Managing Register Side Effects and Exceptions with UVM 1.2 RAL

Steven K. Sherman

Advanced Micro Devices, Inc.

April 27, 2017

Boston

Agenda

Introduction

Register Data Structures

Register Behavior

Register Test

Conclusions

Introduction

Introduction

General Introduction

Side Effects

Exceptions

The Challenge

About Callbacks …

Introduction: General Introduction

“What about register side effects and exceptions?”

“Um … Er … Uh …” <insert clever response here>

Introduction: Side Effects

Side Effects:

General rules about how registers interact
with other registers and the DUT

Per general rules, register value change may change:

- other register accessibility
- other register field values
- design behavior
- and so forth …

Introduction: Exceptions

Exceptions:

Special register corner cases
that differ from the general rules

For example:

Register A holds a data mask for Register B
except when Register A is set to value 0x1234.

Introduction: The Challenge

- Basic register verification is simple ☺
- Side effects make it a little harder 😐
- Exceptions can complicate things ☹
- Multiply by thousands of registers … 😬

Introduction: The Challenge

- UVM RAL has many great features except …
  - No direct support for automated CSR test
  - No direct support for register side effects and exceptions
  - Room for improvement in performance
- Divide and conquer …
  - Register Data Structures
  - Register Behavior
  - Register Test

Introduction: About Callbacks …

- Elected not to bypass some UVM RAL with callbacks
- Leverage UVM RAL present and future infrastructure features:
  - Already developed and robust
  - Readily extensible and customizable
  - Readily migrated to future versions and features

Register Data Structures

Register Data Structures

Reproducibility and Associative Arrays

Register Hash Tables

Register Data Structures: Reproducibility and Associative Arrays

- Register object "hash tables" may be associative arrays or queues (dynamic arrays).
- SystemVerilog
  - Associative arrays preferred over queues when size unknown
- Mantis 4924
  - Associative arrays not order-guaranteed
  - Some register "get" functions have unpredictable order!
- `uvm_reg_block::get_registers` returns list using associative array:

```
        local int unsigned   regs[uvm_reg];
```

- May need to sort results for reproducibility

Register Data Structures: Register Hash Tables

- Register models become huge and complex.
- Searches through the register model are expensive.
- Hash tables can save search results for reuse:
  - my_regs_name
  - my_regs_comp_name

Register Data Structures: Register Hash Tables

- Hash table access is usually faster than repeating a search …
  - BUT NOT ALWAYS!
  - A hash register name string search may be needed to see if a register model search is required.
  - But, string tests of large hashes may be slow!

Register Data Structures: Register Hash Tables

- Based on informal testing
  - **Speculate** that string indices are to blame.
  - Currently encouraging/enforcing efficient register handle reuse:
    - sequences
    - scoreboards
    - and so forth …

Register Data Structures: Register Hash Tables

my_regs_name – all registers indexed by interface and register name:

```
local uvm_reg my_regs_name[interface_e][string][$];
```

Helpful when accessing all registers with same name and interface.

Queue, so no reproducibility issues.

Register Data Structures: Register Hash Tables

my_regs_comp_name – all registers indexed by register component and name:

```
local uvm_reg my_regs_comp_name[int][string];
```

Helpful with registers with same names but different components.

Associative array, but should be only one unique register by name per component.

Register Behavior

Register Behavior

Reset Strapping

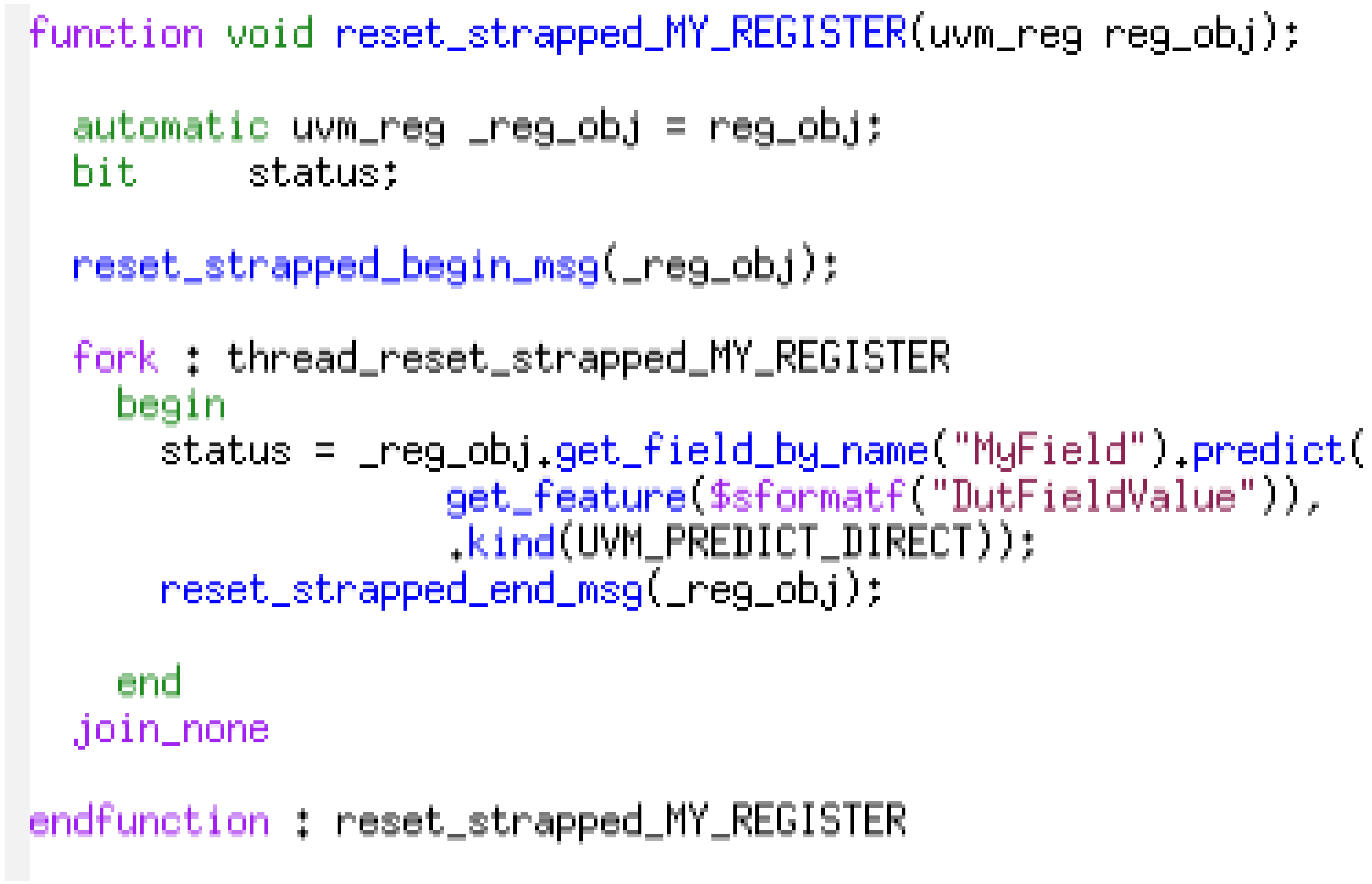
Enabling

Locking

Broadcast

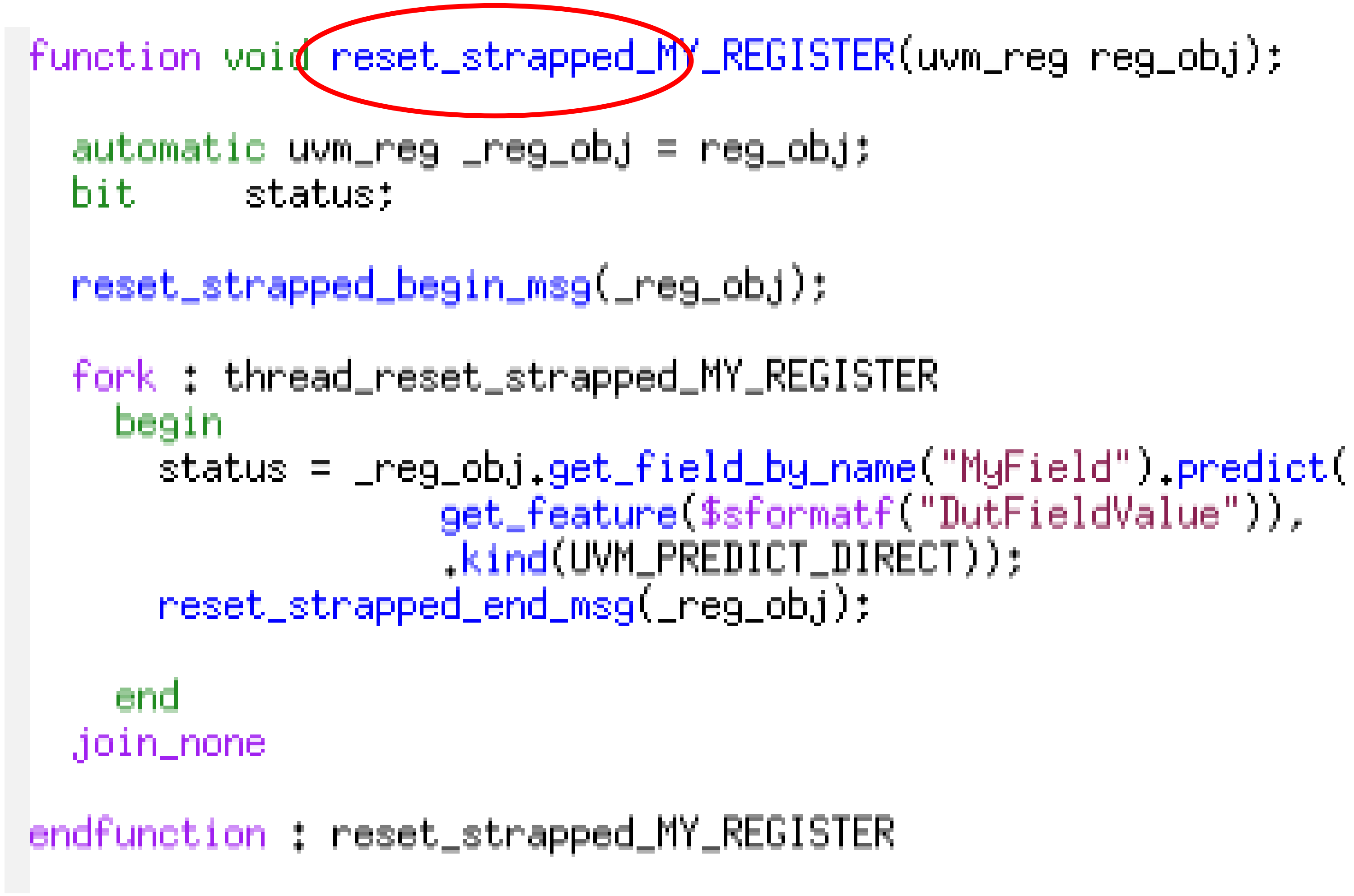
Block Aliasing

Register Behavior: Reset Strapping

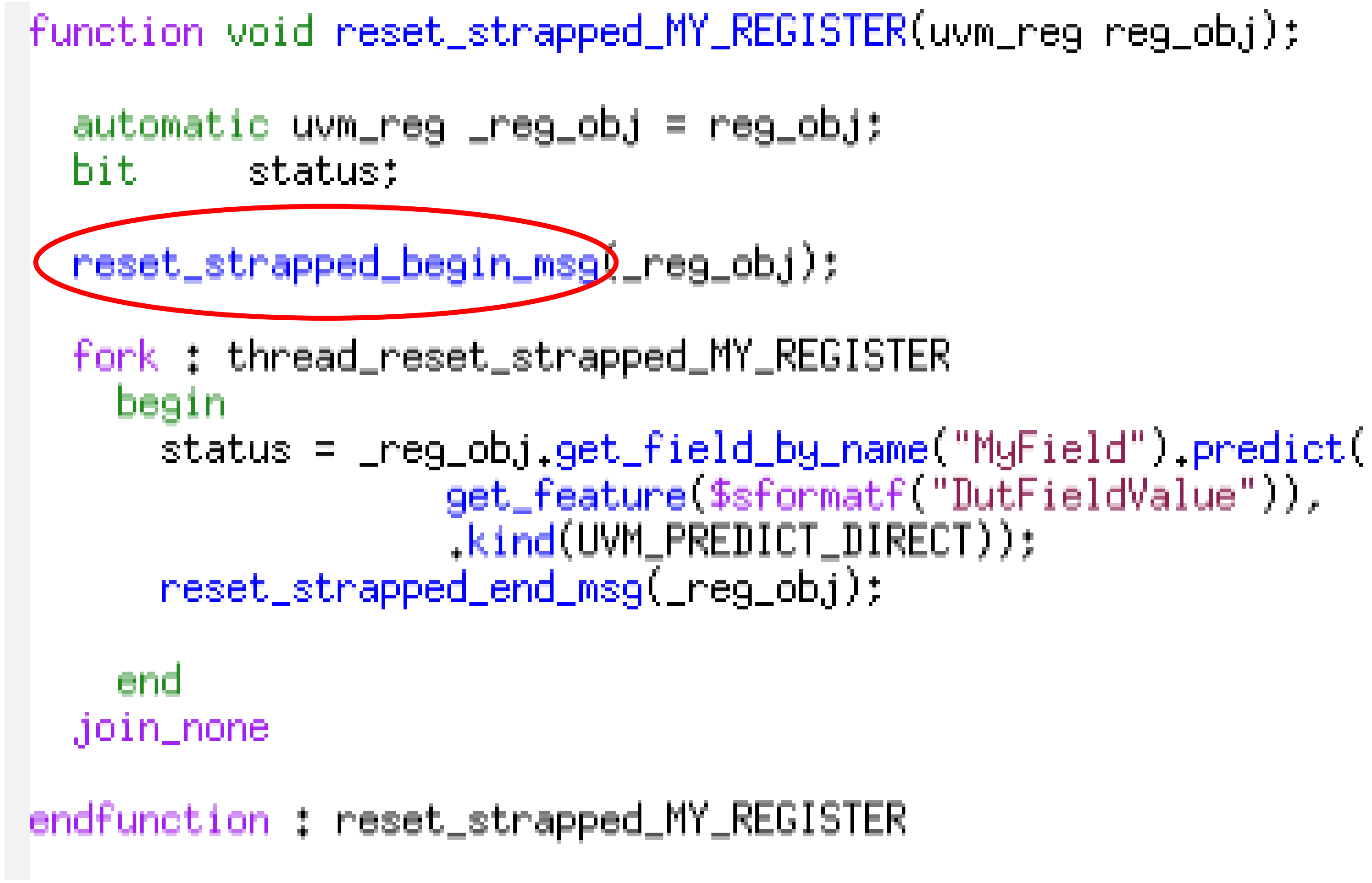


```
function void reset_strapped_MY_REGISTER(uvm_reg reg_obj);

  automatic uvm_reg _reg_obj = reg_obj;
  bit       status;

  reset_strapped_begin_msg(_reg_obj);

  fork : thread_reset_strapped_MY_REGISTER
    begin
      status = _reg_obj.get_field_by_name("MyField").predict(
                   get_feature($sformatf("DutFieldValue")),
                   .kind(UVM_PREDICT_DIRECT));
      reset_strapped_end_msg(_reg_obj);

    end
  join_none

endfunction : reset_strapped_MY_REGISTER
```

Register Behavior: Reset Strapping

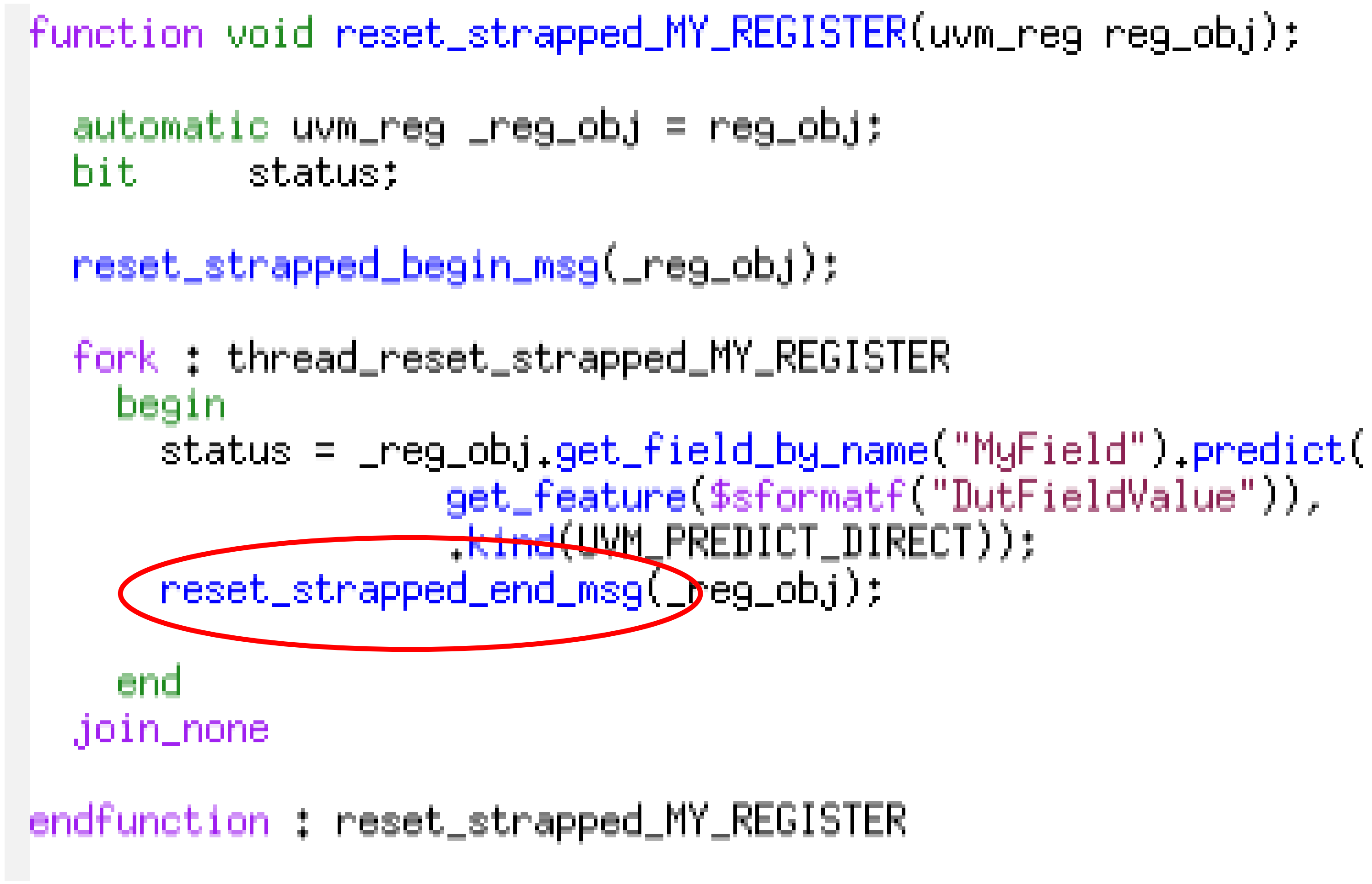

```
function void reset_strapped_MY_REGISTER(uvm_reg reg_obj);

  automatic uvm_reg _reg_obj = reg_obj;
  bit       status;

  reset_strapped_begin_msg(_reg_obj);

  fork : thread_reset_strapped_MY_REGISTER
    begin
      status = _reg_obj.get_field_by_name("MyField").predict(
                           get_feature($sformatf("DutFieldValue")),
                          .kind(UVM_PREDICT_DIRECT));
      reset_strapped_end_msg(_reg_obj);

    end
  join_none

endfunction : reset_strapped_MY_REGISTER
```

