2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Verifying RO registers: Challenges and the solution

Ivana Dobrilovic

Veriest Solutions

ivanad@veriests.com

Veriest

accellera SYSTEMS INITIATIVE

Agenda

- Registers – role and types
- RO registers’ verification challenges
- Suggested solution
- Conclusion

Registers – role and types

- HW configuration, control and monitoring(debug).
- Registers classification
  - Function that serves for
  - Access type
- Registers verification
  - Basic register test
  - Checking functional correctness

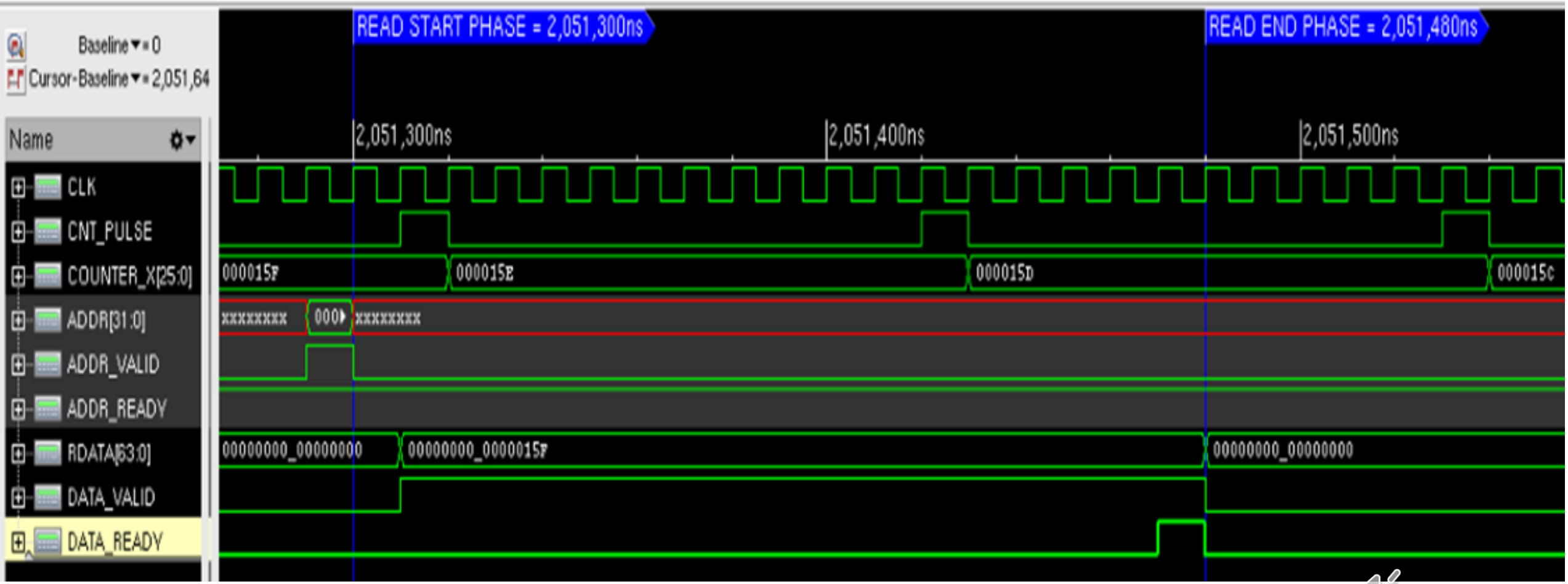
RO Registers and their verification

- Status and debug registers, HW timers and counters
- Interrupt status, error status, link operational status, FSM status, faults
- Check their value occasionally
- Two aspects of the strategy
  - Register polling sequence – when to read register?
  - Checking – where to check its value?
- Complex verification task

