issue in UVM cookbook

In page 277. lack of underline



```
//
// uvm_field configure method prototype
//
function void configure(uvm_reg         parent,    // The containing
register
                        int unsigned    size,      // How many bits wide
                        int unsigned    lsb pos,   // Bit offset within
the register
                        string          access,    // "RW", "RO", "WO"
etc
                        bit             volatile,  // Volatile if bit is
```

In page 278. lack of underline.

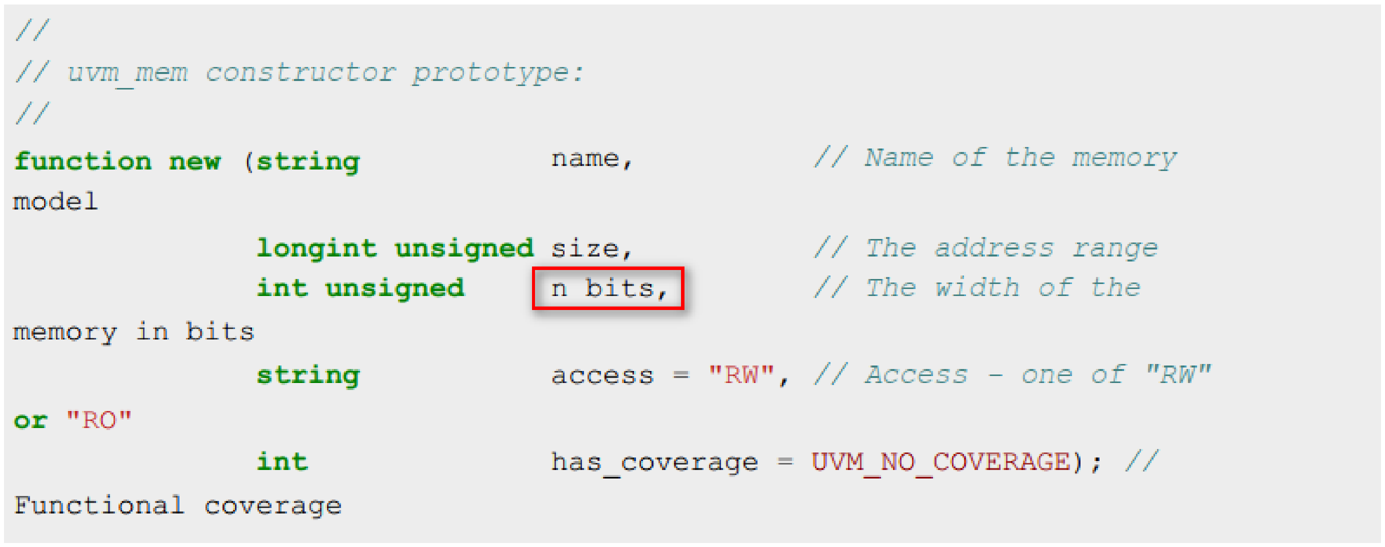


```
 updated by hardware
                        uvm reg data t reset,      // The reset value
                        bit            has reset,  // Whether the bit is
 reset
                        bit            is rand,    // Whether the bit
can be randomized
                        bit            individually accessible); //
i.e. Totally contained within a byte lane
```

How the configure method is used is shown in the register code example.

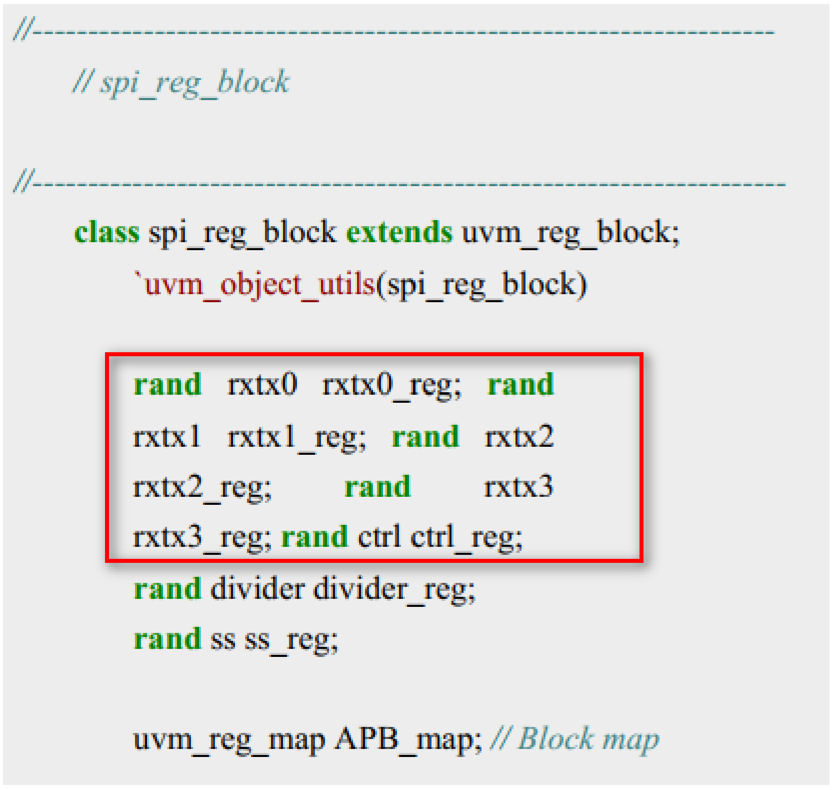
When the field is created, it takes its name from the string passed to its create method which by convention is the same as the name of its handle.

In page 280. lack of underline.