Register Behavior: Reset Strapping



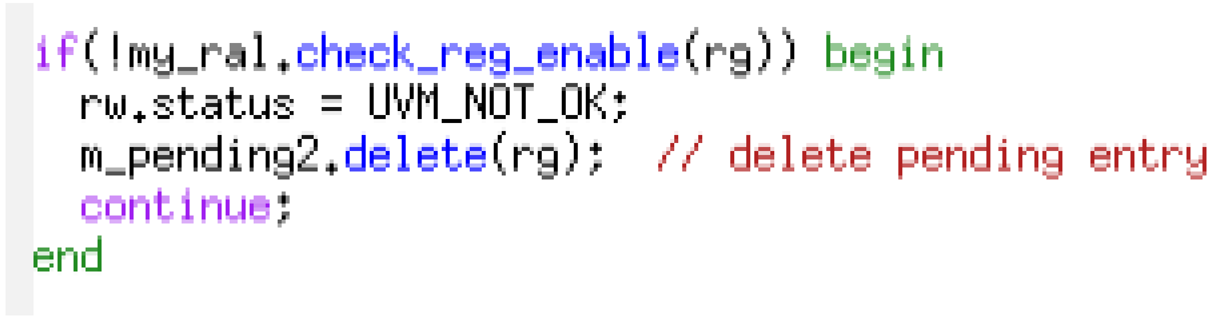
```
function void reset_strapped_MY_REGISTER(uvm_reg reg_obj);

  automatic uvm_reg _reg_obj = reg_obj;
  bit     status;

  reset_strapped_begin_msg(_reg_obj);

  fork : thread_reset_strapped_MY_REGISTER
    begin
      status = _reg_obj.get_field_by_name("MyField").predict(
                   get_feature($sformatf("DutFieldValue")),
                   .kind(UVM_PREDICT_DIRECT));
      reset_strapped_end_msg(_reg_obj);

    end
  join_none

endfunction : reset_strapped_MY_REGISTER
```

Register Behavior: Reset Strapping



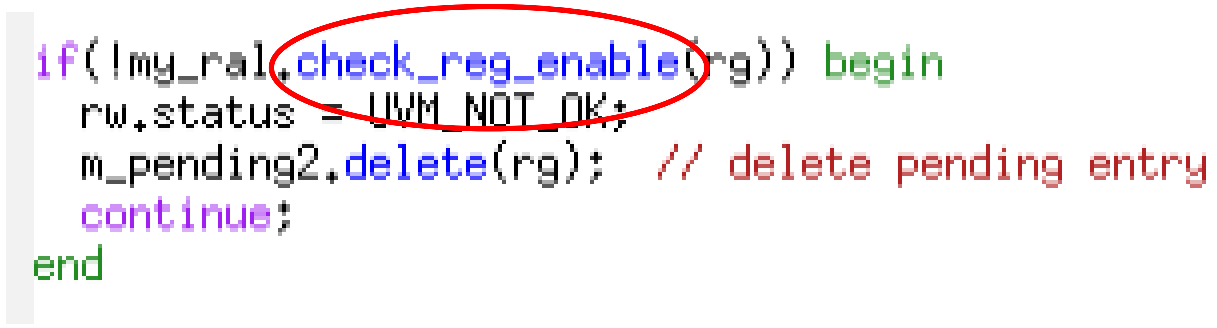
```
function void reset_strapped_MY_REGISTER(uvm_reg reg_obj);

  automatic uvm_reg _reg_obj = reg_obj;
  bit      status;

  reset_strapped_begin_msg(_reg_obj);

  fork : thread_reset_strapped_MY_REGISTER
    begin
      status = _reg_obj.get_field_by_name("MyField").predict(
                    get_feature($sformatf("DutFieldValue")),
                    .kind(UVM_PREDICT_DIRECT));
      reset_strapped_end_msg(_reg_obj);

    end
  join_none

endfunction : reset_strapped_MY_REGISTER
```

Register Behavior: Enabling



```
if(!my_ral.check_reg_enable(rg)) begin
  rw.status = UVM_NOT_OK;
  m_pending2.delete(rg);  // delete pending entry
  continue;
end
```

Register Behavior: Enabling

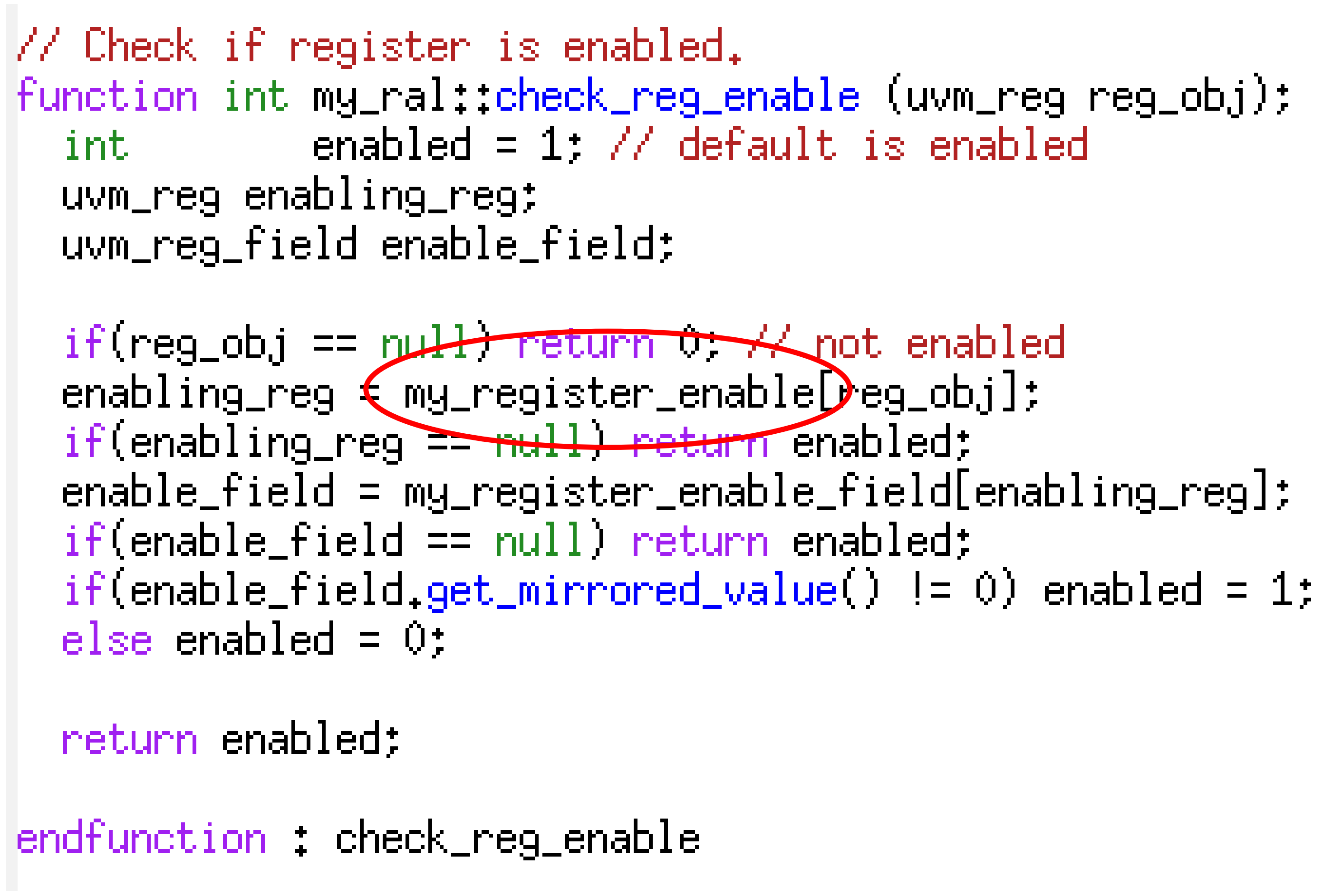


```
if(!my_ral.check_reg_enable(rg)) begin
  rw.status = UVM_NOT_OK;
  m_pending2.delete(rg);  // delete pending entry
  continue;
end
```

Register Behavior: Enabling

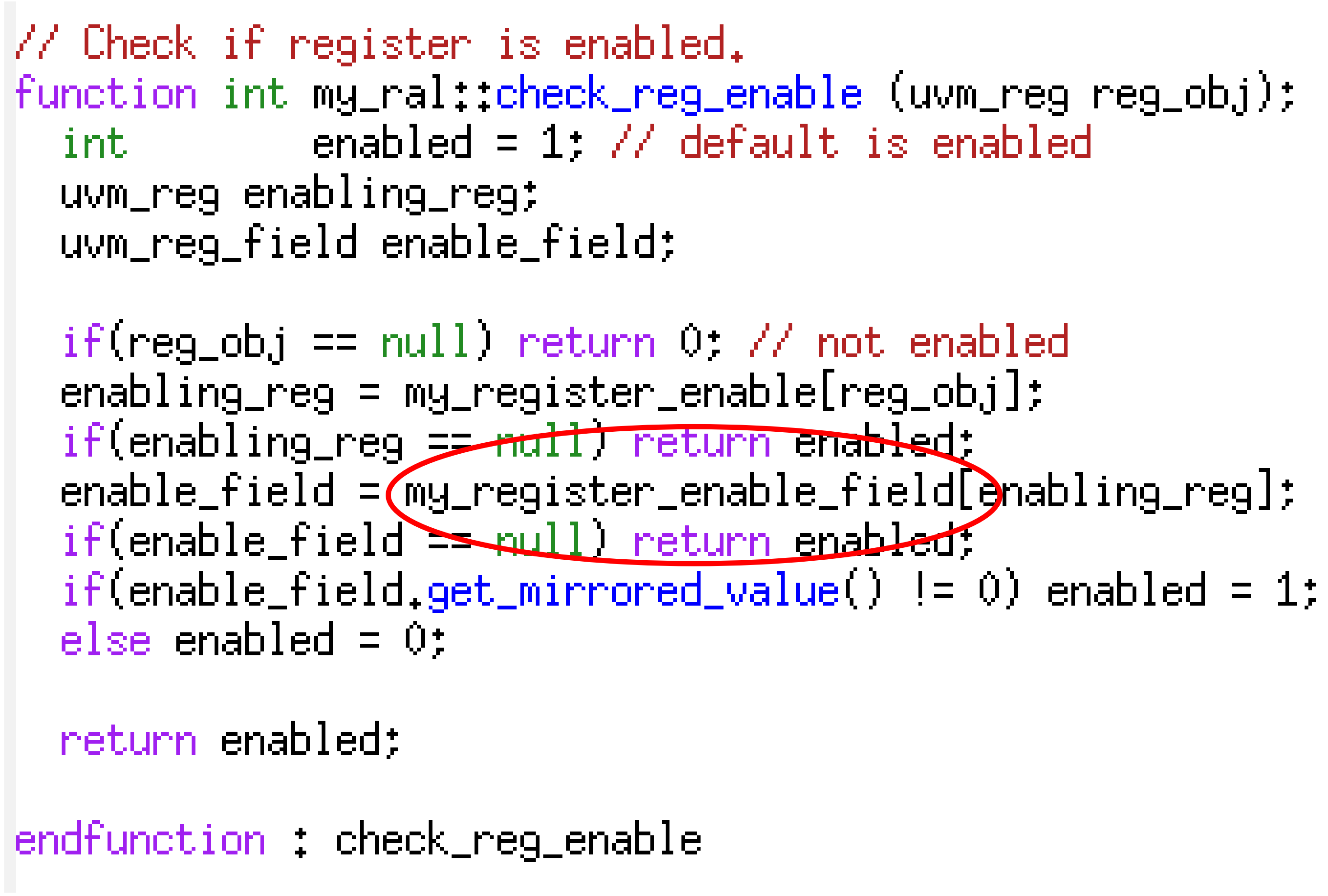
```
// Check if register is enabled.
function int my_ral::check_reg_enable (uvm_reg reg_obj);
  int         enabled = 1; // default is enabled
  uvm_reg enabling_reg;
  uvm_reg_field enable_field;

  if(reg_obj == null) return 0; // not enabled
  enabling_reg = my_register_enable[reg_obj];
  if(enabling_reg == null) return enabled;
  enable_field = my_register_enable_field[enabling_reg];
  if(enable_field == null) return enabled;
  if(enable_field.get_mirrored_value() != 0) enabled = 1;
  else enabled = 0;

  return enabled;

endfunction : check_reg_enable
```

Register Behavior: Enabling



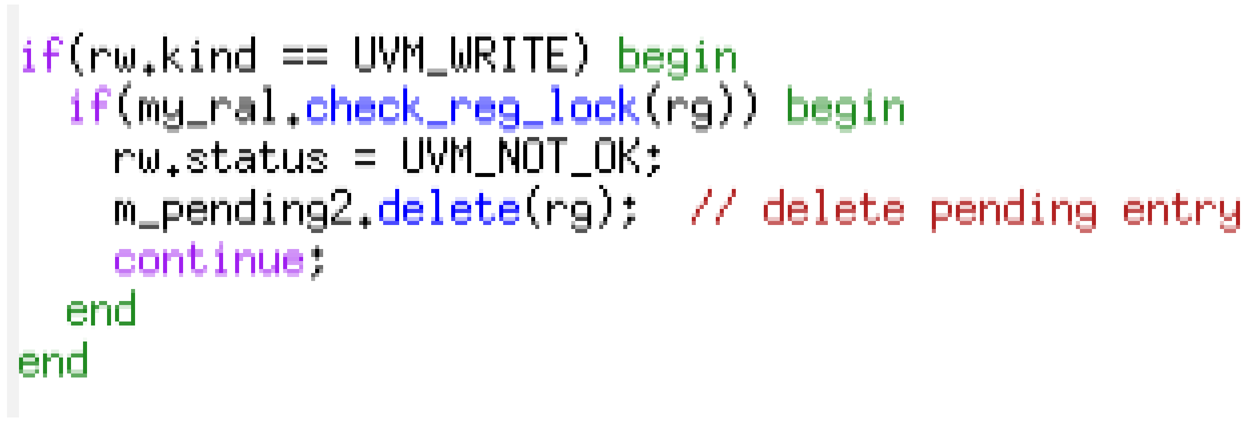
```
// Check if register is enabled.
function int my_ral::check_reg_enable (uvm_reg reg_obj);
  int        enabled = 1; // default is enabled
  uvm_reg enabling_reg;
  uvm_reg_field enable_field;

  if(reg_obj == null) return 0; // not enabled
  enabling_reg = my_register_enable[reg_obj];
  if(enabling_reg == null) return enabled;
  enable_field = my_register_enable_field[enabling_reg];
  if(enable_field == null) return enabled;
  if(enable_field.get_mirrored_value() != 0) enabled = 1;
  else enabled = 0;

  return enabled;

endfunction : check_reg_enable
```

Register Behavior: Enabling



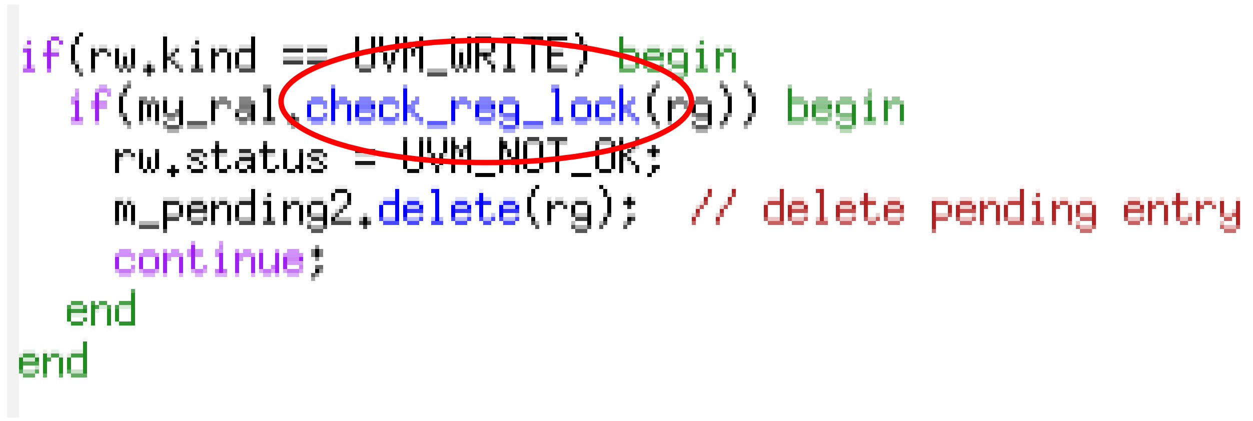
```
// Check if register is enabled.
function int my_ral::check_reg_enable (uvm_reg reg_obj);
  int        enabled = 1; // default is enabled
  uvm_reg enabling_reg;
  uvm_reg_field enable_field;

  if(reg_obj == null) return 0; // not enabled
  enabling_reg = my_register_enable[reg_obj];
  if(enabling_reg == null) return enabled;
  enable_field = my_register_enable_field[enabling_reg];
  if(enable_field == null) return enabled;
  if(enable_field.get_mirrored_value() != 0) enabled = 1;
  else enabled = 0;

  return enabled;

endfunction : check_reg_enable
```