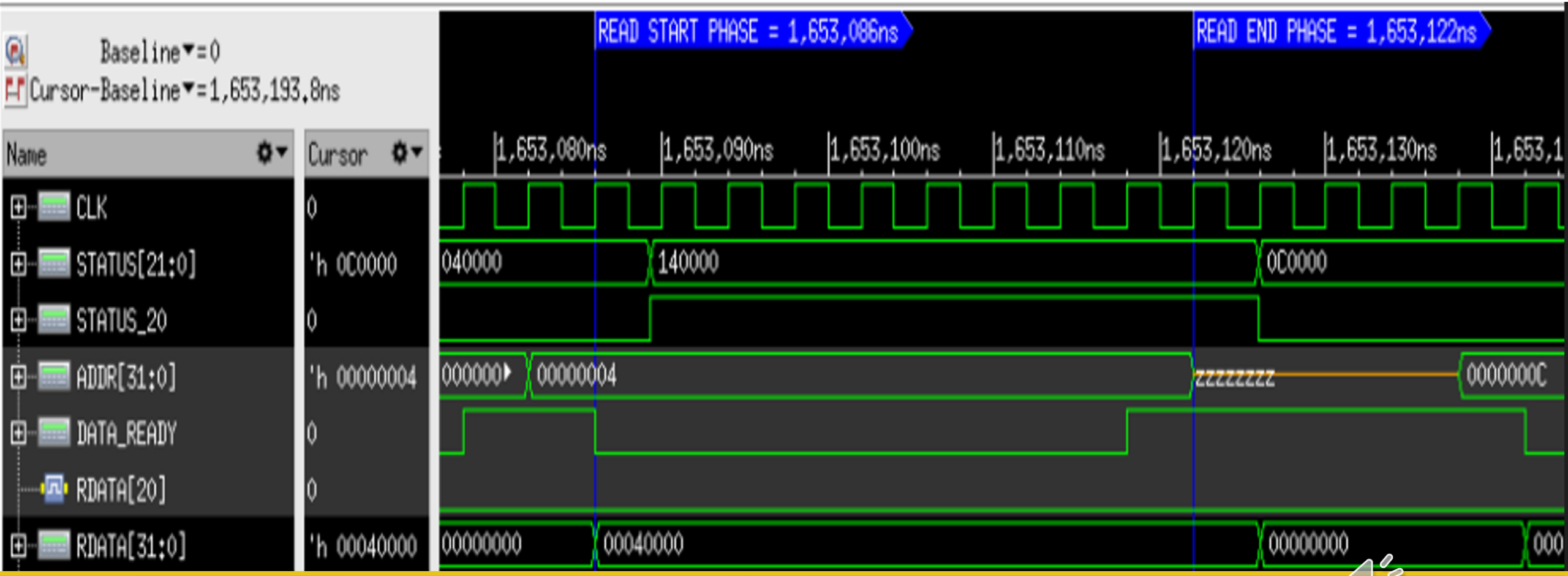
Challenges

- Having expected register value reliably predicted
- Checker relaxation
- Challenges
  - Register value changes too fast
  - Read cycle duration
  - Registers containing fields of various types