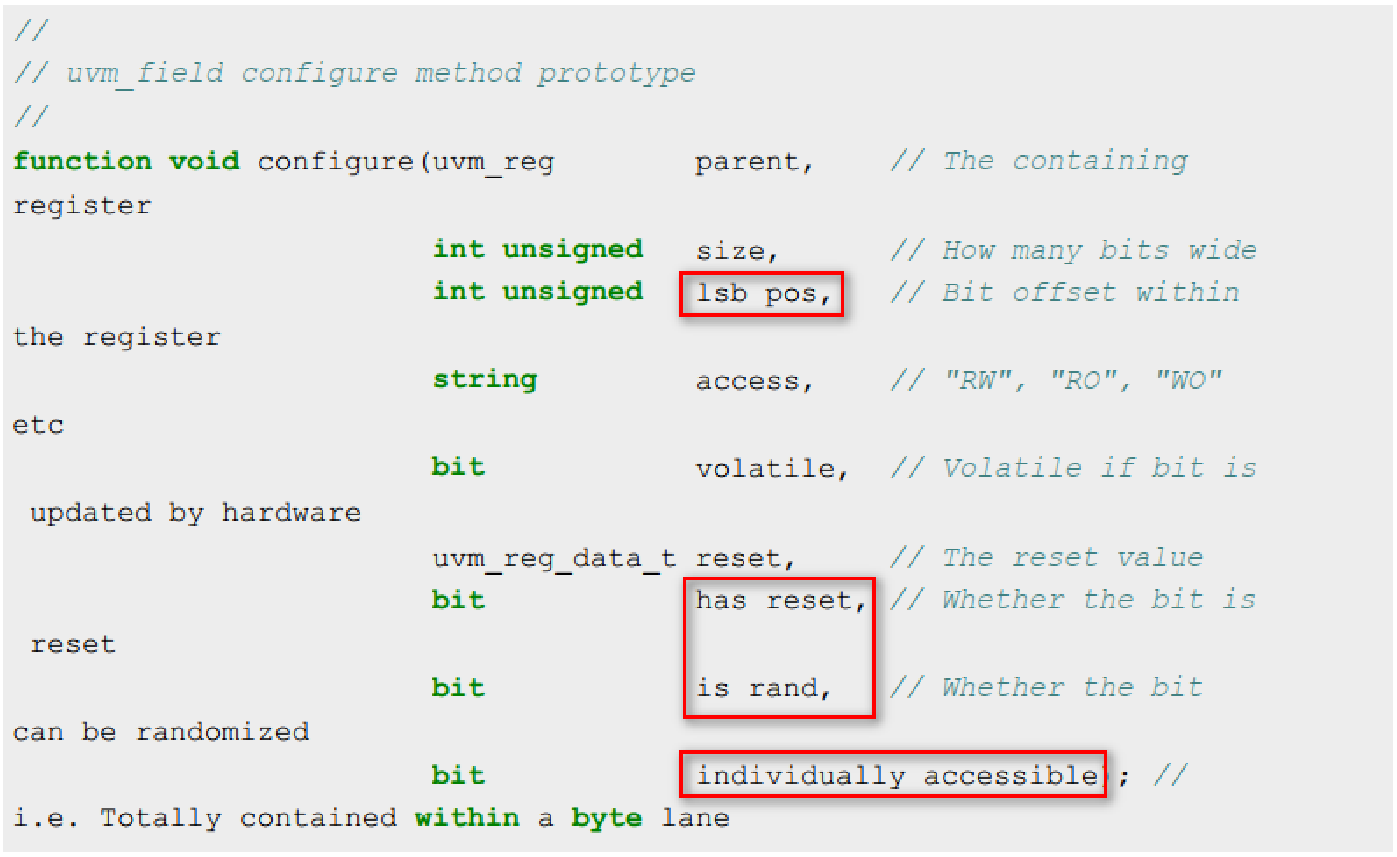
```
//
// uvm_mem constructor prototype:
//
function new (string             name,           // Name of the memory
model
              longint unsigned size,             // The address range
              int unsigned      n bits,          // The width of the
memory in bits
              string            access = "RW",   // Access - one of "RW"
or "RO"
              int               has_coverage = UVM_NO_COVERAGE); //
Functional coverage
```

In page 283. Code format is weird.



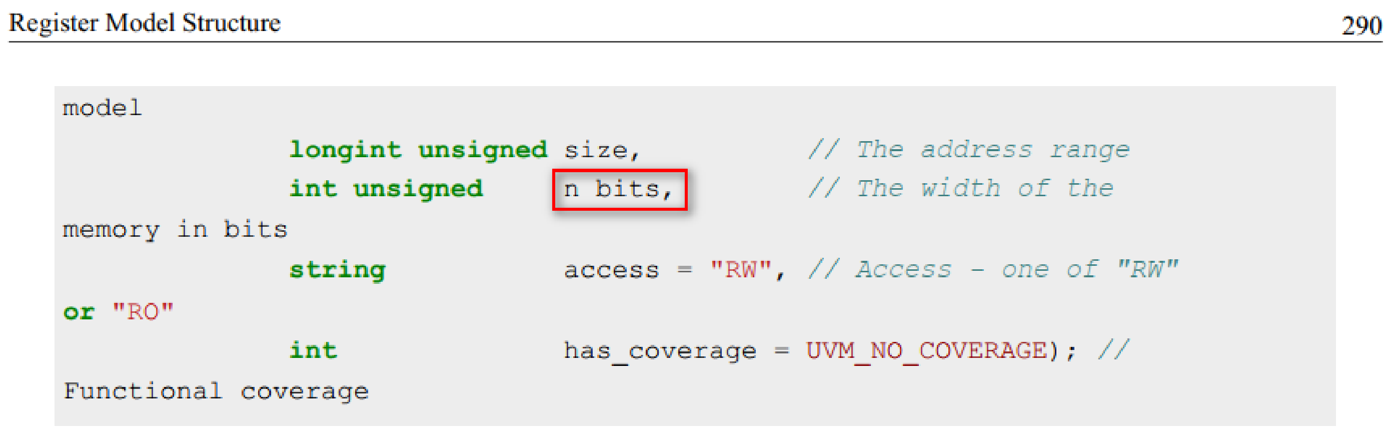
```
//-------------------------------------------------------------------
    // spi_reg_block

//--------------------------------------------------------------------
    class spi_reg_block extends uvm_reg_block;
        `uvm_object_utils(spi_reg_block)

        rand  rxtx0  rxtx0_reg;  rand
        rxtx1  rxtx1_reg;  rand  rxtx2
        rxtx2_reg;       rand       rxtx3
        rxtx3_reg; rand ctrl ctrl_reg;
        rand divider divider_reg;
        rand ss ss_reg;

        uvm_reg_map APB_map; // Block map
```

In page 287. lack of underline.