Register Behavior: Locking



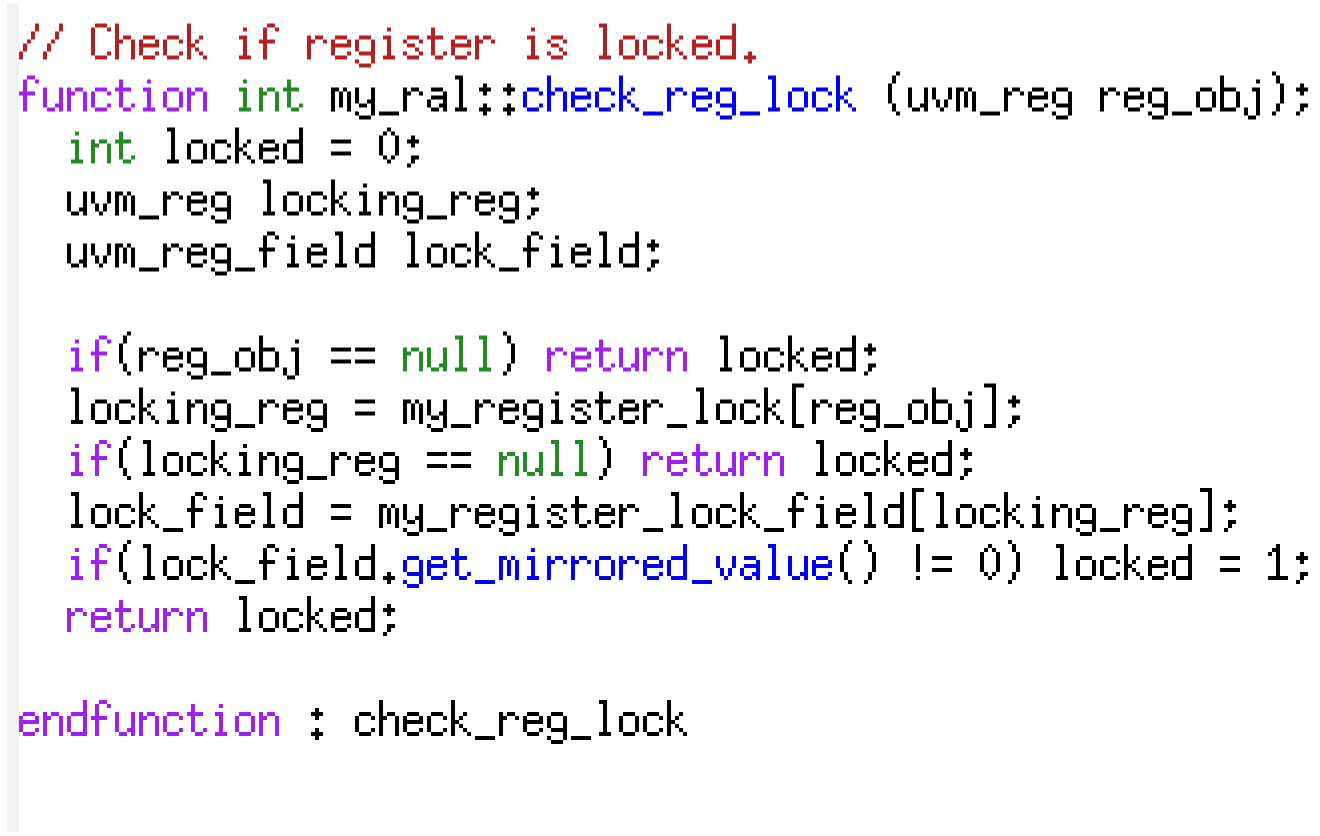
```
if(rw.kind == UVM_WRITE) begin
  if(my_ral.check_reg_lock(rg)) begin
    rw.status = UVM_NOT_OK;
    m_pending2.delete(rg);  // delete pending entry
    continue;
  end
end
```

Register Behavior: Locking



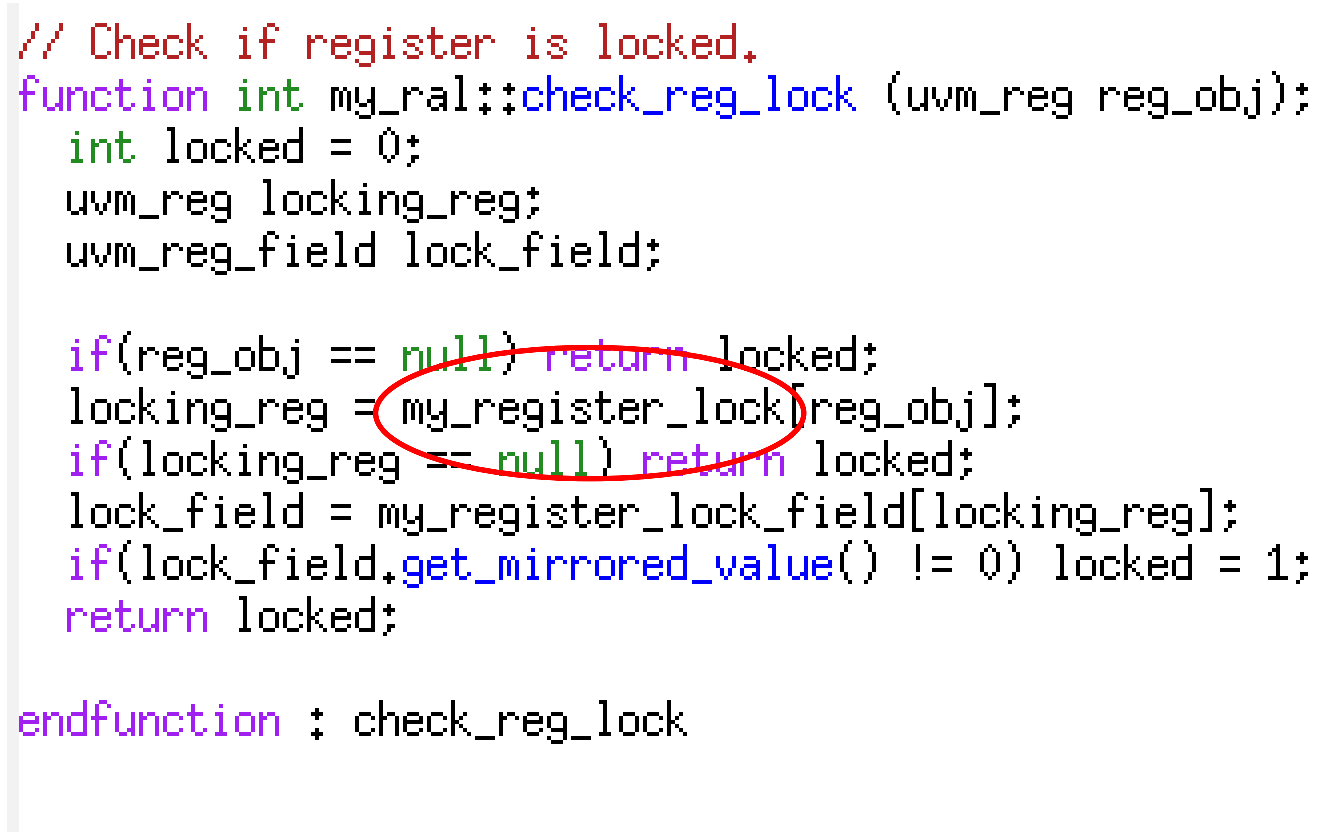
```
if(rw.kind == UVM_WRITE) begin
  if(my_ral.check_reg_lock(rg)) begin
    rw.status = UVM_NOT_OK;
    m_pending2.delete(rg);  // delete pending entry
    continue;
  end
end
```

Register Behavior: Locking



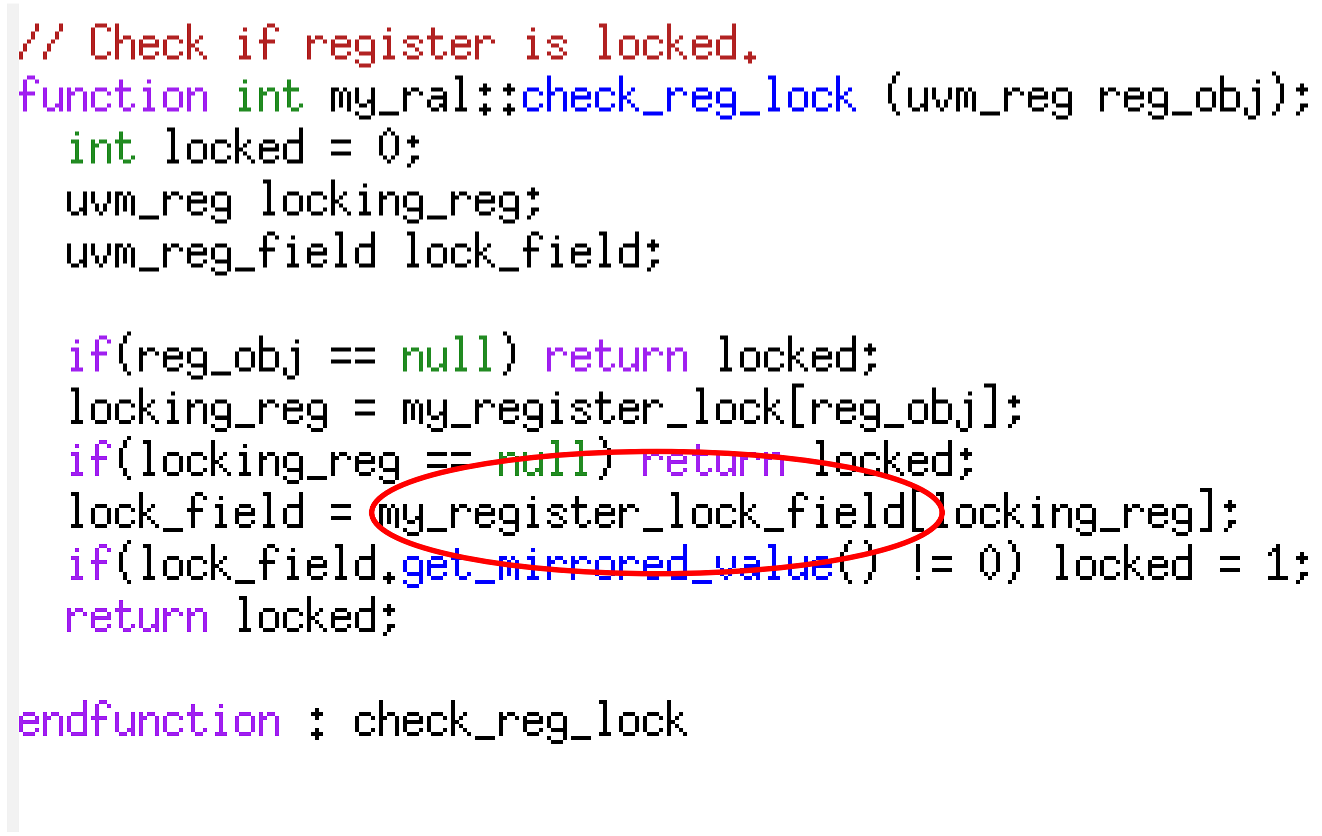
```
// Check if register is locked.
function int my_ral::check_reg_lock (uvm_reg reg_obj);
  int locked = 0;
  uvm_reg locking_reg;
  uvm_reg_field lock_field;

  if(reg_obj == null) return locked;
  locking_reg = my_register_lock[reg_obj];
  if(locking_reg == null) return locked;
  lock_field = my_register_lock_field[locking_reg];
  if(lock_field.get_mirrored_value() != 0) locked = 1;
  return locked;

endfunction : check_reg_lock
```

Register Behavior: Locking



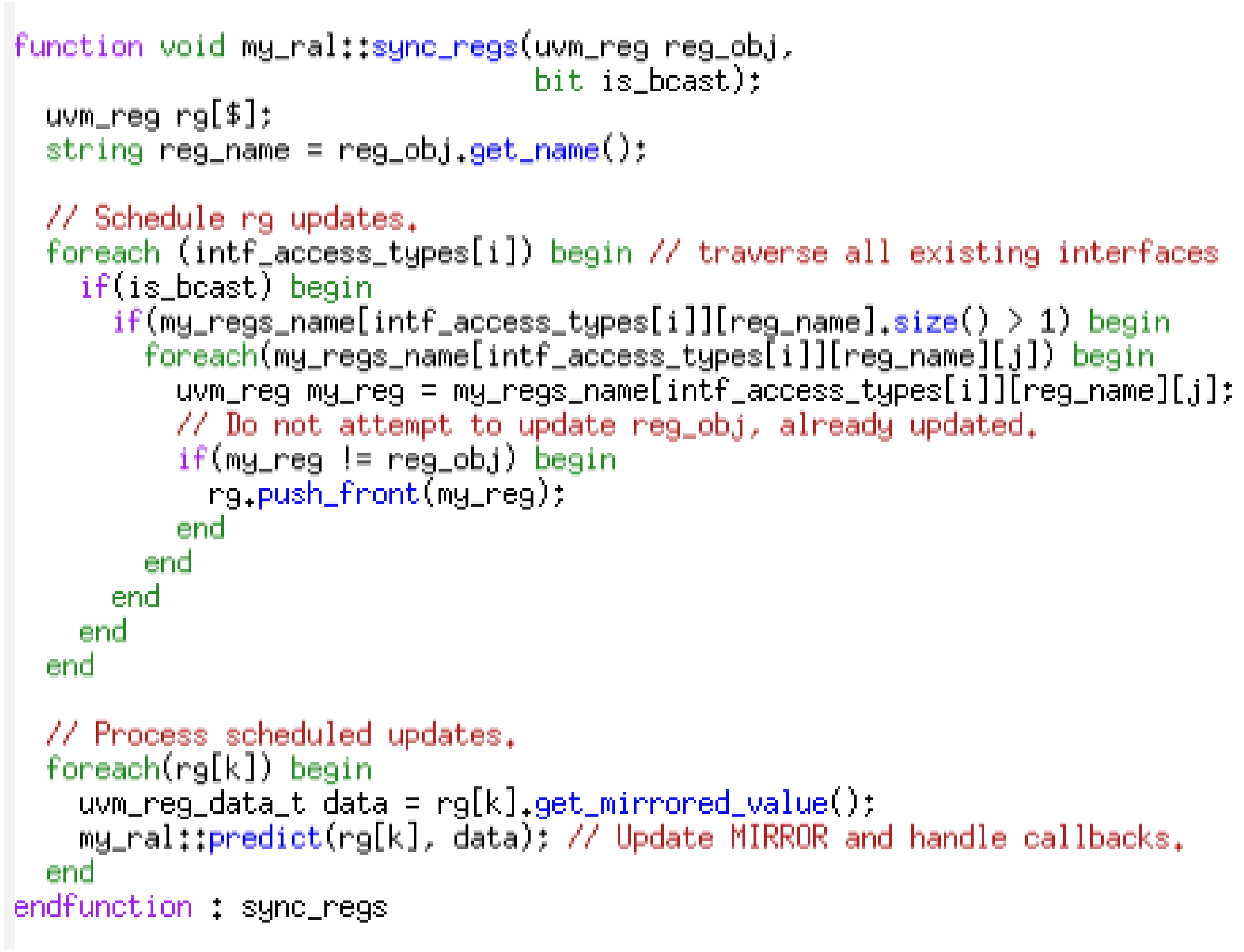
```
// Check if register is locked.
function int my_ral::check_reg_lock (uvm_reg reg_obj);
  int locked = 0;
  uvm_reg locking_reg;
  uvm_reg_field lock_field;

  if(reg_obj == null) return locked;
  locking_reg = my_register_lock[reg_obj];
  if(locking_reg == null) return locked;
  lock_field = my_register_lock_field[locking_reg];
  if(lock_field.get_mirrored_value() != 0) locked = 1;
  return locked;

endfunction : check_reg_lock
```

Register Behavior: Locking



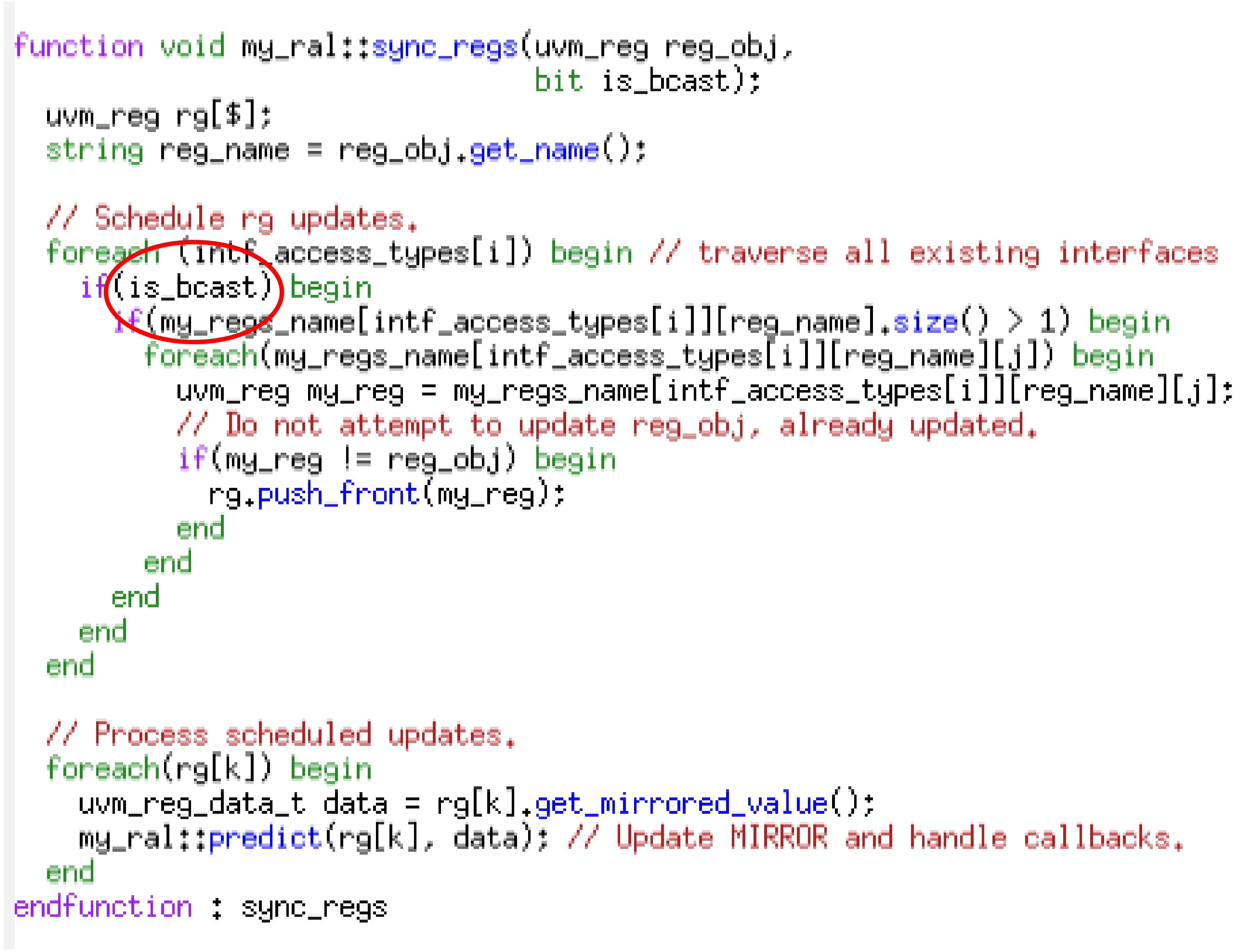
```
// Check if register is locked.
function int my_ral::check_reg_lock (uvm_reg reg_obj);
  int locked = 0;
  uvm_reg locking_reg;
  uvm_reg_field lock_field;

  if(reg_obj == null) return locked;
  locking_reg = my_register_lock[reg_obj];
  if(locking_reg == null) return locked;
  lock_field = my_register_lock_field[locking_reg];
  if(lock_field.get_mirrored_value() != 0) locked = 1;
  return locked;

endfunction : check_reg_lock
```