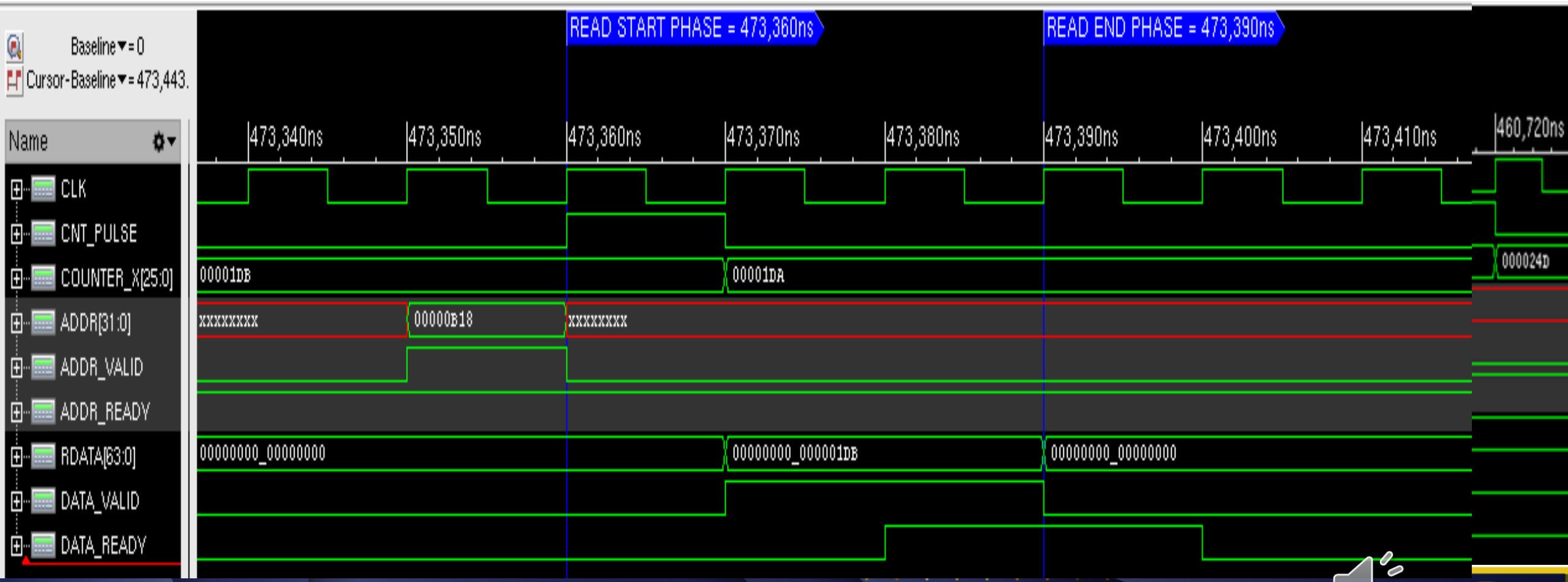
Challenges - examples



Challenges - examples



The most challenging part



Solution

- Having common approach covers several issues
- Expected register/field value approximately predicted
- Pool of all possible values in the given time

Solution - Example

- StatusA - uvm_reg_field
- REG1 - uvm_reg
- BLOK1 – uvm_reg_block
- Register protocol - Protocol X
- Model - common UVM object where checking is done

Solution mechanism - Declaration

- Register block declaration
- 3 variables per register or register field in the reference model
- The idea is to collect all the expected register values during the read transaction targeting the specific register

```
class model extends uvm_component;

    ..................................
    //BLOCK1 declaration
    block1_regs BLOCK1;


    //Status register check fields
    bit                     reg1_read_ongoing;
    bit [FAW-1:0]           reg1_statusA;
    bit [FAW-1:0]           reg1_statusA_q[$];
    ..................................
endclass
```

Solution mechanism - Queueing

- calculate_reg1_statusA()
- Value calculation
- Queueing - pool of the expected values which enables flexibility in checking the register field value

```
function model::calculate_reg1_statusA();
    ..................................
    // valueA is the calculated value
    // and it is assigned to expected reg1_statusA
    reg1_statusA  = valueA;

    // Putting calculated value in a pull of expected values
    // in case that read is ongoing.
    if (reg1_read_ongoing == 1)begin
        reg1_statusA_q.push_back(reg1_statusA);
    end
    ..................................
endfunction
```

Solution mechanism – Register read start

- write_X()
- Indicating that REG1 read starts
- Queueing current value predicted for statusA field

```
function model::write_X(X_seq_item tr);
        if(tr.phase == START && tr.direction == READ ) begin
        //Read operation started
        //Start collecting expected possible values for read data
        if (tr.address  == BLOCK1.REG1.get_address() ) begin
            reg1_read_ongoing = 1;
            reg1_statusA_q.push_back(reg1_statusA);
        end
endclass
```