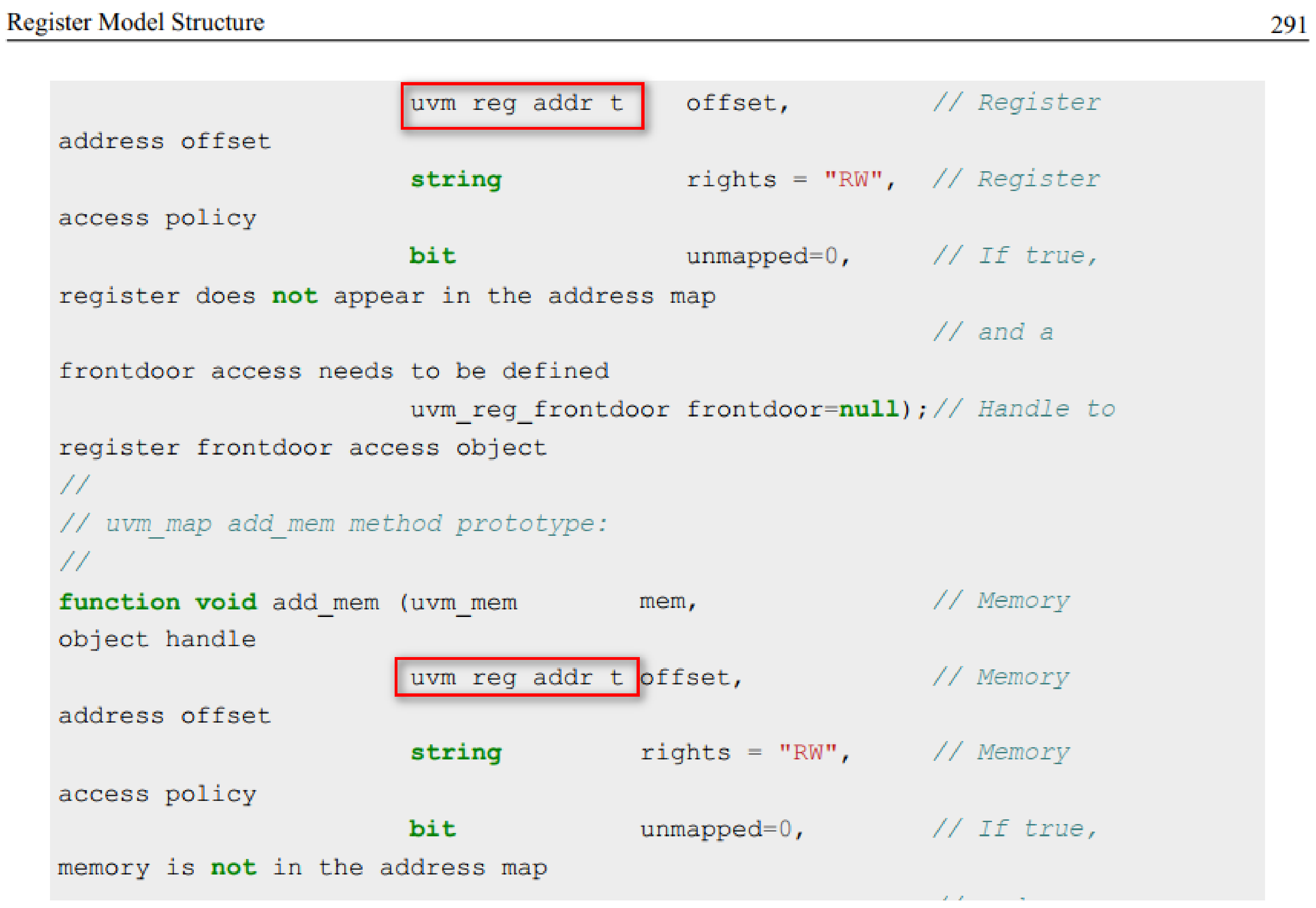
```
//
// uvm_field configure method prototype
//
function void configure(uvm_reg        parent,    // The containing
register
                        int unsigned   size,      // How many bits wide
                        int unsigned   lsb pos,   // Bit offset within
the register
                        string         access,    // "RW", "RO", "WO"
etc
                        bit            volatile,  // Volatile if bit is
 updated by hardware
                        uvm_reg_data_t reset,     // The reset value
                        bit            has reset, // Whether the bit is
 reset
                        bit            is rand,   // Whether the bit
can be randomized
                        bit            individually accessible); //
i.e. Totally contained within a byte lane
```

In page 290. lack of underline.

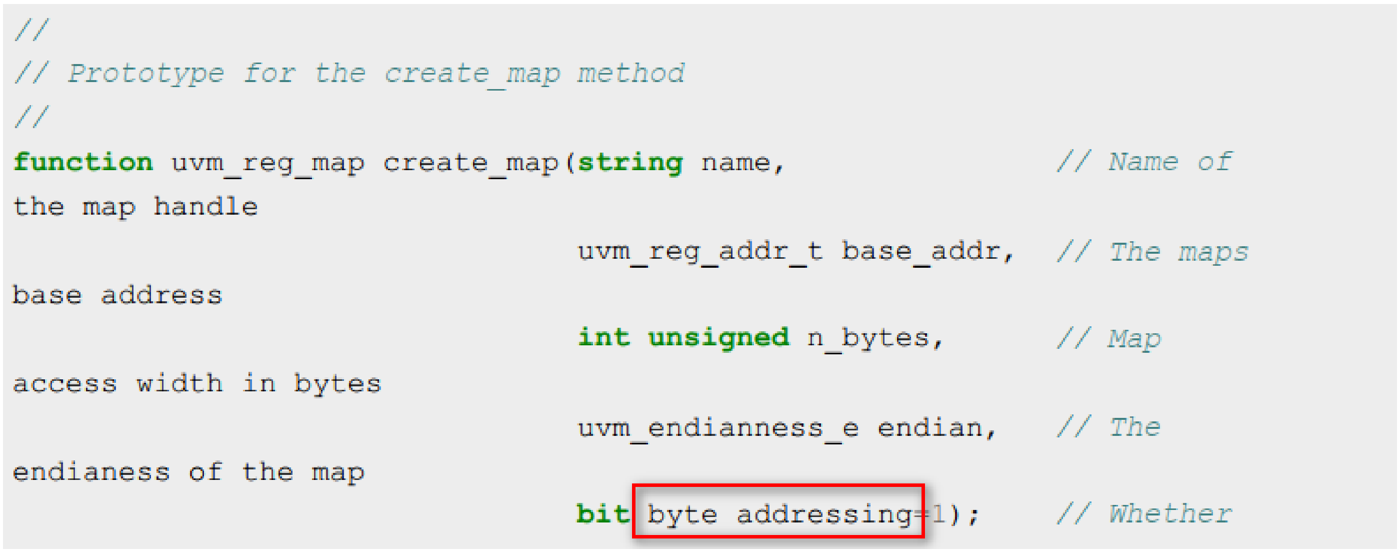


```
model
              longint unsigned size,          // The address range
              int unsigned     n bits,        // The width of the
memory in bits
              string           access = "RW", // Access - one of "RW"
or "RO"
              int              has_coverage = UVM_NO_COVERAGE); //
Functional coverage
```

In page 291. lack of underline.