Register Behavior: Broadcast



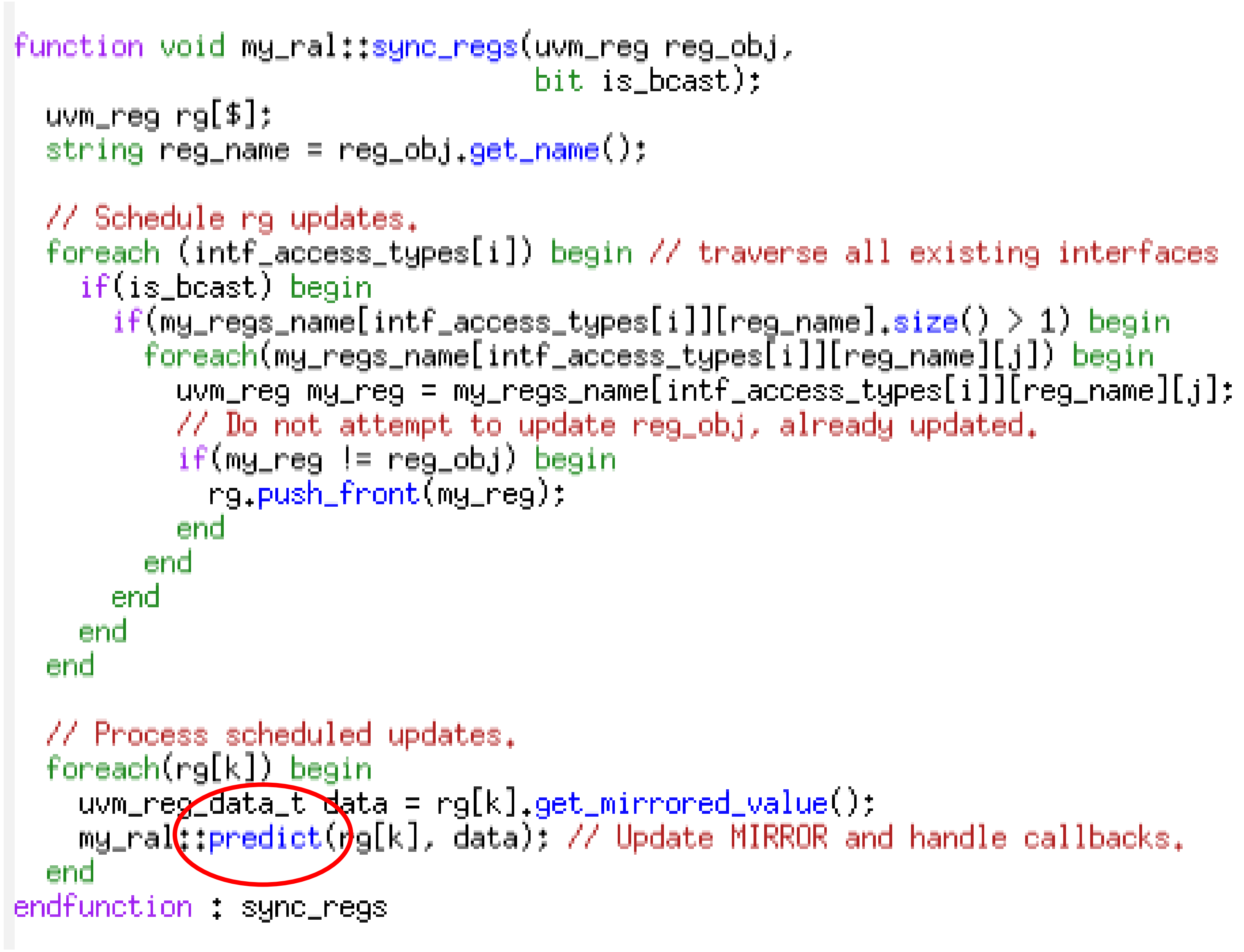
```
function void my_ral::sync_regs(uvm_reg reg_obj,
                                bit is_bcast);
  uvm_reg rg[$];
  string reg_name = reg_obj.get_name();

  // Schedule rg updates.
  foreach (intf_access_types[i]) begin // traverse all existing interfaces
    if(is_bcast) begin
      if(my_regs_name[intf_access_types[i]][reg_name].size() > 1) begin
        foreach(my_regs_name[intf_access_types[i]][reg_name][j]) begin
          uvm_reg my_reg = my_regs_name[intf_access_types[i]][reg_name][j];
          // Do not attempt to update reg_obj, already updated.
          if(my_reg != reg_obj) begin
            rg.push_front(my_reg);
          end
        end
      end
    end
  end

  // Process scheduled updates.
  foreach(rg[k]) begin
    uvm_reg_data_t data = rg[k].get_mirrored_value();
    my_ral::predict(rg[k], data); // Update MIRROR and handle callbacks.
  end
endfunction : sync_regs
```

Register Behavior: Broadcast



```
function void my_ral::sync_regs(uvm_reg reg_obj,
                                bit is_bcast);
  uvm_reg rg[$];
  string reg_name = reg_obj.get_name();

  // Schedule rg updates.
  foreach (intf_access_types[i]) begin // traverse all existing interfaces
    if(is_bcast) begin
      if(my_regs_name[intf_access_types[i]][reg_name].size() > 1) begin
        foreach(my_regs_name[intf_access_types[i]][reg_name][j]) begin
          uvm_reg my_reg = my_regs_name[intf_access_types[i]][reg_name][j];
          // Do not attempt to update reg_obj, already updated.
          if(my_reg != reg_obj) begin
            rg.push_front(my_reg);
          end
        end
      end
    end
  end

  // Process scheduled updates.
  foreach(rg[k]) begin
    uvm_reg_data_t data = rg[k].get_mirrored_value();
    my_ral::predict(rg[k], data); // Update MIRROR and handle callbacks.
  end
endfunction : sync_regs
```

Register Behavior: Broadcast



```
function void my_ral::sync_regs(uvm_reg reg_obj,
                                bit is_bcast);
  uvm_reg rg[$];
  string reg_name = reg_obj.get_name();

  // Schedule rg updates.
  foreach (intf_access_types[i]) begin // traverse all existing interfaces
    if(is_bcast) begin
      if(my_regs_name[intf_access_types[i]][reg_name].size() > 1) begin
        foreach(my_regs_name[intf_access_types[i]][reg_name][j]) begin
          uvm_reg my_reg = my_regs_name[intf_access_types[i]][reg_name][j];
          // Do not attempt to update reg_obj, already updated.
          if(my_reg != reg_obj) begin
            rg.push_front(my_reg);
          end
        end
      end
    end
  end

  // Process scheduled updates.
  foreach(rg[k]) begin
    uvm_reg_data_t data = rg[k].get_mirrored_value();
    my_ral::predict(rg[k], data); // Update MIRROR and handle callbacks.
  end
endfunction : sync_regs
```

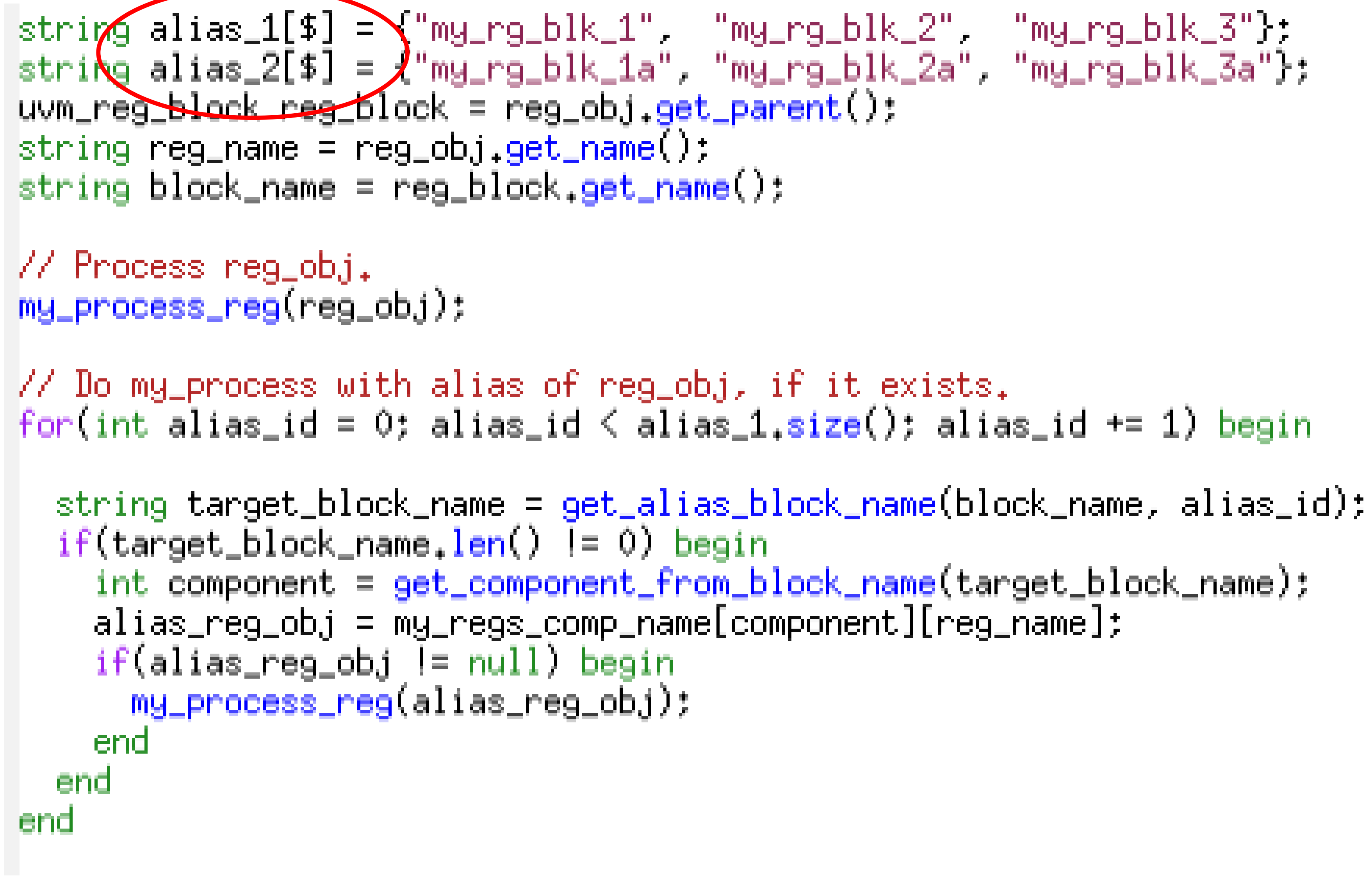
Register Behavior: Block Aliasing

- Block Aliasing:
  - Separate UVM blocks with registers of matching name
- Register write:
  - Matching registers are updated in all aliased blocks
  - May have different fields updated
- Register read:
  - All registers OR’d for read value
  - Registers may have enables and locks to consider

Register Behavior: Block Aliasing

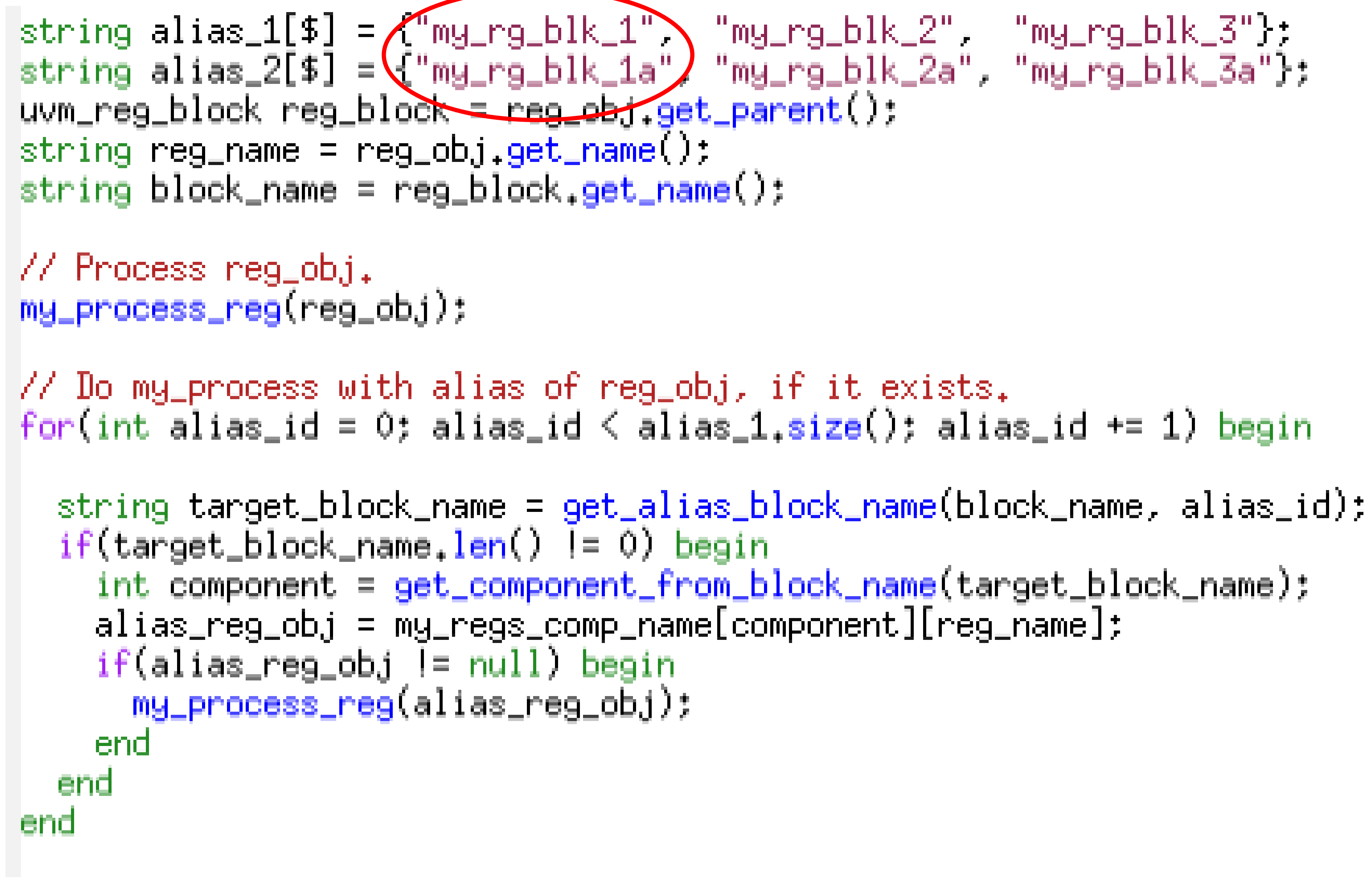
```
string alias_1[$] = {"my_rg_blk_1",  "my_rg_blk_2",  "my_rg_blk_3"};
string alias_2[$] = {"my_rg_blk_1a", "my_rg_blk_2a", "my_rg_blk_3a"};
uvm_reg_block reg_block = reg_obj.get_parent();
string reg_name = reg_obj.get_name();
string block_name = reg_block.get_name();

// Process reg_obj.
my_process_reg(reg_obj);

// Do my_process with alias of reg_obj, if it exists.
for(int alias_id = 0; alias_id < alias_1.size(); alias_id += 1) begin

  string target_block_name = get_alias_block_name(block_name, alias_id);
  if(target_block_name.len() != 0) begin
    int component = get_component_from_block_name(target_block_name);
    alias_reg_obj = my_regs_comp_name[component][reg_name];
    if(alias_reg_obj != null) begin
      my_process_reg(alias_reg_obj);
    end
  end
end
```

Register Behavior: Block Aliasing



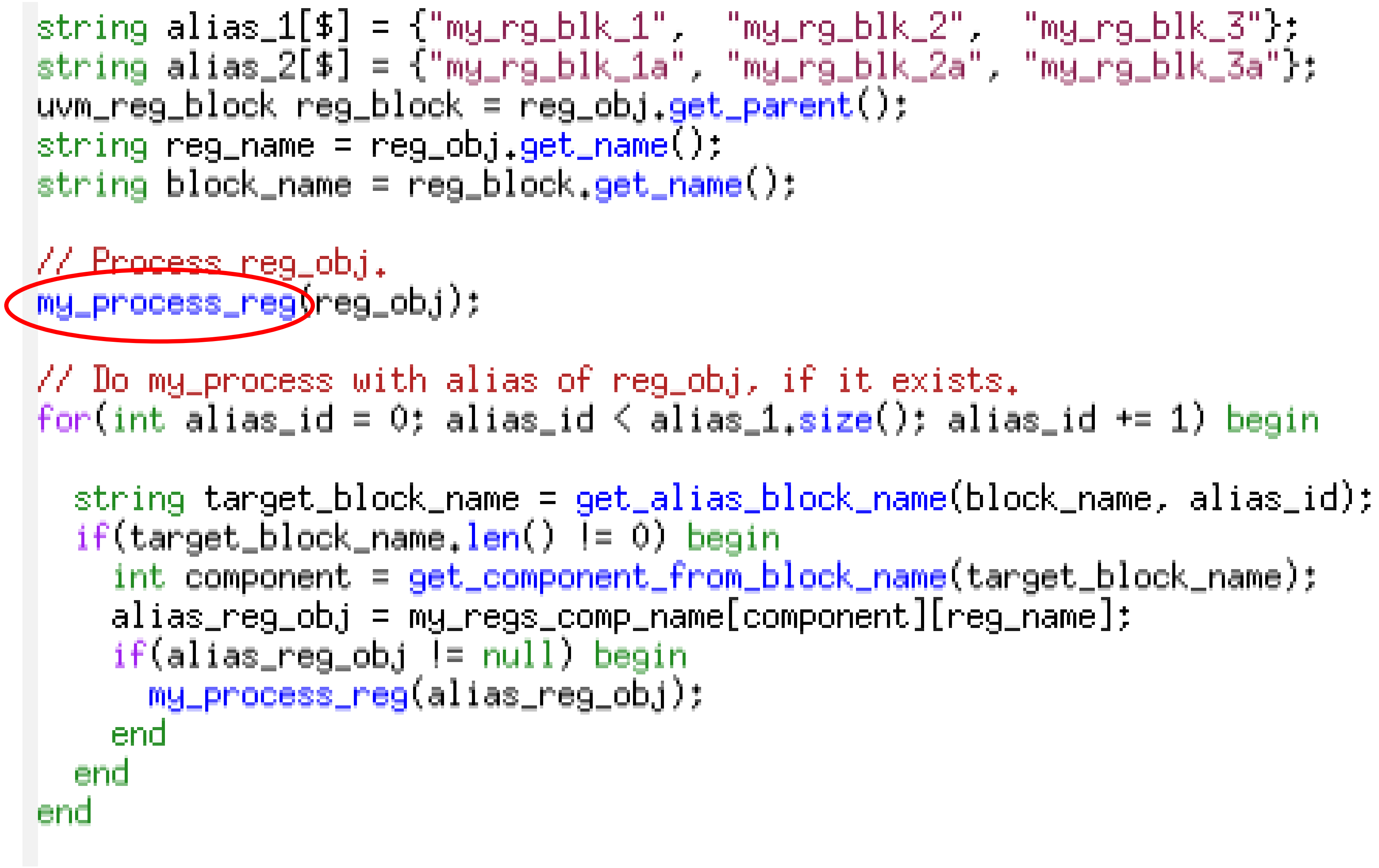
```
string alias_1[$] = {"my_rg_blk_1",  "my_rg_blk_2",  "my_rg_blk_3"};
string alias_2[$] = {"my_rg_blk_1a", "my_rg_blk_2a", "my_rg_blk_3a"};
uvm_reg_block reg_block = reg_obj.get_parent();
string reg_name = reg_obj.get_name();
string block_name = reg_block.get_name();

// Process reg_obj.
my_process_reg(reg_obj);

// Do my_process with alias of reg_obj, if it exists.
for(int alias_id = 0; alias_id < alias_1.size(); alias_id += 1) begin

  string target_block_name = get_alias_block_name(block_name, alias_id);
  if(target_block_name.len() != 0) begin
    int component = get_component_from_block_name(target_block_name);
    alias_reg_obj = my_regs_comp_name[component][reg_name];
    if(alias_reg_obj != null) begin
      my_process_reg(alias_reg_obj);
    end
  end
end
```