Solution mechanism – Register read end

- write_X - end phase
- Data checking
- Ongoing flag reset
- Deleting a queue

```
function model::write_X(X_seq_item tr);
    if(tr.phase == END && tr.direction == READ ) begin
        //Read operation ended
        //Compare the read data against the queue elements
        if (tr.address  == BLOCK1.REG1.get_address() ) begin
            //check REG1 field
            if(tr.rdata[FAW-1:0] inside {reg1_statusA_q})begin
                `uvm_info("reg1_statusA_check",
                          $sformatf("REG1 read:
                                     Observed rdata has correct value"),UVM_LOW);
            end else begin
                `uvm_info("reg1_statusA_check",
                          $sformatf("REG1 read - Observed mismatch:
                                    read data %0x, expected values %p",
                                    tr.rdata, reg1_statusA_q));
            end
            reg1_read_ongoing = 0;
            reg1_statusA_q.delete();
        end
    end
endfunction
```

Solution mechanism – multiple fields checking

- Same approach for more different fields of the same register
- Common register ongoing flag
- Fields and field queues for each field separately
- Start queueing predicted values for each fields

```
//Status register check fields
bit                     reg1_read_ongoing;
bit [FAW-1:0]           reg1_statusA;
bit [FAW-1:0]           reg1_statusA_q[$];
bit [FBW-1:0]           reg1_statusB;
bit [FBW-1:0]           reg1_statusB_q[$];
bit                     reg1_statusC;
bit                     reg1_statusC_q[$];
```

```
if(tr.phase == START && tr.direction == READ ) begin
    //Read just started
    //Start collecting expected possible values of read data
    if (tr.address  == BLOCK1.REG1.get_address() ) begin
        reg1_read_ongoing = 1;
        reg1_statusA_q.push_back(reg1_statusA);
        reg1_statusB_q.push_back(reg1_statusB);
        reg1_statusC_q.push_back(reg1_statusC);
    end
end
```

Solution mechanism – multiple fields checking

- Read data slices will be check against expected queues

```
if(tr.phase == END && tr.direction == READ ) begin
    //Read just started
    //Start collecting expected possible values for read data
    if (tr.address   == BLOCK1.REG1.get_address() ) begin
        //check REG1 field
        if(tr.rdata[A_MSB:A_LSB] inside {reg1_statusA_q})begin
            `uvm_info("reg1_statusA_check",
                      $sformatf("REG1 read:
                                 Observed rdata has correct value"),UVM_LOW);
        end else begin
            `uvm_info("reg1_statusA_check",
                      $sformatf("REG1 read - Observed mismatch:
                                 read data %0x, expected values %p",
                                 tr.rdata[A_MSB:A_LSB], reg1_statusA_q));
        end
        if(tr.rdata[B_MSB:B_LSB] inside {reg1_statusB_q})begin
            `uvm_info("reg1_statusB_check",
                      $sformatf("REG1 read:
                                 Observed rdata has correct value"),UVM_LOW);
        end else begin
            `uvm_info("reg1_statusB_check",
                      $sformatf("REG1 read - Observed mismatch:
                                 read data %0x, expected values %p",
                                 tr.rdata[B_MSB:B_LSB], reg1_statusB_q));
        end
        if(tr.rdata[C_POS] inside {reg1_statusC_q})begin
            `uvm_info("reg1_statusC_check",
                      $sformatf("REG1 read:
                                 Observed rdata has correct value"),UVM_LOW);
        end else begin
            `uvm_info("reg1_statusC_check",
                      $sformatf("REG1 read - Observed mismatch:
                                 read data %0x, expected values %p",
                                 tr.rdata[C_POS], reg1_statusC_q));
        end

        reg1_read_ongoing = 0;
        reg1_statusA_q.delete();
        reg1_statusB_q.delete();
        reg1_statusC_q.delete();
    end
end
```

Solution mechanism – Coverage

- coverage point to prove that exact register/field value was checked
- cover the size of the reg1_statusA_q distinguishing queue size of 1

Conclusion – why this solution is good

- Common solution for one of the most challenging tasks in chip development process
- Starts with flexible checker instead of exact one
- More readable and easy to maintain code
- Idea that could be easily adopted to other issues

Q&A?

2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Thank you for your attention!

accellera
SYSTEMS INITIATIVE