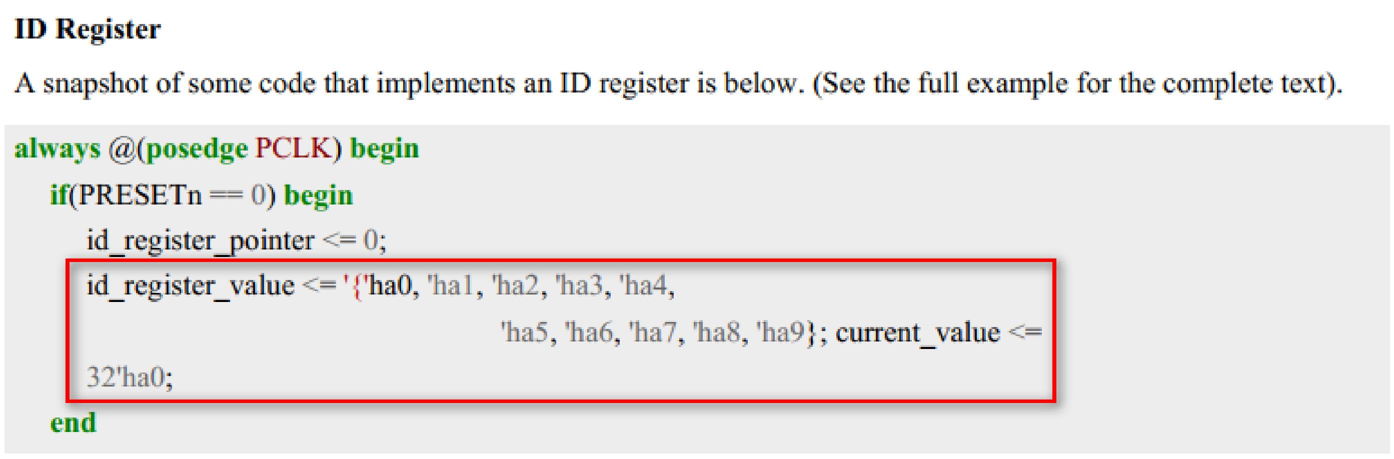


```
                       uvm reg addr t    offset,          // Register
address offset
                       string            rights = "RW",   // Register
access policy
                       bit               unmapped=0,      // If true,
register does not appear in the address map
                                                          // and a
frontdoor access needs to be defined
                       uvm_reg_frontdoor frontdoor=null);// Handle to
register frontdoor access object
//
// uvm_map add_mem method prototype:
//
function void add_mem (uvm_mem        mem,                // Memory
object handle
                      uvm reg addr t offset,              // Memory
address offset
                       string         rights = "RW",      // Memory
access policy
                       bit            unmapped=0,         // If true,
memory is not in the address map
```



```
//
// Prototype for the create_map method
//
function uvm_reg_map create_map(string name,               // Name of
the map handle
                                uvm_reg_addr_t base_addr,  // The maps
base address
                                int unsigned n_bytes,      // Map
access width in bytes
                                uvm_endianness_e endian,   // The
endianess of the map
                                bit byte addressing=1);    // Whether
```

In page 314. Code format is weird.

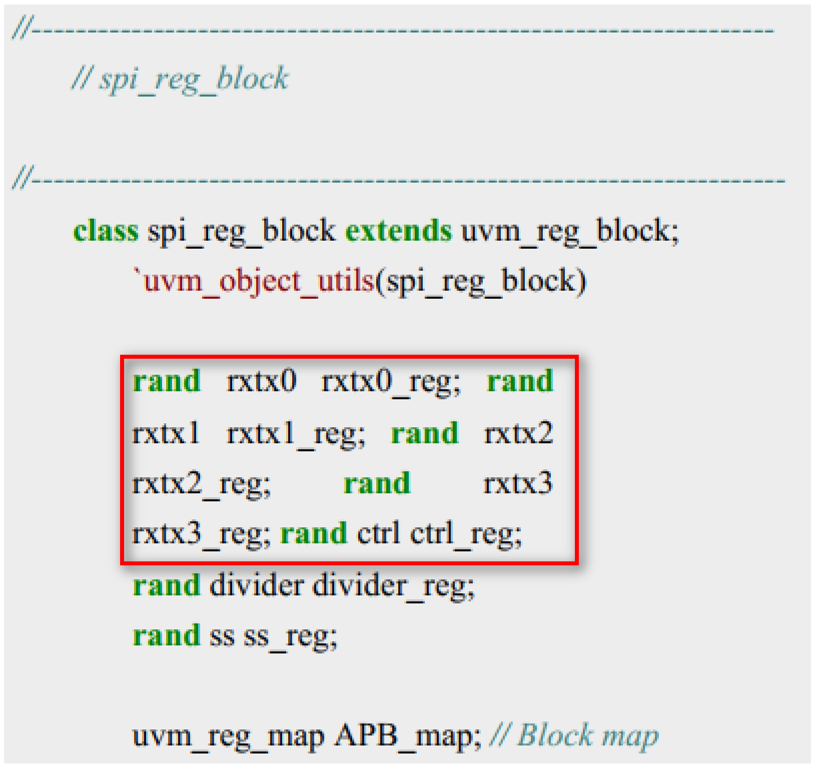


ID Register

A snapshot of some code that implements an ID register is below. (See the full example for the complete text).