Register Behavior: Block Aliasing



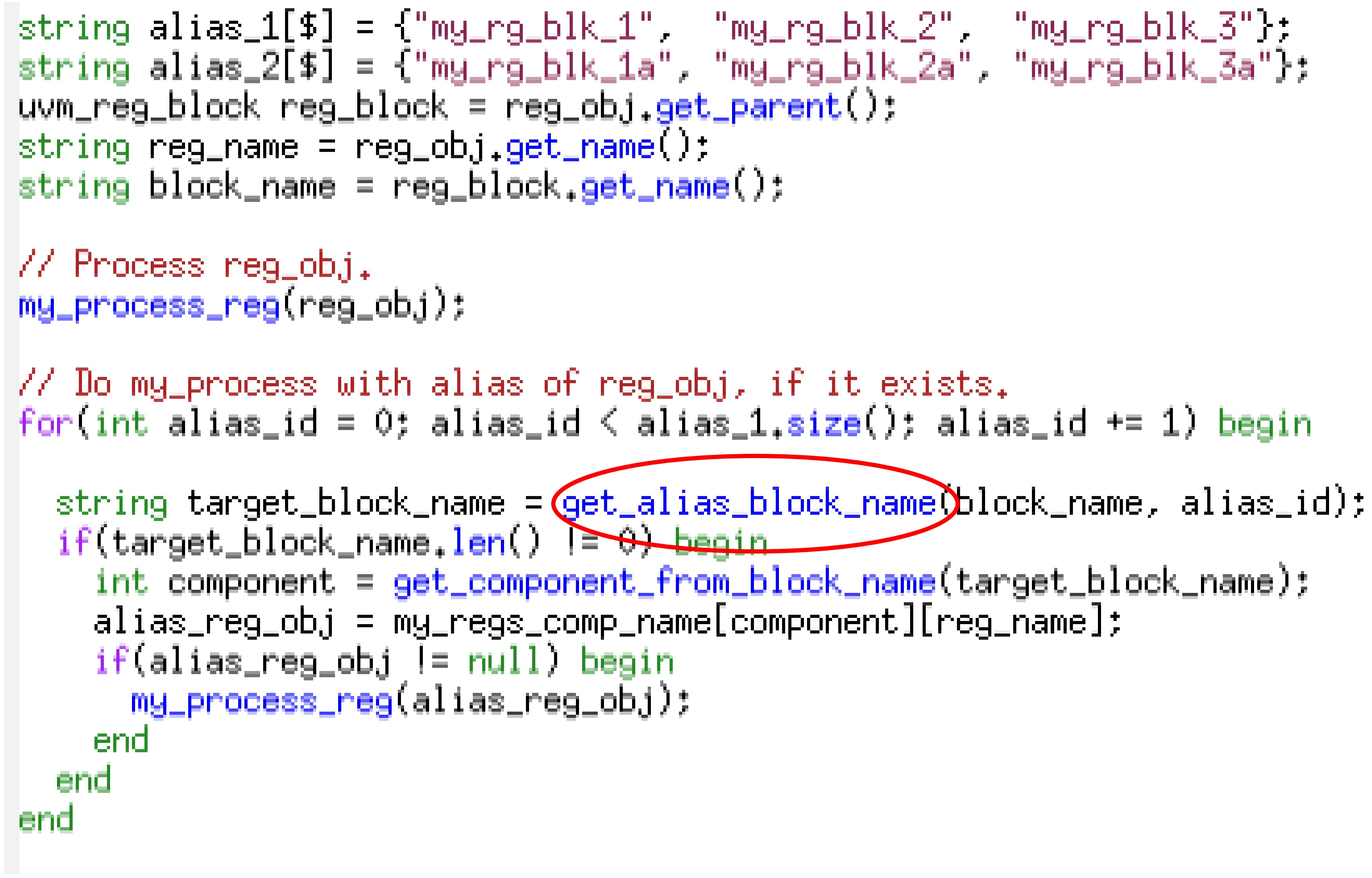
```
string alias_1[$] = {"my_rg_blk_1",  "my_rg_blk_2",  "my_rg_blk_3"};
string alias_2[$] = {"my_rg_blk_1a", "my_rg_blk_2a", "my_rg_blk_3a"};
uvm_reg_block reg_block = reg_obj.get_parent();
string reg_name = reg_obj.get_name();
string block_name = reg_block.get_name();

// Process reg_obj.
my_process_reg(reg_obj);

// Do my_process with alias of reg_obj, if it exists.
for(int alias_id = 0; alias_id < alias_1.size(); alias_id += 1) begin

  string target_block_name = get_alias_block_name(block_name, alias_id);
  if(target_block_name.len() != 0) begin
    int component = get_component_from_block_name(target_block_name);
    alias_reg_obj = my_regs_comp_name[component][reg_name];
    if(alias_reg_obj != null) begin
      my_process_reg(alias_reg_obj);
    end
  end
end
```

Register Behavior: Block Aliasing



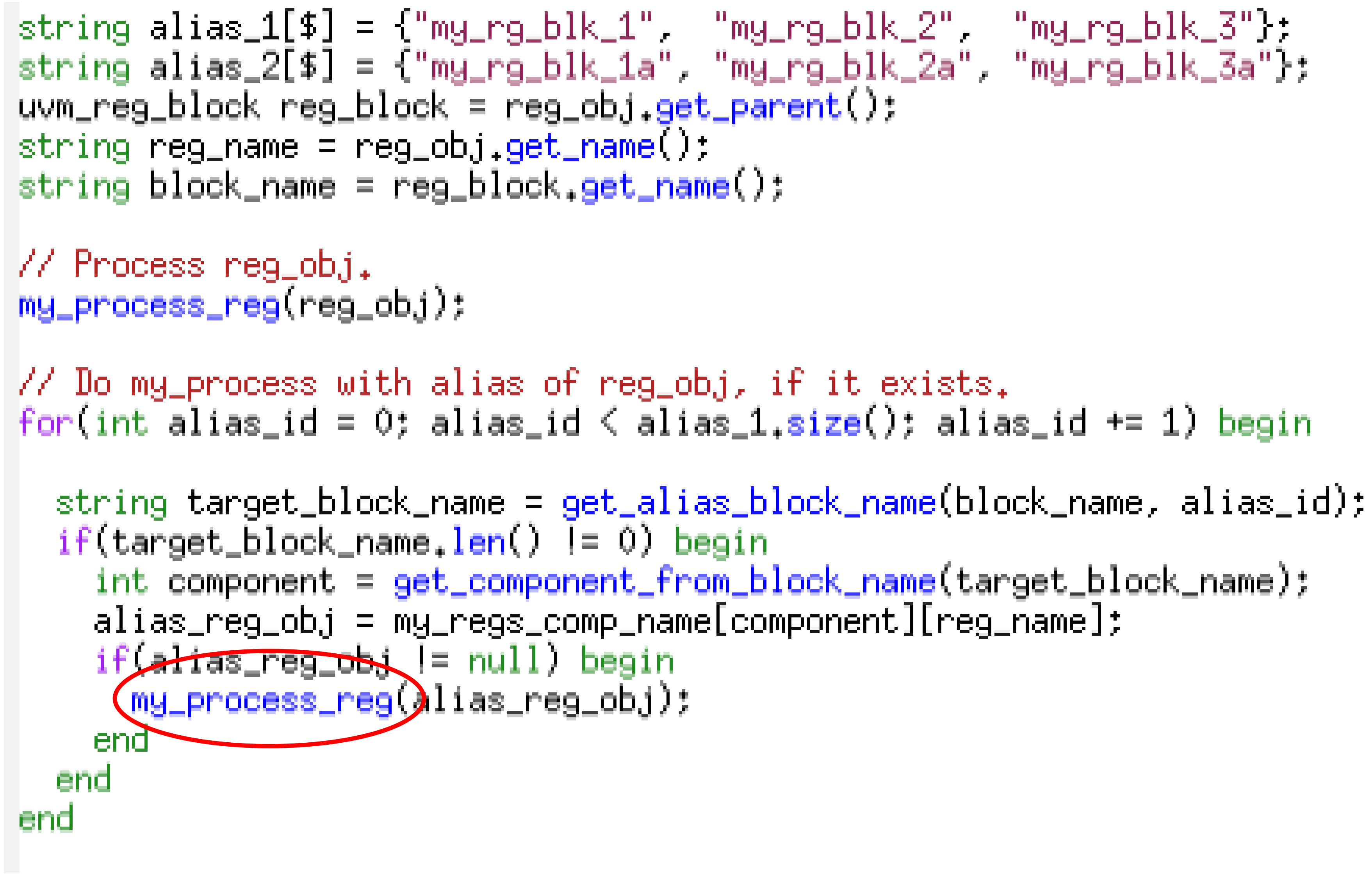
```
string alias_1[$] = {"my_rg_blk_1",  "my_rg_blk_2",  "my_rg_blk_3"};
string alias_2[$] = {"my_rg_blk_1a", "my_rg_blk_2a", "my_rg_blk_3a"};
uvm_reg_block reg_block = reg_obj.get_parent();
string reg_name = reg_obj.get_name();
string block_name = reg_block.get_name();

// Process reg_obj.
my_process_reg(reg_obj);

// Do my_process with alias of reg_obj, if it exists.
for(int alias_id = 0; alias_id < alias_1.size(); alias_id += 1) begin

  string target_block_name = get_alias_block_name(block_name, alias_id);
  if(target_block_name.len() != 0) begin
    int component = get_component_from_block_name(target_block_name);
    alias_reg_obj = my_regs_comp_name[component][reg_name];
    if(alias_reg_obj != null) begin
      my_process_reg(alias_reg_obj);
    end
  end
end
```

Register Behavior: Block Aliasing



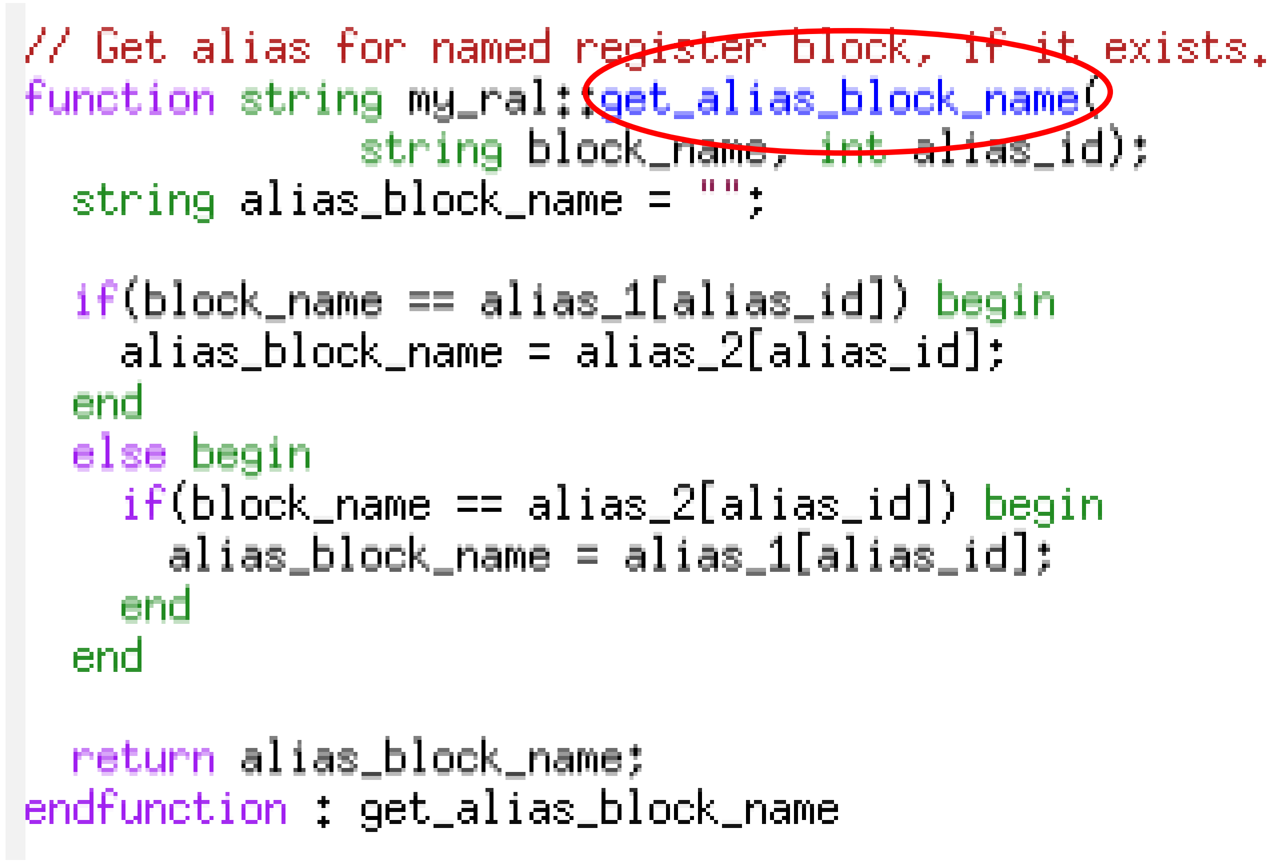
```
string alias_1[$] = {"my_rg_blk_1",  "my_rg_blk_2",  "my_rg_blk_3"};
string alias_2[$] = {"my_rg_blk_1a", "my_rg_blk_2a", "my_rg_blk_3a"};
uvm_reg_block reg_block = reg_obj.get_parent();
string reg_name = reg_obj.get_name();
string block_name = reg_block.get_name();

// Process reg_obj.
my_process_reg(reg_obj);

// Do my_process with alias of reg_obj, if it exists.
for(int alias_id = 0; alias_id < alias_1.size(); alias_id += 1) begin

  string target_block_name = get_alias_block_name(block_name, alias_id);
  if(target_block_name.len() != 0) begin
    int component = get_component_from_block_name(target_block_name);
    alias_reg_obj = my_regs_comp_name[component][reg_name];
    if(alias_reg_obj != null) begin
      my_process_reg(alias_reg_obj);
    end
  end
end
```

Register Behavior: Block Aliasing



```
string alias_1[$] = {"my_rg_blk_1",  "my_rg_blk_2",  "my_rg_blk_3"};
string alias_2[$] = {"my_rg_blk_1a", "my_rg_blk_2a", "my_rg_blk_3a"};
uvm_reg_block reg_block = reg_obj.get_parent();
string reg_name = reg_obj.get_name();
string block_name = reg_block.get_name();

// Process reg_obj.
my_process_reg(reg_obj);

// Do my_process with alias of reg_obj, if it exists.
for(int alias_id = 0; alias_id < alias_1.size(); alias_id += 1) begin

  string target_block_name = get_alias_block_name(block_name, alias_id);
  if(target_block_name.len() != 0) begin
    int component = get_component_from_block_name(target_block_name);
    alias_reg_obj = my_regs_comp_name[component][reg_name];
    if(alias_reg_obj != null) begin
      my_process_reg(alias_reg_obj);
    end
  end
end
```

Register Behavior: Block Aliasing



```
// Get alias for named register block, if it exists.
function string my_ral::get_alias_block_name(
                  string block_name, int alias_id);
  string alias_block_name = "";

  if(block_name == alias_1[alias_id]) begin
    alias_block_name = alias_2[alias_id];
  end
  else begin
    if(block_name == alias_2[alias_id]) begin
      alias_block_name = alias_1[alias_id];
    end
  end

  return alias_block_name;
endfunction : get_alias_block_name
```

Register Test

Register Test

Automated Register Test

Mirror Check Control

Register Test: Automated Register Test

A Simple Register Sequence

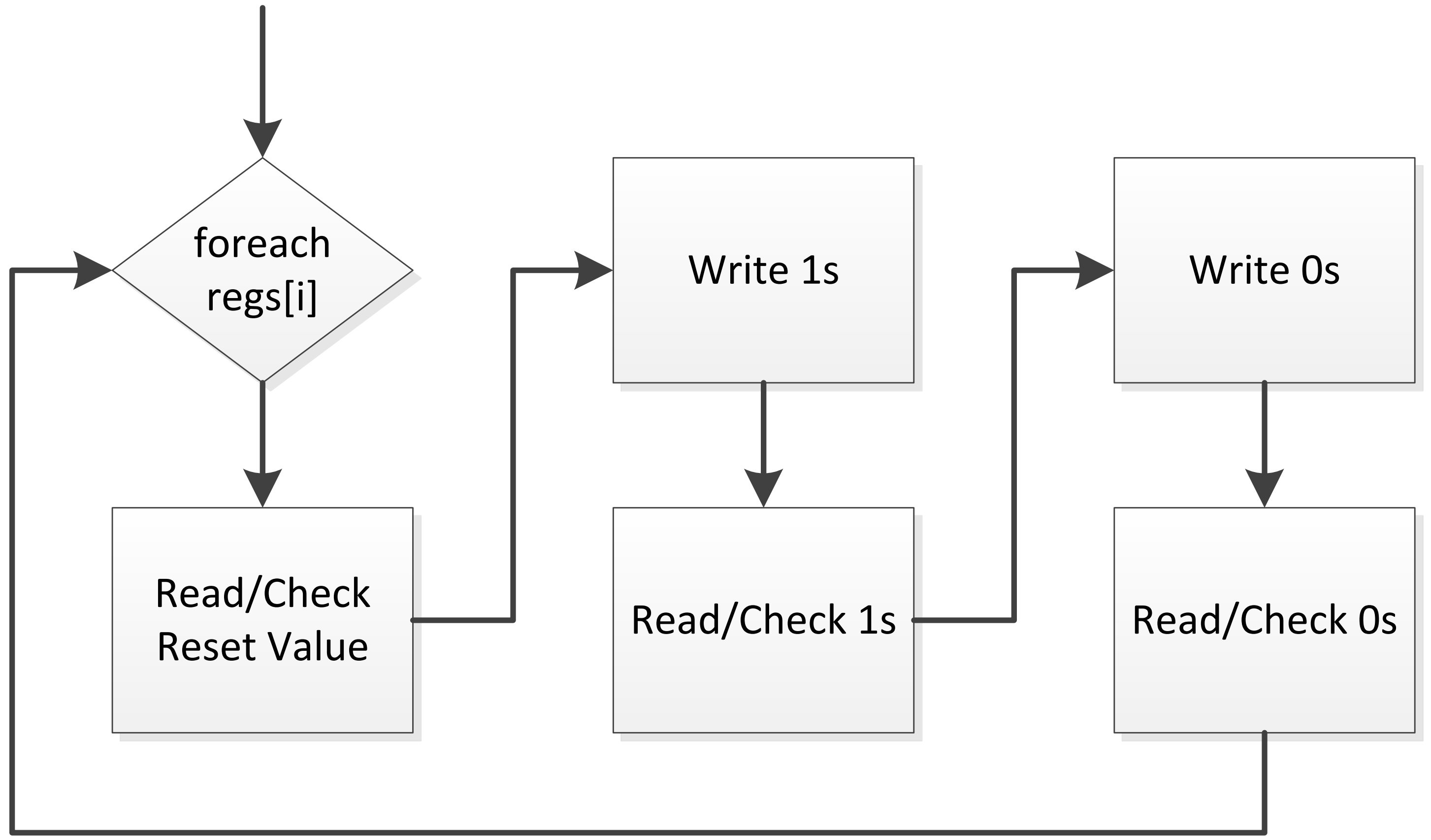
- A register list is built with something like:

```
reg_model.get_registers (regs, 1);
regs.sort() with (item.get_full_name());
```

- A sort is included to help ensure reproducibility.