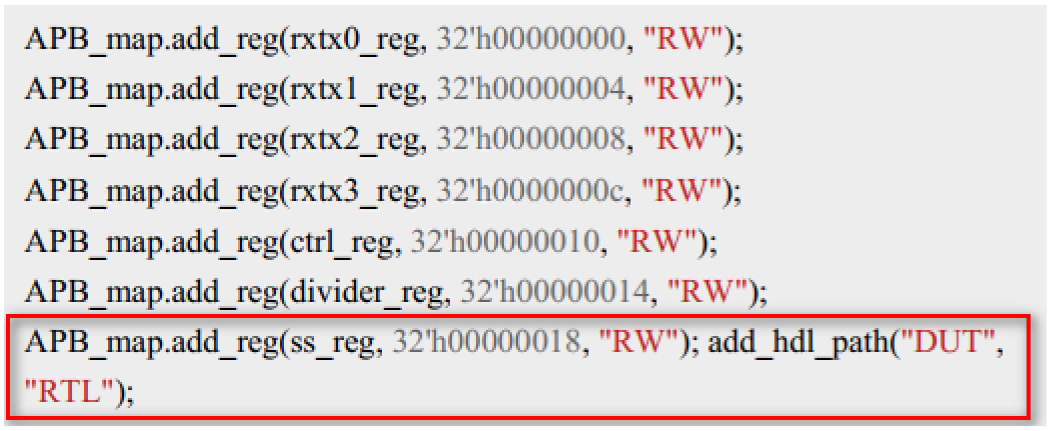
```
always @(posedge PCLK) begin
   if(PRESETn == 0) begin
      id_register_pointer <= 0;
      id_register_value <= '{'ha0, 'ha1, 'ha2, 'ha3, 'ha4,
                                         'ha5, 'ha6, 'ha7, 'ha8, 'ha9}; current_value <=
      32'ha0;
   end
```

In page 320. Code format is weird.



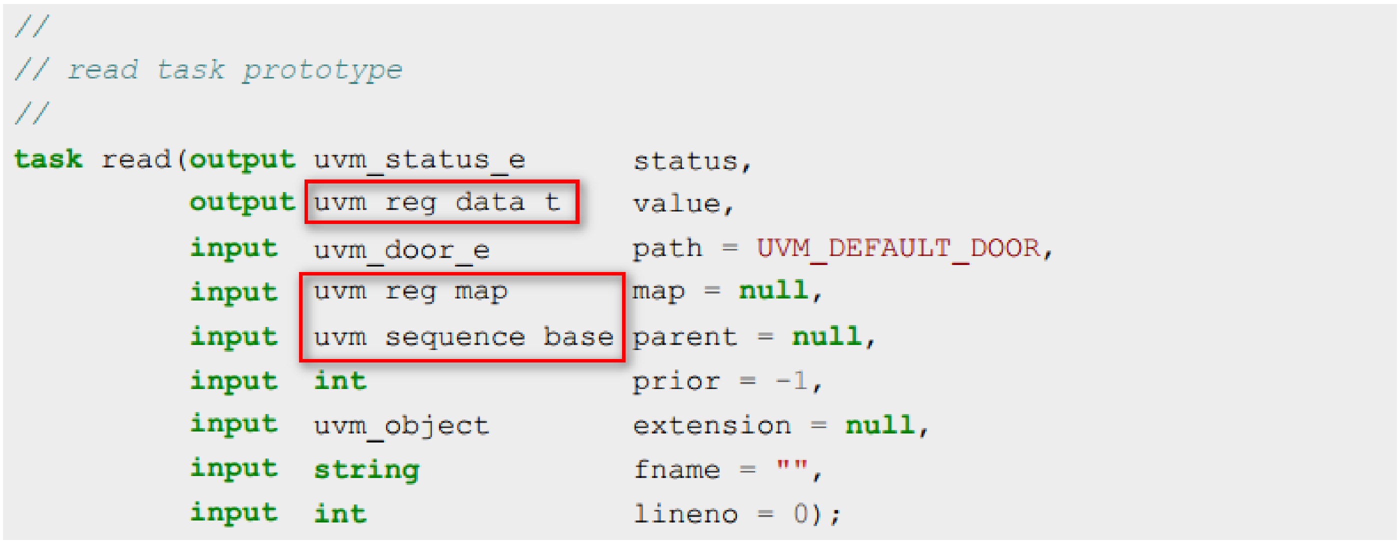
```
//-------------------------------------------------------------------
     // spi_reg_block

//--------------------------------------------------------------------
     class spi_reg_block extends uvm_reg_block;
          `uvm_object_utils(spi_reg_block)

          rand   rxtx0   rxtx0_reg;   rand
          rxtx1   rxtx1_reg;   rand   rxtx2
          rxtx2_reg;         rand         rxtx3
          rxtx3_reg; rand ctrl ctrl_reg;
          rand divider divider_reg;
          rand ss ss_reg;

          uvm_reg_map APB_map; // Block map
```

In page 321. Code format is weird.