Register Test: Automated Register Test

A Simple Register Sequence



Register Test: Automated Register Test

Single Register Add

- Unpredictable Field
- Simple Corner Case
- Complex Corner Case
- Any Write Attempt
- Any Read Attempt
- Register Sequence with Modifications

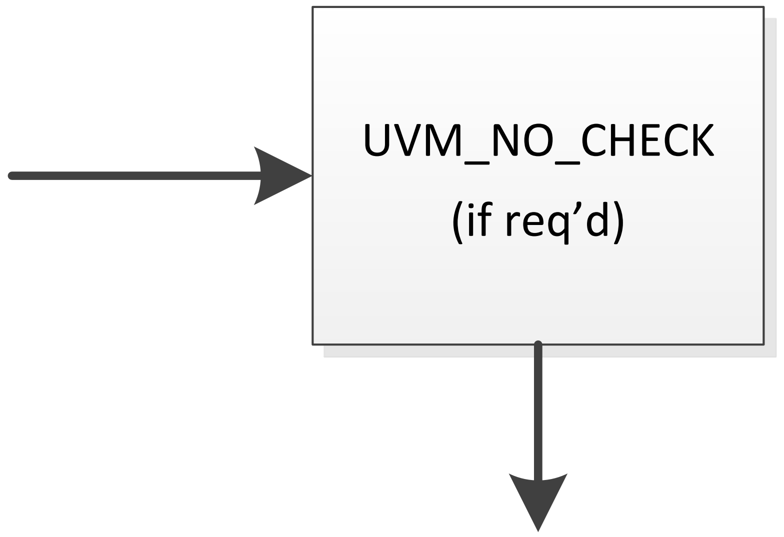
Register Test: Automated Register Test

Unpredictable Field

For example, a field value is impossible to predict.

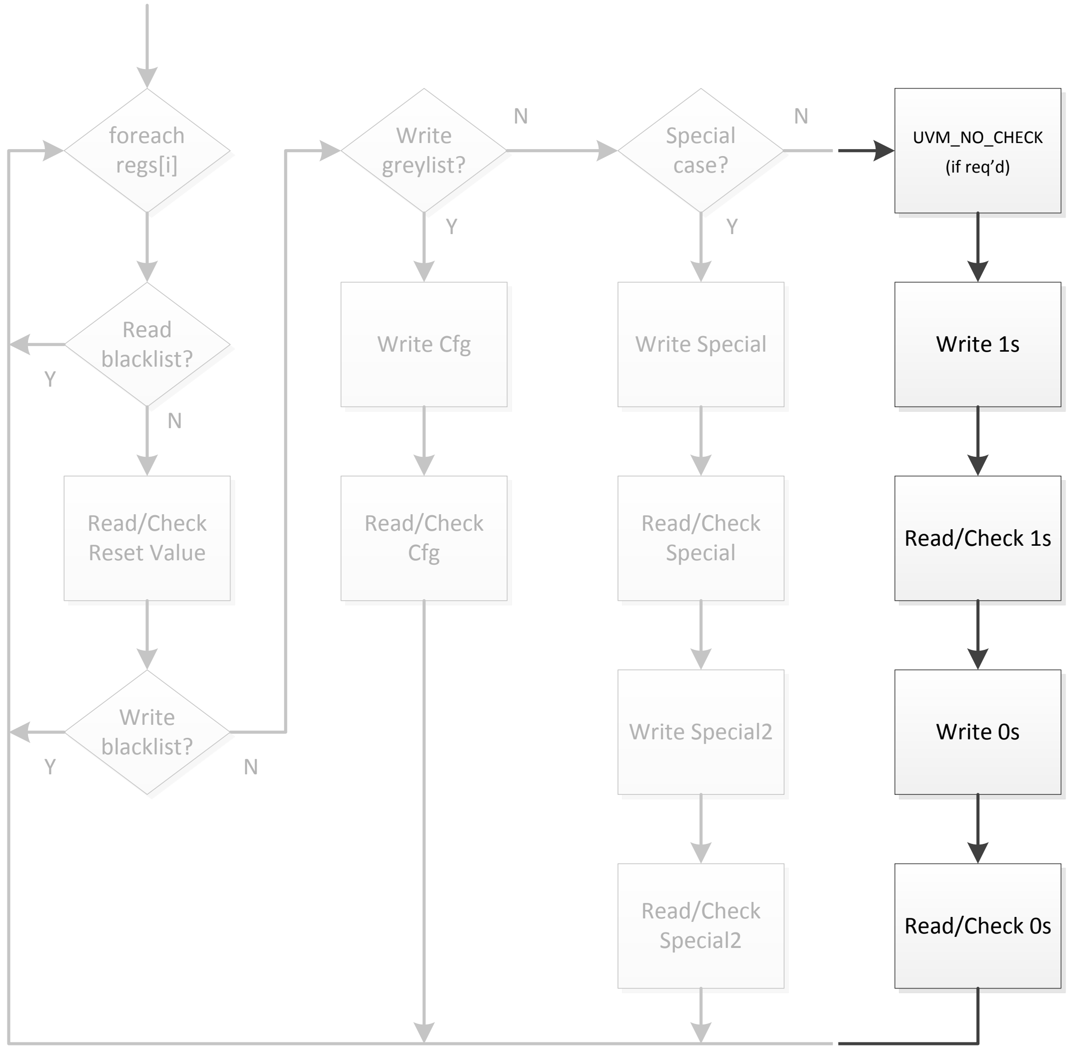
Field set_compare with UVM_NO_CHECK

(low coverage impact):



Register Test: Automated Register Test

Unpredictable Field

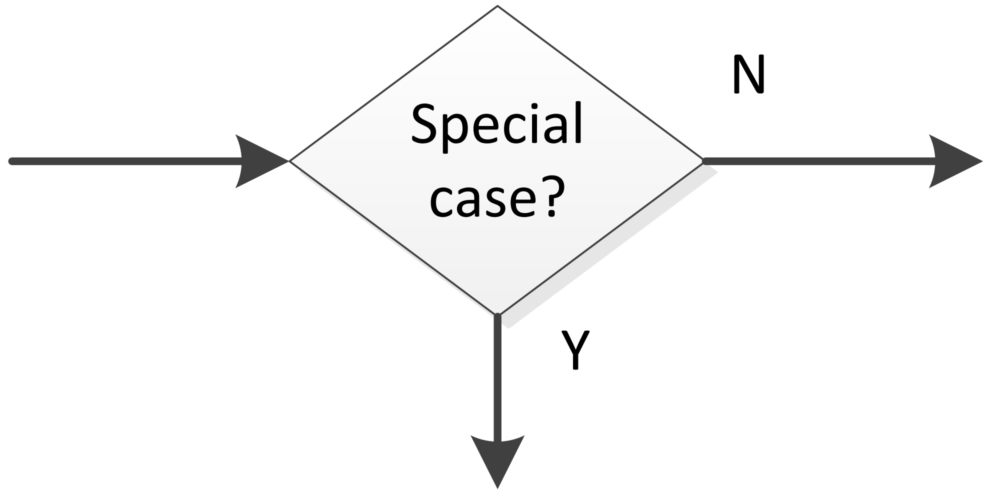


Register Test: Automated Register Test

Simple Corner Case

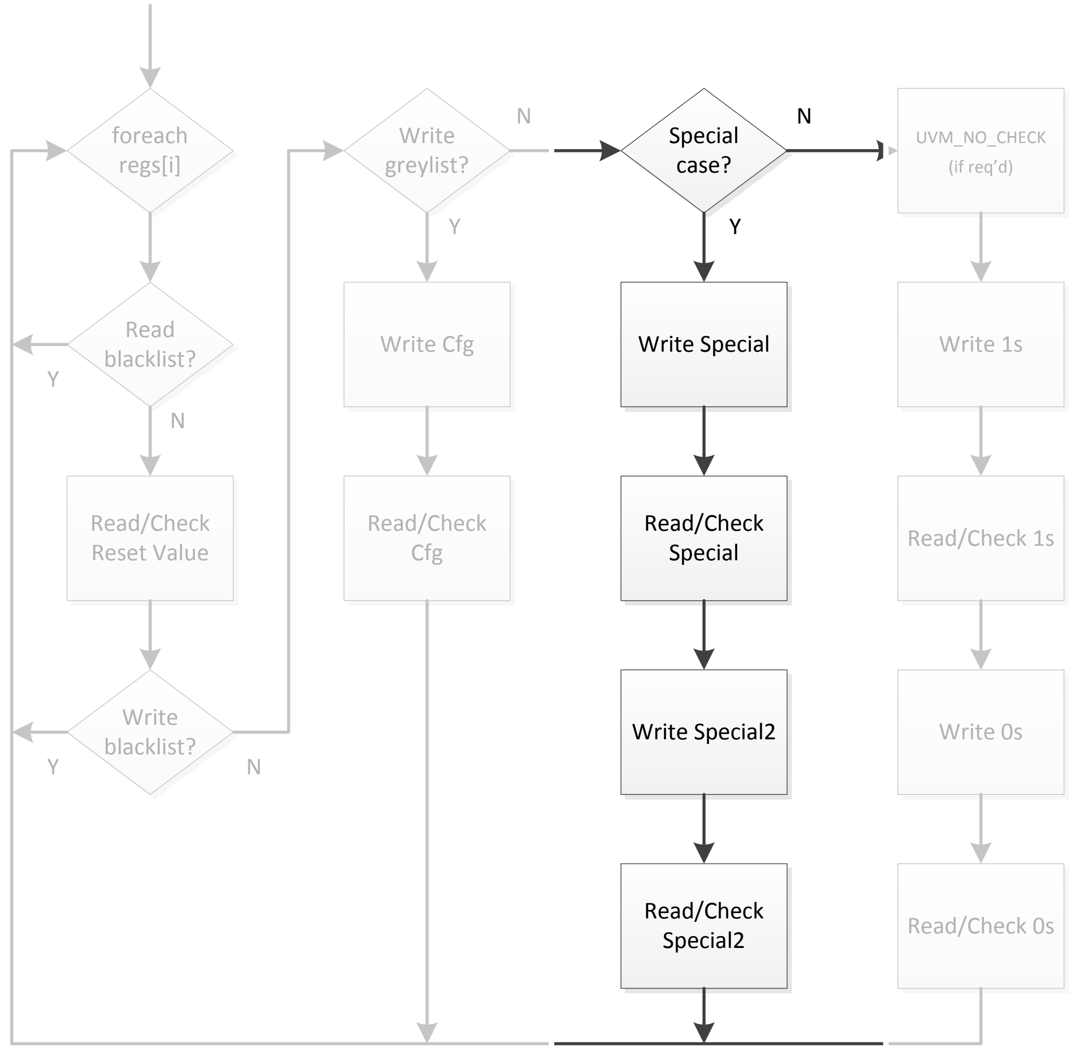
For example, enable bit should not be randomly set.

Special case:



Register Test: Automated Register Test

Simple Corner Case

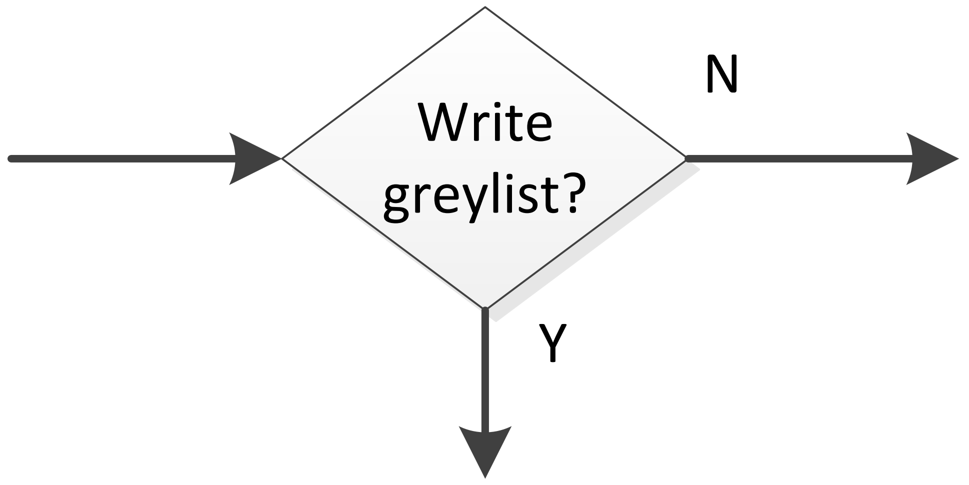


Register Test: Automated Register Test

Complex Corner Case

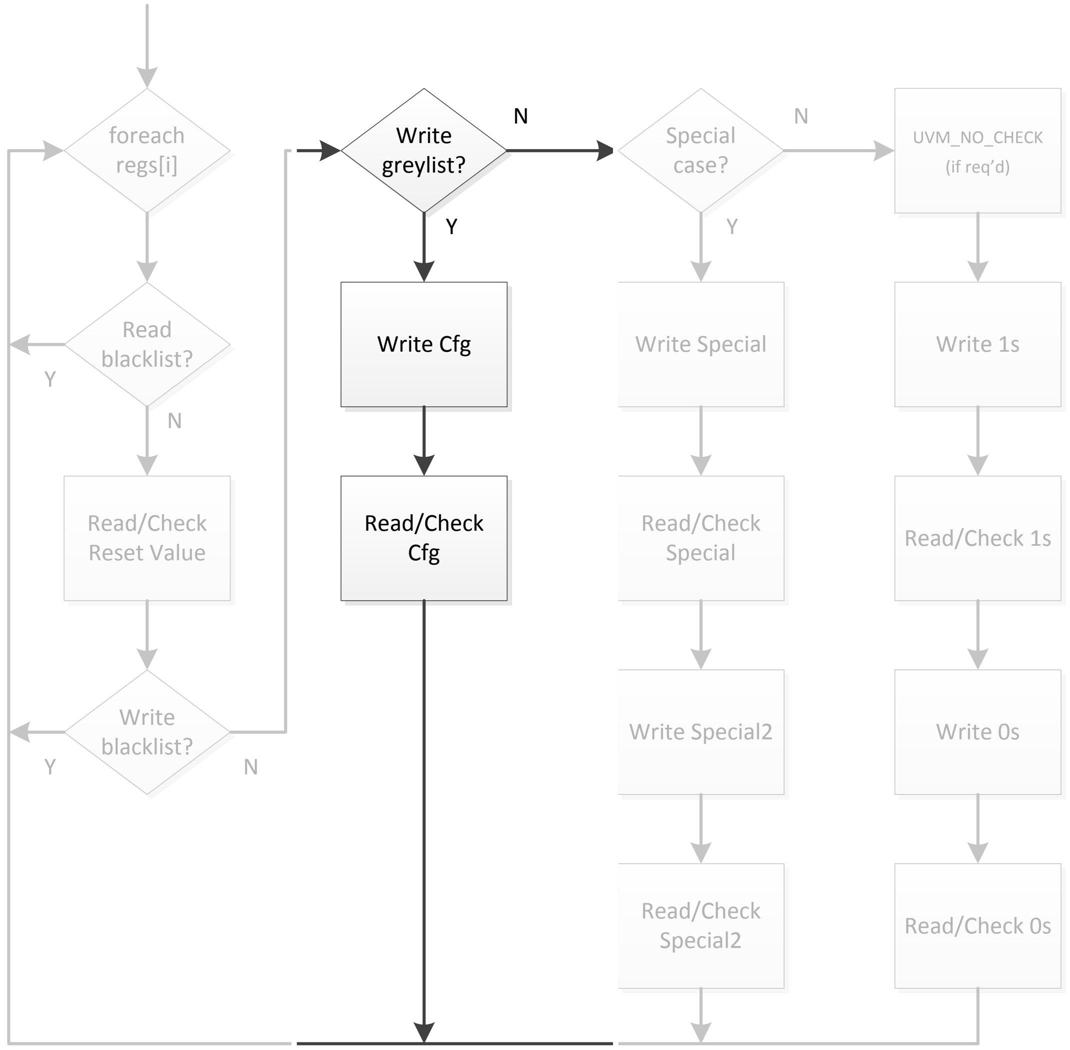
For example, many bit combinations result in errors.

Add to write greylist, write calculated config value:



Register Test: Automated Register Test

Complex Corner Case

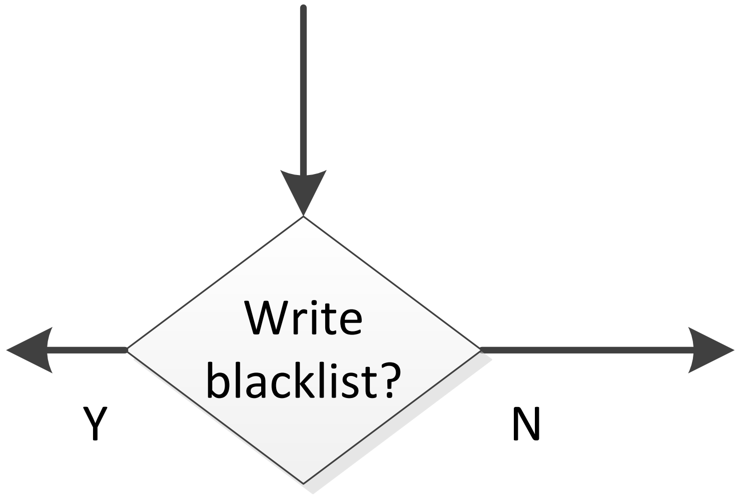


Register Test: Automated Register Test

Any Write Attempt

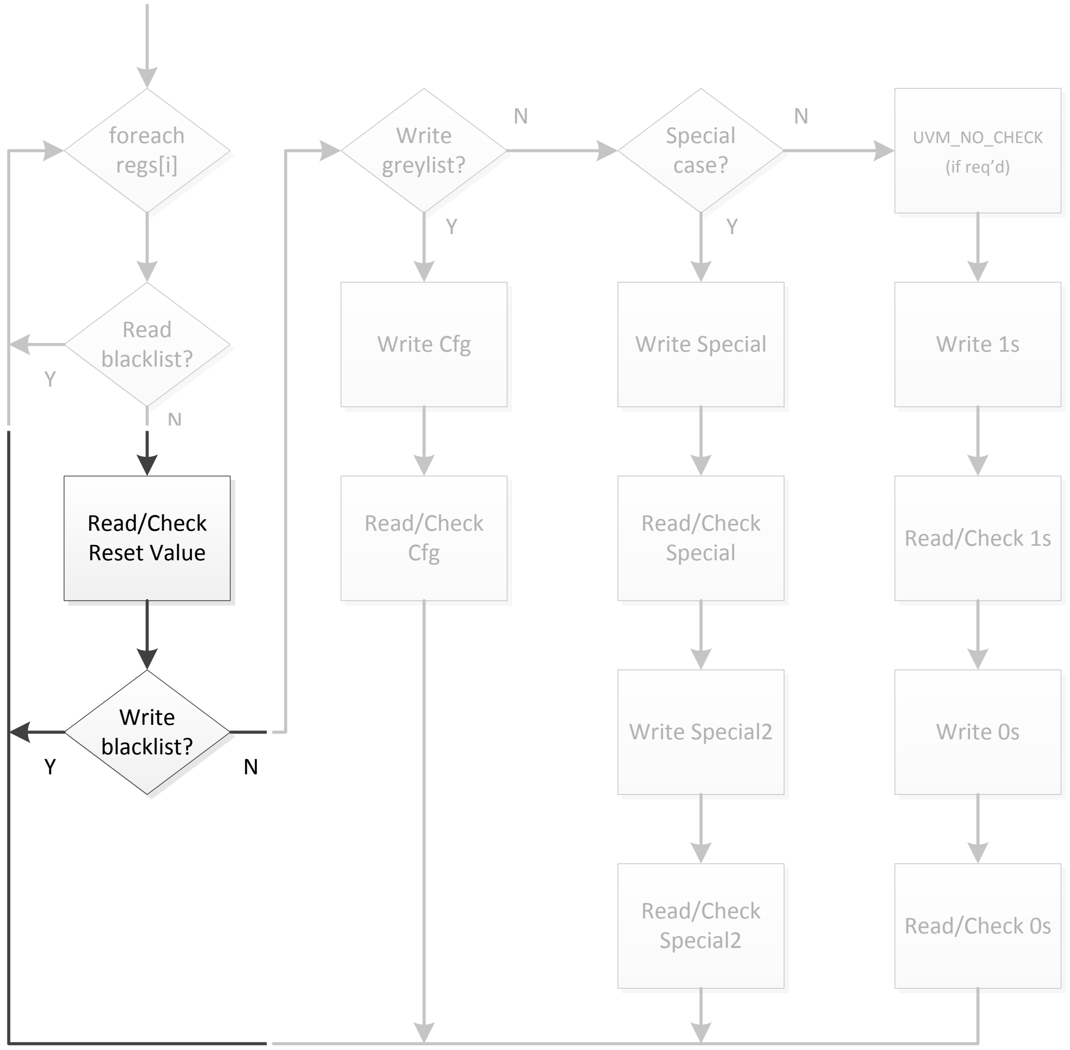
For example, write results in error regardless of value.

Add to write blacklist, no writes:



Register Test: Automated Register Test

Any Write Attempt

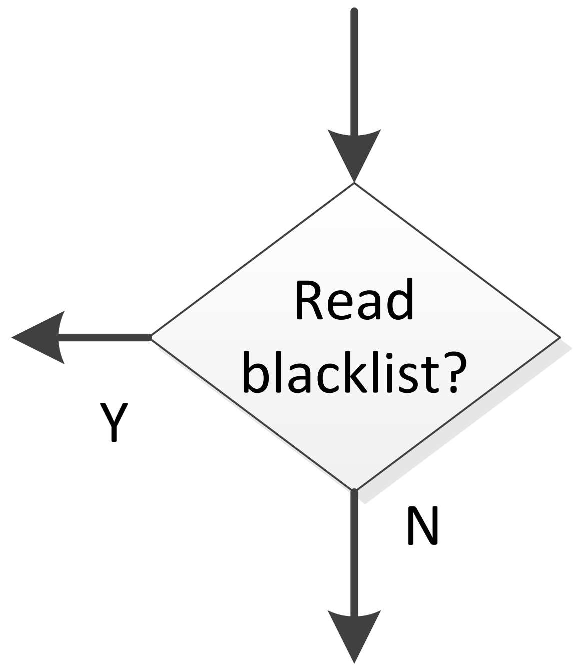


Register Test: Automated Register Test

Any Read Attempt

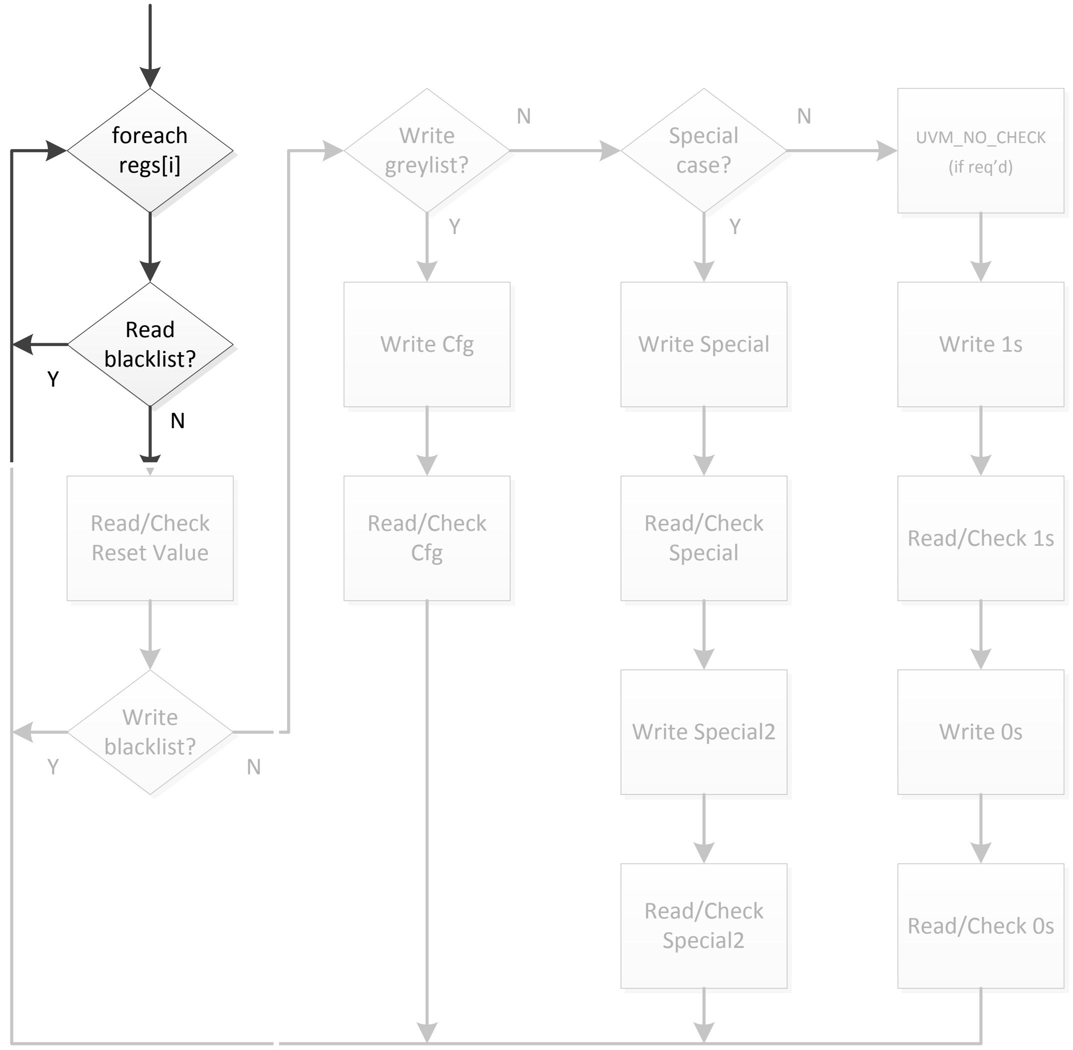
For example, any read results in error regardless of register value.

Read blacklist (highest coverage impact):



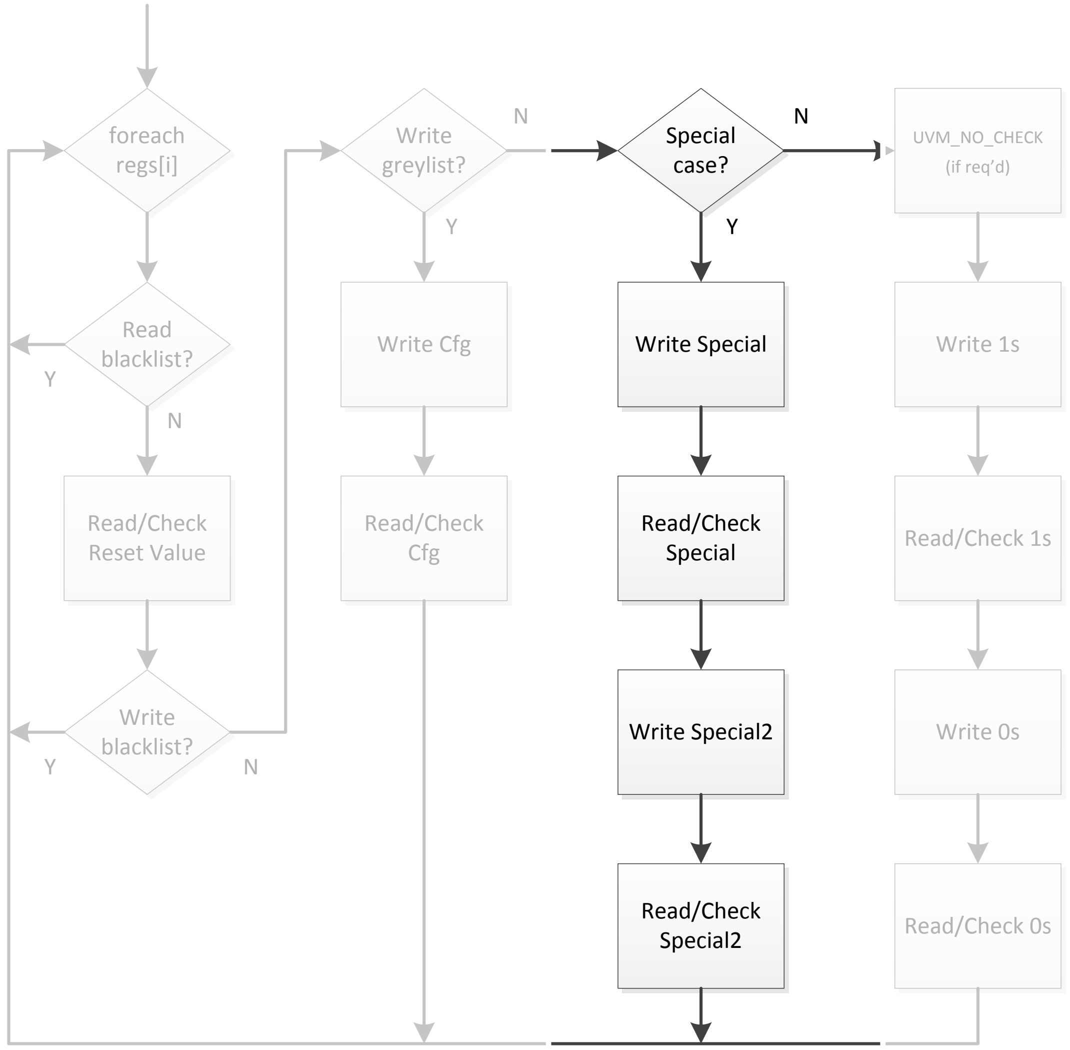
Register Test: Automated Register Test

Any Read Attempt



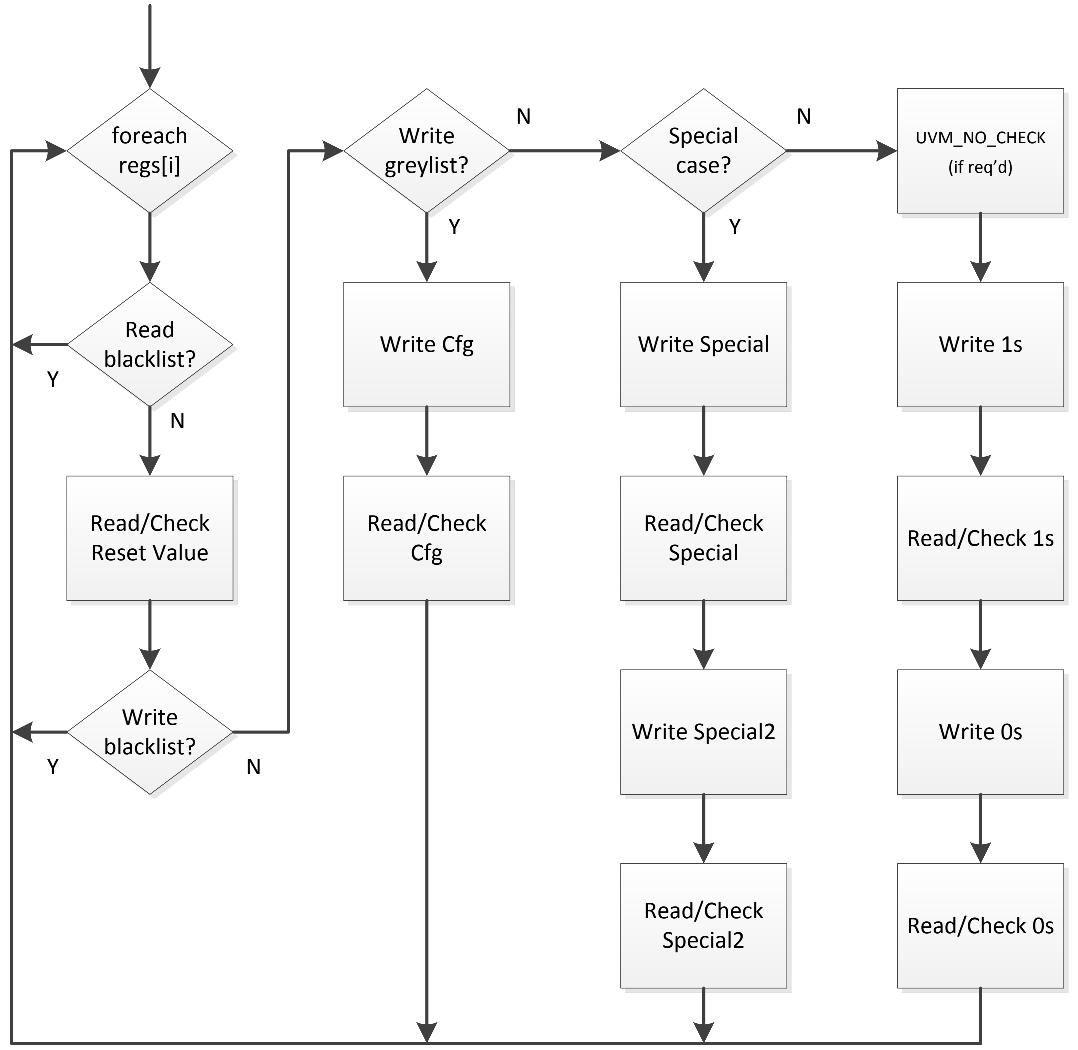
Register Test: Automated Register Test

Any Read Attempt (Special Blocked Register Case)