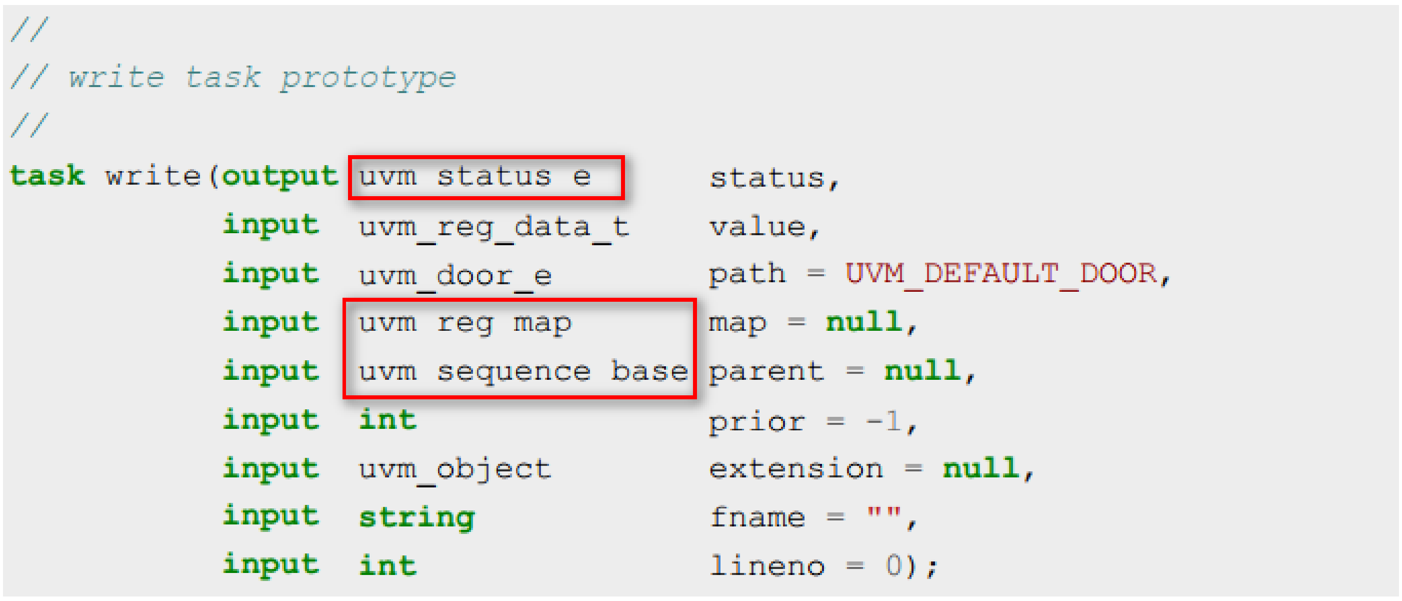
```
APB_map.add_reg(rxtx0_reg, 32'h00000000, "RW");
APB_map.add_reg(rxtx1_reg, 32'h00000004, "RW");
APB_map.add_reg(rxtx2_reg, 32'h00000008, "RW");
APB_map.add_reg(rxtx3_reg, 32'h0000000c, "RW");
APB_map.add_reg(ctrl_reg, 32'h00000010, "RW");
APB_map.add_reg(divider_reg, 32'h00000014, "RW");
APB_map.add_reg(ss_reg, 32'h00000018, "RW"); add_hdl_path("DUT",
"RTL");
```

In page 327. lack of underline.



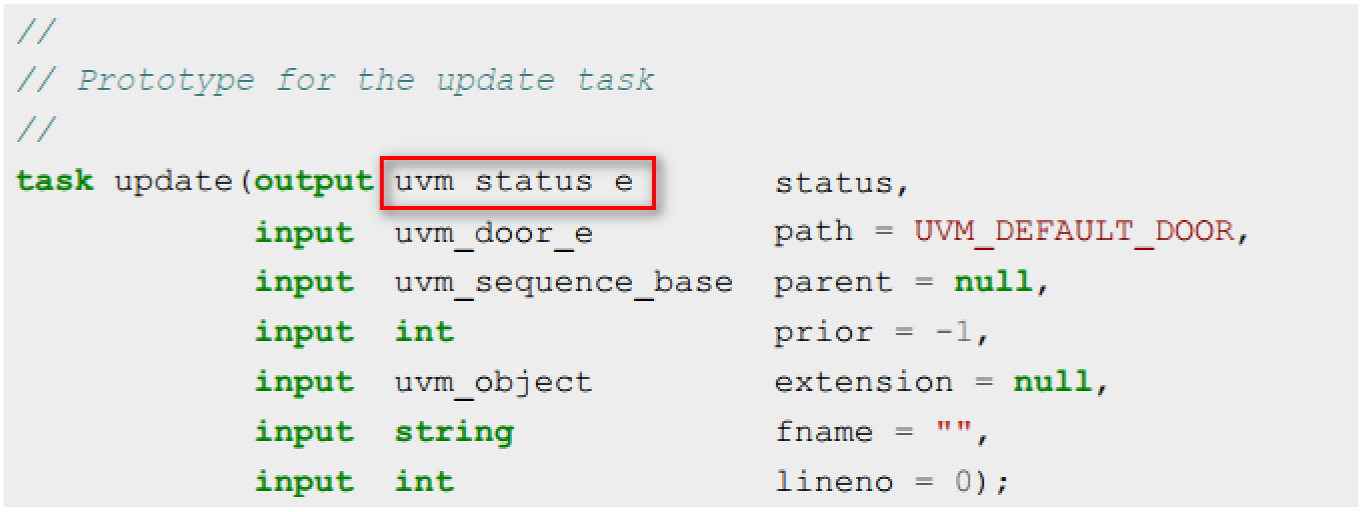
```
//
// read task prototype
//
task read(output uvm_status_e       status,
          output uvm reg data t     value,
          input  uvm_door_e         path = UVM_DEFAULT_DOOR,
          input  uvm reg map        map = null,
          input  uvm sequence base  parent = null,
          input  int                prior = -1,
          input  uvm_object         extension = null,
          input  string             fname = "",
          input  int                lineno = 0);
```

In page 328. lack of underline.