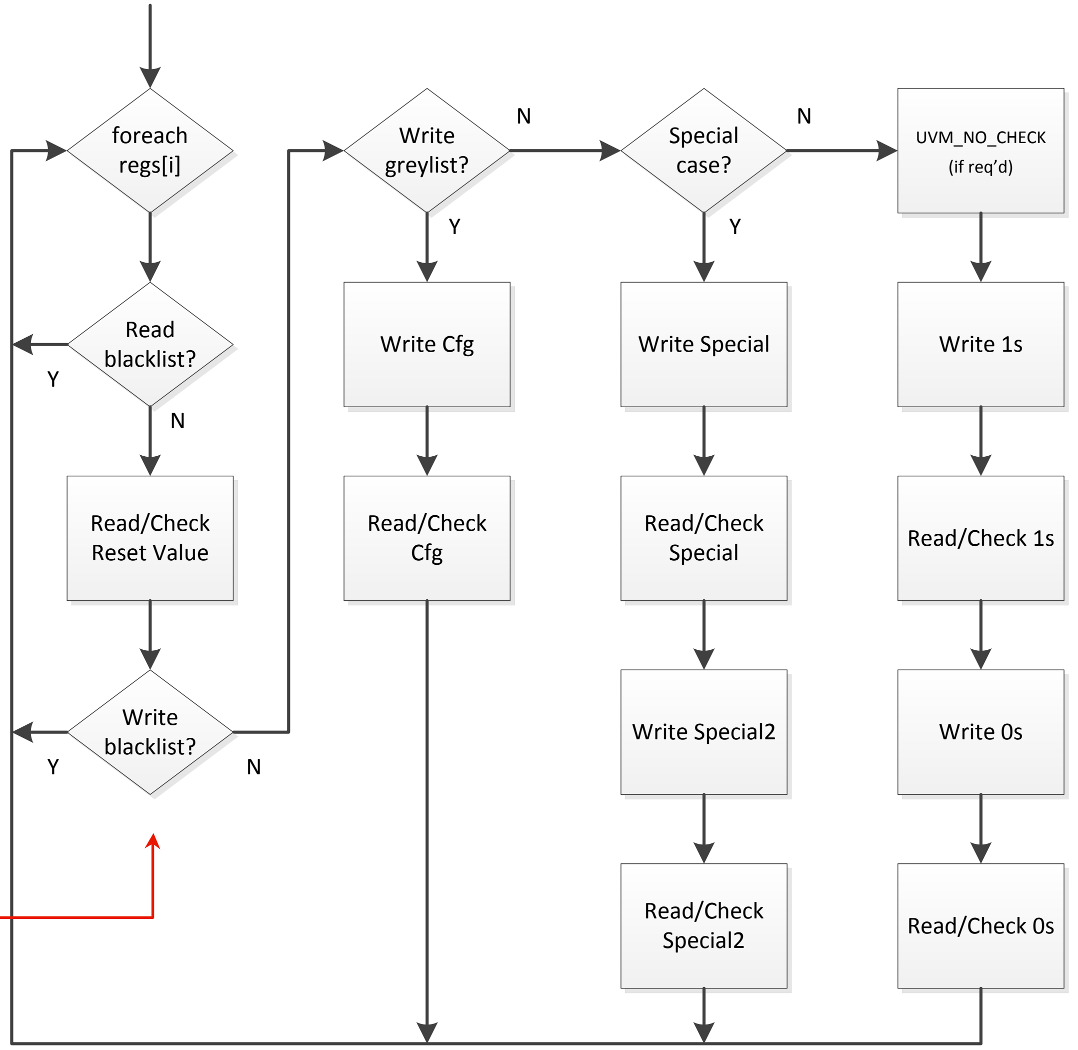
Register Test: Automated Register Test

Register Sequence with Modifications



Register Test: Automated Register Test

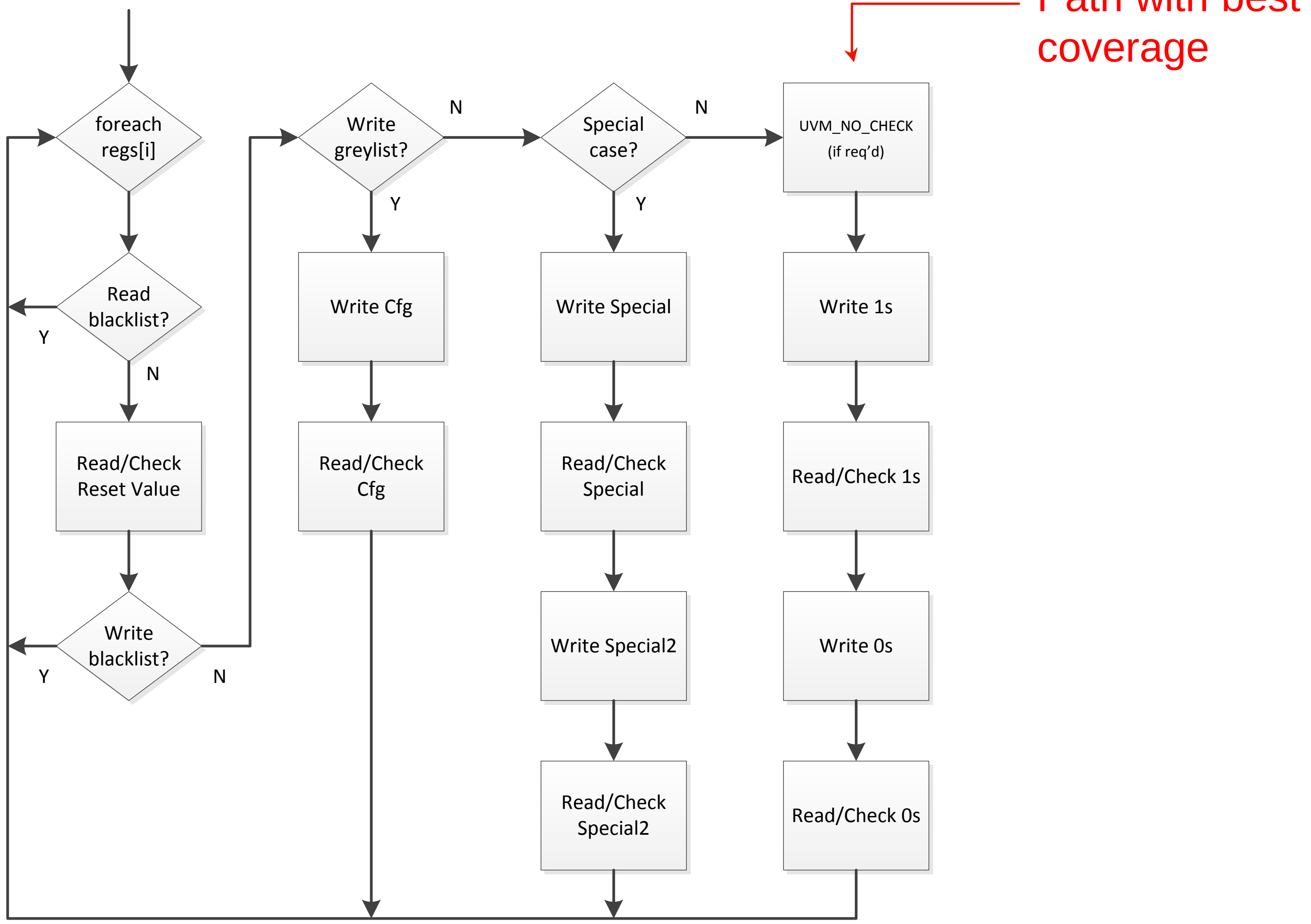
Register Sequence with Modifications



Paths with worst coverage

Register Test: Automated Register Test

Register Sequence with Modifications



Register Test: Mirror Check Control

- Command Line: no_mirror_check
- Function: set_reg_check

Register Test: Mirror Check Control

Command Line: no_mirror_check

- Simple parameter passed from command line to infrastructure
- Simple function called by predictors
  - Tests against parameter passed from command line
  - Called before checking against mirrored values
- Drastic approach, blocks all predictor checks

Register Test: Mirror Check Control

Function: set_reg_check

- Function called from a sequence
- Limited to specific register object

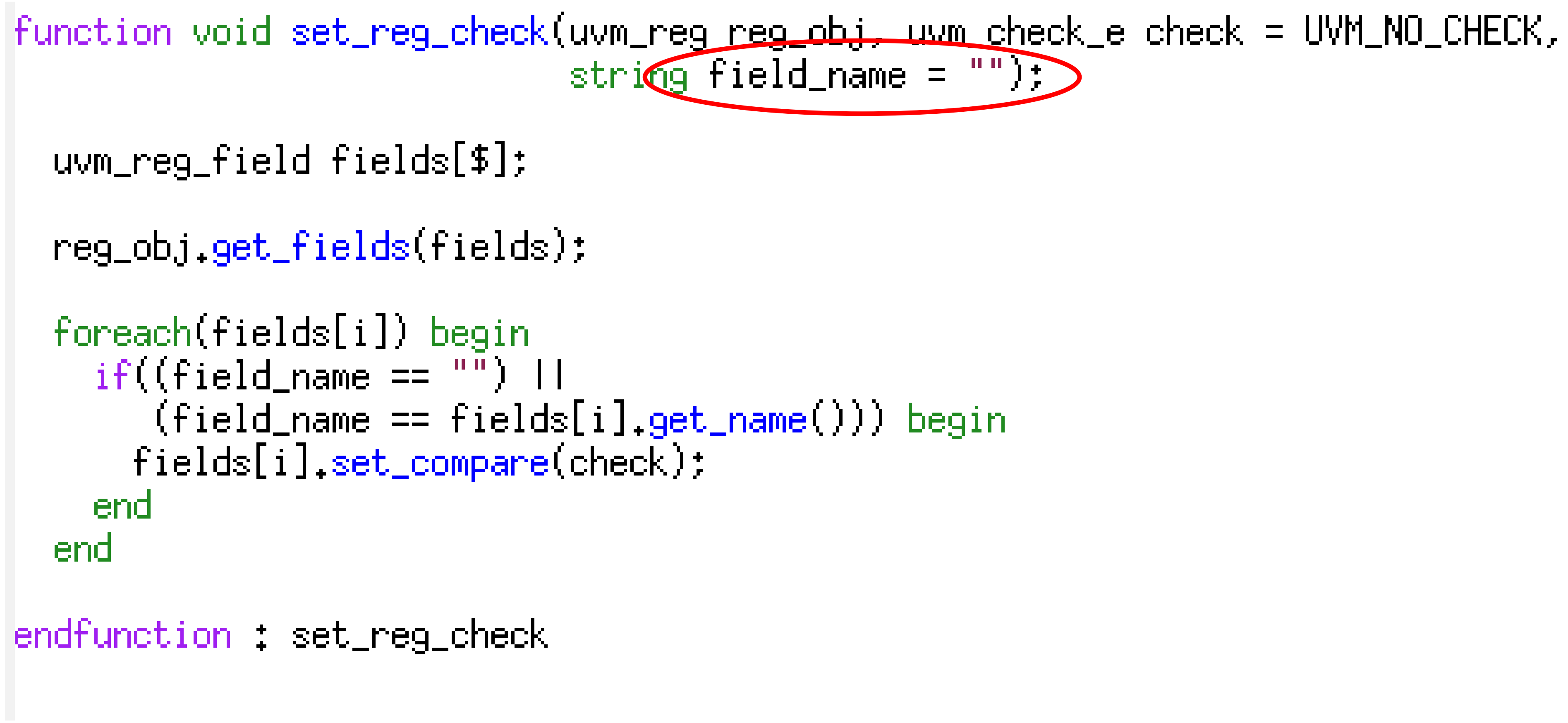
Register Test: Mirror Check Control

Function: set_reg_check

```
function void set_reg_check(uvm_reg reg_obj, uvm_check_e check = UVM_NO_CHECK,
                            string field_name = "");

  uvm_reg_field fields[$];

  reg_obj.get_fields(fields);

  foreach(fields[i]) begin
    if((field_name == "") ||
       (field_name == fields[i].get_name())) begin
      fields[i].set_compare(check);
    end
  end

endfunction : set_reg_check
```

Register Test: Mirror Check Control

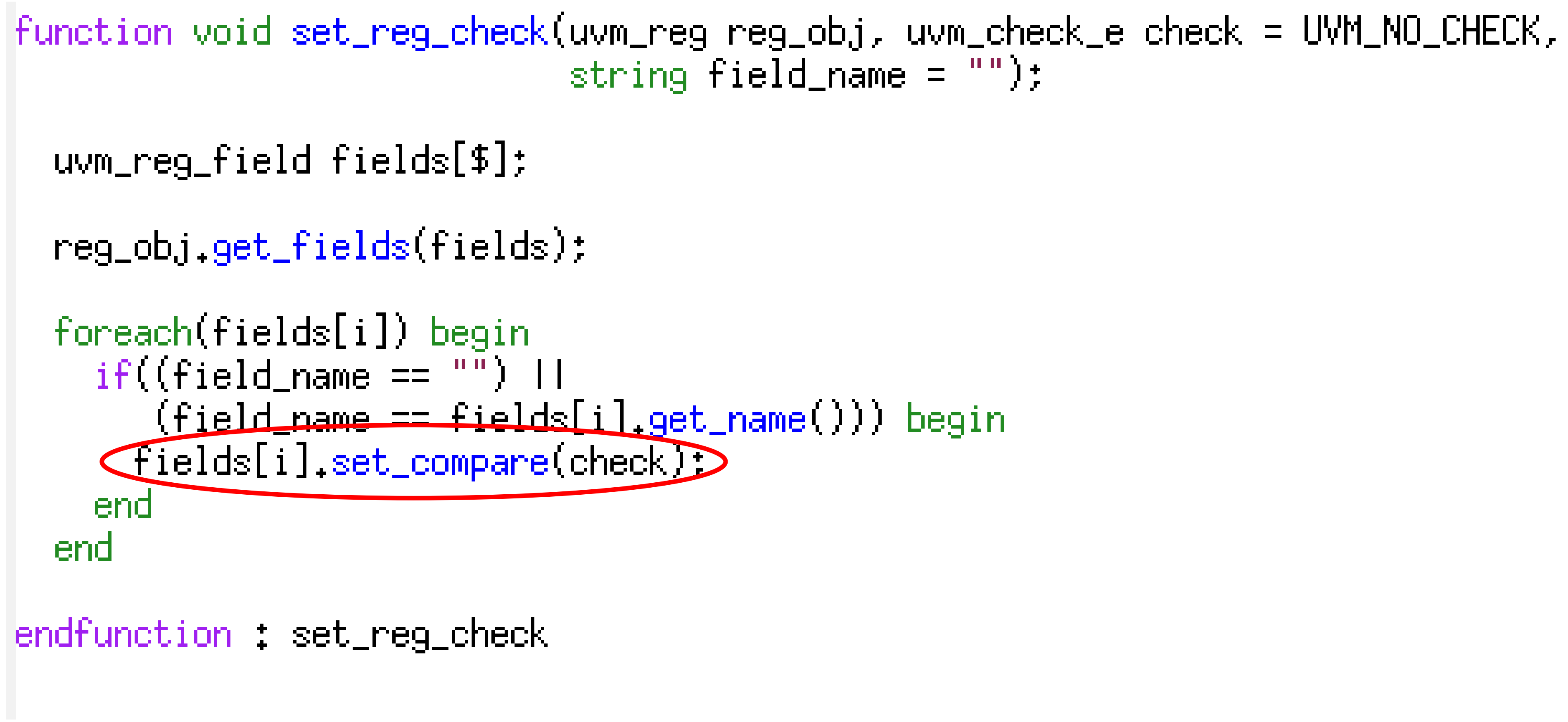
Function: set_reg_check



```
function void set_reg_check(uvm_reg reg_obj, uvm_check_e check = UVM_NO_CHECK,
                            string field_name = "");

  uvm_reg_field fields[$];

  reg_obj.get_fields(fields);

  foreach(fields[i]) begin
    if((field_name == "") ||
       (field_name == fields[i].get_name())) begin
      fields[i].set_compare(check);
    end
  end

endfunction : set_reg_check
```

Register Test: Mirror Check Control

Function: set_reg_check



```
function void set_reg_check(uvm_reg reg_obj, uvm_check_e check = UVM_NO_CHECK,
                            string field_name = "");

  uvm_reg_field fields[$];

  reg_obj.get_fields(fields);

  foreach(fields[i]) begin
    if((field_name == "") ||
       (field_name == fields[i].get_name())) begin
      fields[i].set_compare(check);
    end
  end

endfunction : set_reg_check
```

Conclusions

Conclusions

General Conclusions

Future Enhancements

Conclusions: General Conclusions

- Large register models with side effects and exceptions managed in:
  - Register data structures
  - Register behavior
  - Register test
- Care taken to improve performance and maintainability

Conclusions: Future Enhancements

- Sort on “get” return
- Enhance hash table support
- Replace string names with enumerated types

Conclusions: Future Enhancements

Sort on “get” return

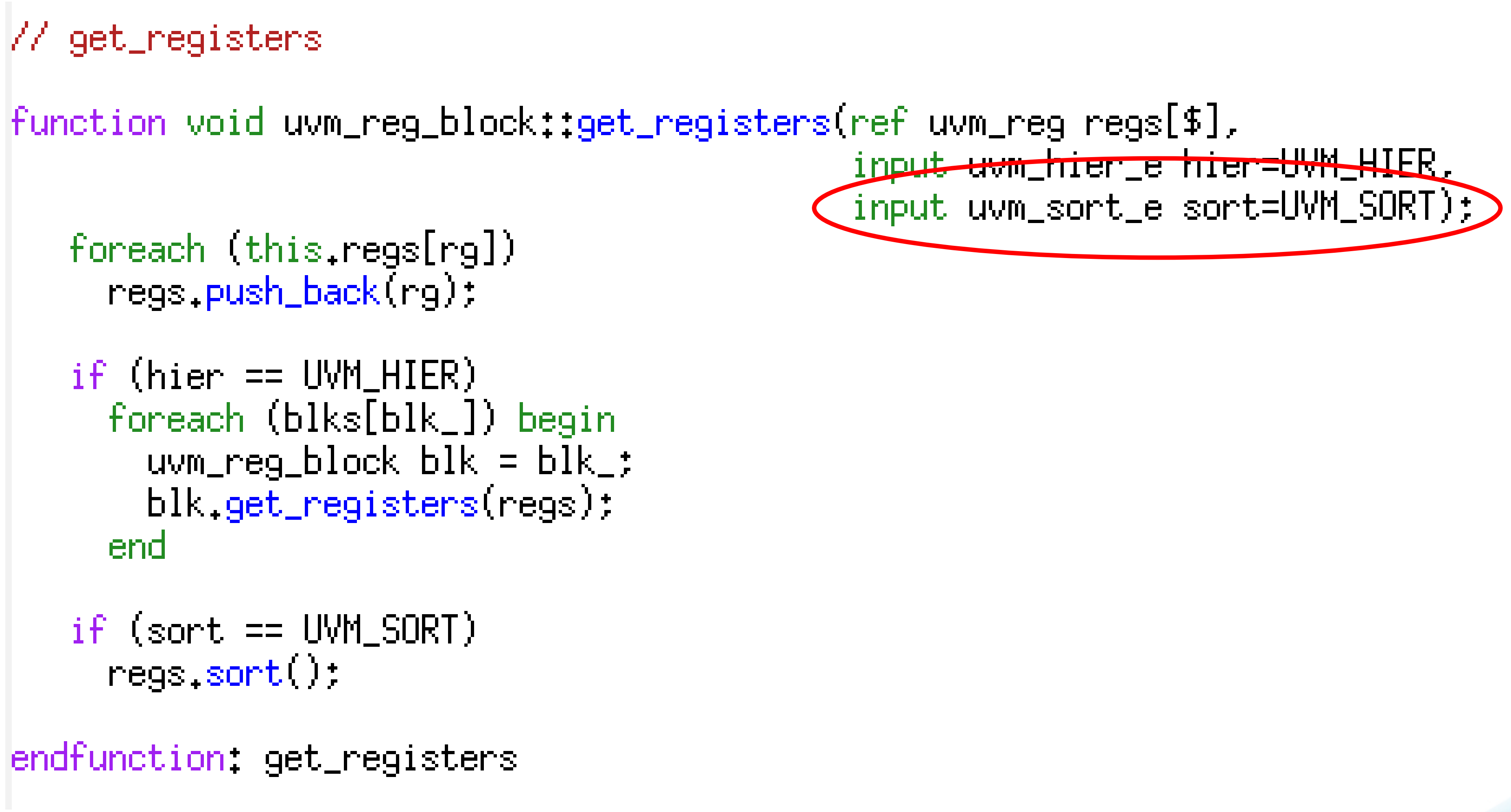
For example:

```
// get_registers

function void uvm_reg_block::get_registers(ref uvm_reg regs[$],
                                           input uvm_hier_e hier=UVM_HIER,
                                           input uvm_sort_e sort=UVM_SORT);

   foreach (this.regs[rg])
     regs.push_back(rg);

   if (hier == UVM_HIER)
     foreach (blks[blk_]) begin
       uvm_reg_block blk = blk_;
       blk.get_registers(regs);
     end

   if (sort == UVM_SORT)
     regs.sort();

endfunction: get_registers
```

Conclusions: Future Enhancements

Sort on “get” return

For example:



```
// get_registers

function void uvm_reg_block::get_registers(ref uvm_reg regs[$],
                                           input uvm_hier_e hier=UVM_HIER,
                                           input uvm_sort_e sort=UVM_SORT);

   foreach (this.regs[rg])
     regs.push_back(rg);

   if (hier == UVM_HIER)
     foreach (blks[blk_]) begin
       uvm_reg_block blk = blk_;
       blk.get_registers(regs);
     end

   if (sort == UVM_SORT)
     regs.sort();

endfunction: get_registers
```

Conclusions: Future Enhancements

Enhance hash table support

Might be nice with unchanged register model between sims …

- Generate hash table?
- Save hash table?
- Restore hash table?

Conclusions: Future Enhancements

Replace string names with enumerated types

Using string names:

```
uvm_reg reg_obj = my_block.get_reg_by_name("My_Register");
```

Using proposed enumerated types (faster searches):

```
uvm_reg reg_obj = my_block.get_reg_by_enum(My_Register);
```

Compatible with enumerated types in hash tables (faster hash testing):

```
local uvm_reg my_regs_enum[interface_e][reg_e][$];
local uvm_reg my_regs_comp_enum[int][reg_e];
```

Synopsys Users Group
Boston

Thank You