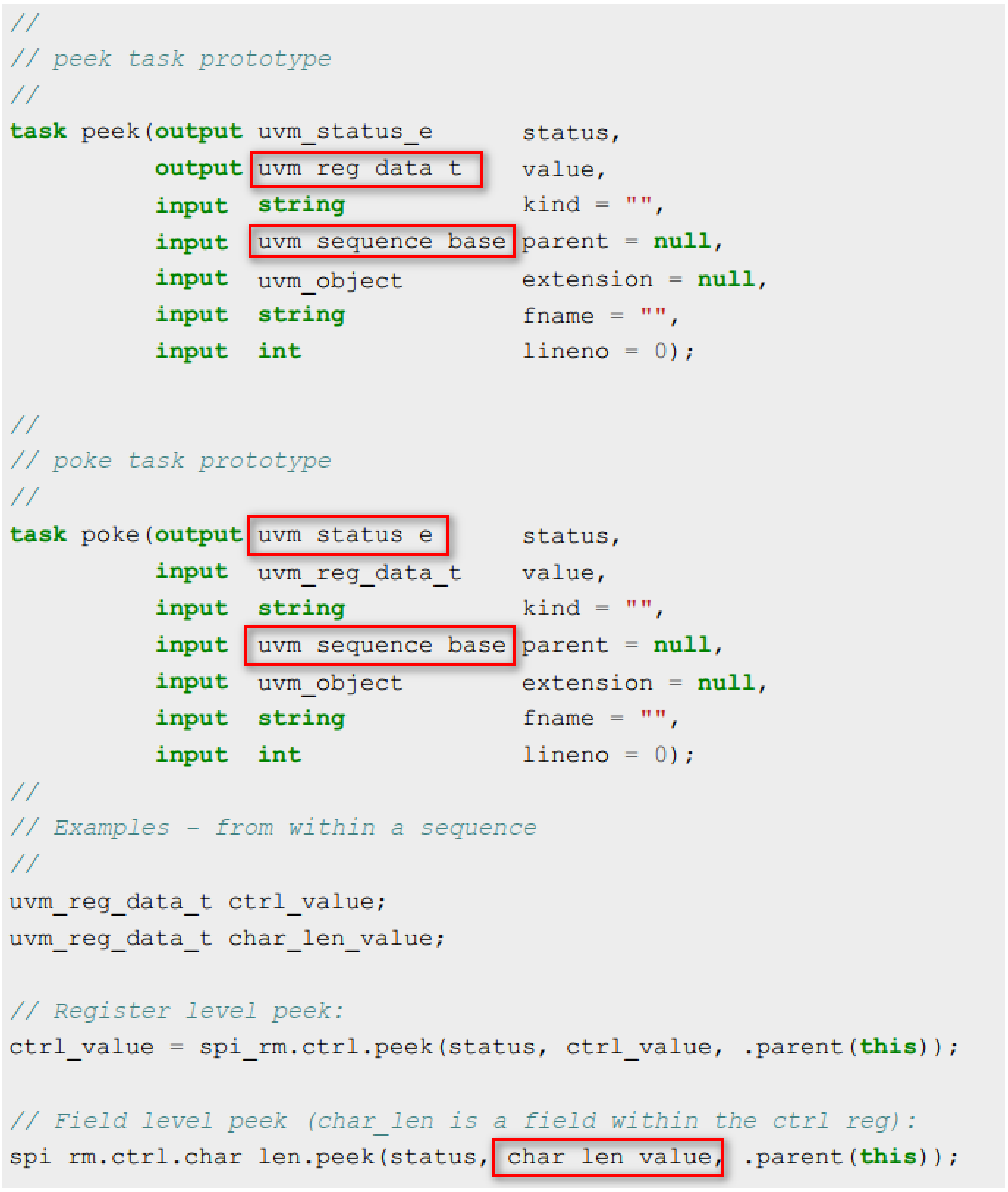
```
//
// write task prototype
//
task write(output uvm status e      status,
           input  uvm_reg_data_t    value,
           input  uvm_door_e        path = UVM_DEFAULT_DOOR,
           input  uvm reg map       map = null,
           input  uvm sequence base parent = null,
           input  int               prior = -1,
           input  uvm_object        extension = null,
           input  string            fname = "",
           input  int               lineno = 0);
```

In page 331. lack of underline.



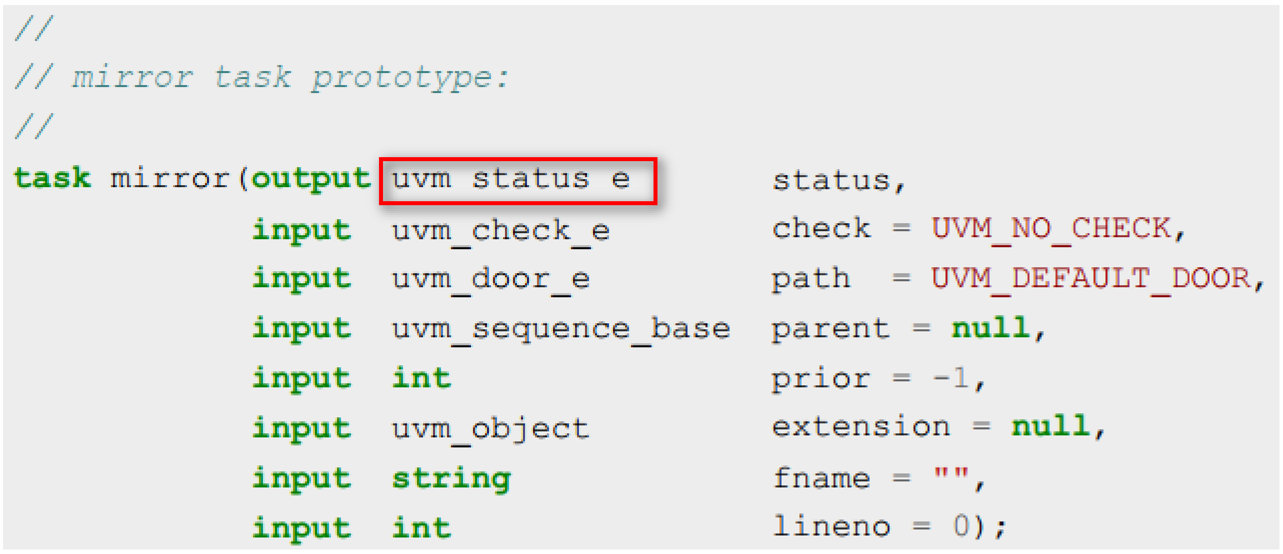
```
//
// Prototype for the update task
//
task update(output uvm status e       status,
            input  uvm_door_e         path = UVM_DEFAULT_DOOR,
            input  uvm_sequence_base  parent = null,
            input  int                prior = -1,
            input  uvm_object         extension = null,
            input  string             fname = "",
            input  int                lineno = 0);
```

In page 332. lack of underline.



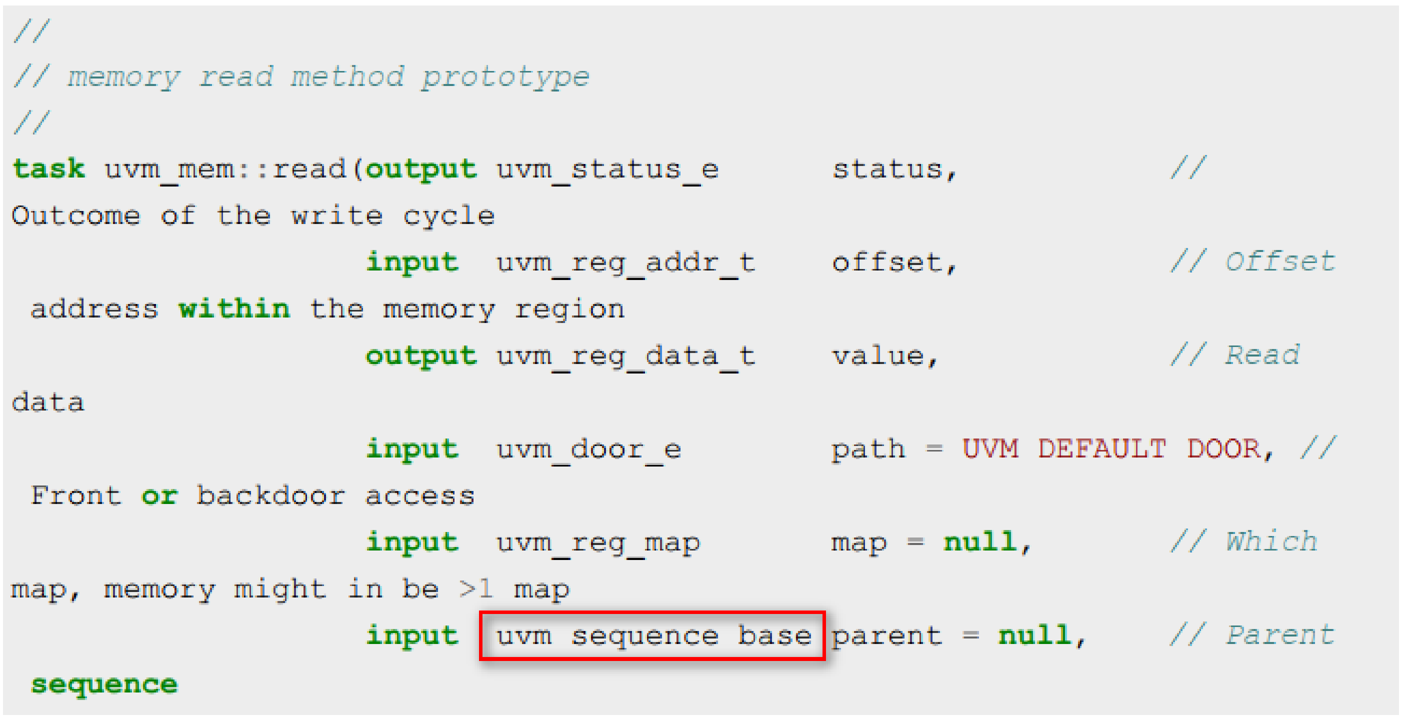
```
//
// peek task prototype
//
task peek(output uvm_status_e        status,
          output uvm reg data t      value,
          input  string              kind = "",
          input  uvm sequence base   parent = null,
          input  uvm_object          extension = null,
          input  string              fname = "",
          input  int                 lineno = 0);

//
// poke task prototype
//
task poke(output uvm status e        status,
          input  uvm_reg_data_t      value,
          input  string              kind = "",
          input  uvm sequence base   parent = null,
          input  uvm_object          extension = null,
          input  string              fname = "",
          input  int                 lineno = 0);
//
// Examples - from within a sequence
//
uvm_reg_data_t ctrl_value;
uvm_reg_data_t char_len_value;

// Register level peek:
ctrl_value = spi_rm.ctrl.peek(status, ctrl_value, .parent(this));

// Field level peek (char_len is a field within the ctrl reg):
spi rm.ctrl.char len.peek(status, char len value, .parent(this));
```

In page 333. lack of underline.



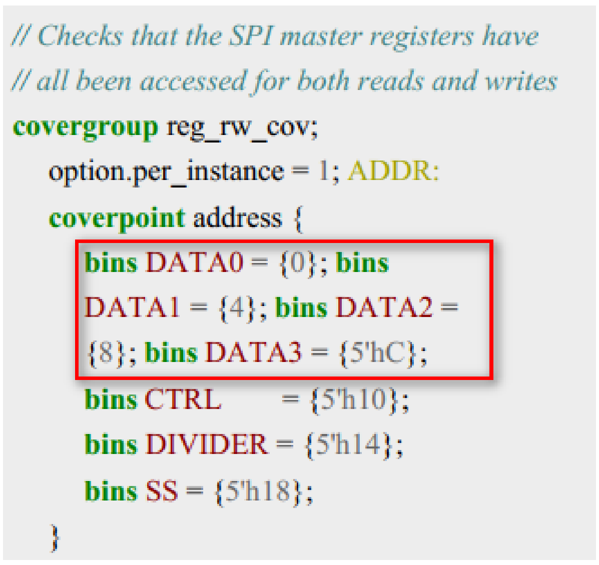
```
//
// mirror task prototype:
//
task mirror(output uvm status e       status,
            input  uvm_check_e         check = UVM_NO_CHECK,
            input  uvm_door_e          path  = UVM_DEFAULT_DOOR,
            input  uvm_sequence_base   parent = null,
            input  int                 prior = -1,
            input  uvm_object          extension = null,
            input  string              fname = "",
            input  int                 lineno = 0);
```

In page 336. lack of underline.



```
//
// memory read method prototype
//
task uvm_mem::read(output uvm_status_e      status,           //
Outcome of the write cycle
                   input  uvm_reg_addr_t    offset,           // Offset
 address within the memory region
                   output uvm_reg_data_t    value,            // Read
data
                   input  uvm_door_e        path = UVM DEFAULT DOOR, //
 Front or backdoor access
                   input  uvm_reg_map       map = null,       // Which
map, memory might in be >1 map
                   input  uvm sequence base parent = null,    // Parent
 sequence
```

In page 362. Code format is weird.



```
// Checks that the SPI master registers have
// all been accessed for both reads and writes
covergroup reg_rw_cov;
   option.per_instance = 1; ADDR:
   coverpoint address {
      bins DATA0 = {0}; bins
      DATA1 = {4}; bins DATA2 =
      {8}; bins DATA3 = {5'hC};
      bins CTRL      = {5'h10};
      bins DIVIDER = {5'h14};
      bins SS = {5'h18};
   }
```