2022
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Automate Interrupt Checking with UVM Macros and Python

Aleksandra Dimanic & Nemanja Stevanovic, Vtool Ltd

Yoav Furman & Itay Henigsberg, Chain Reaction Ltd

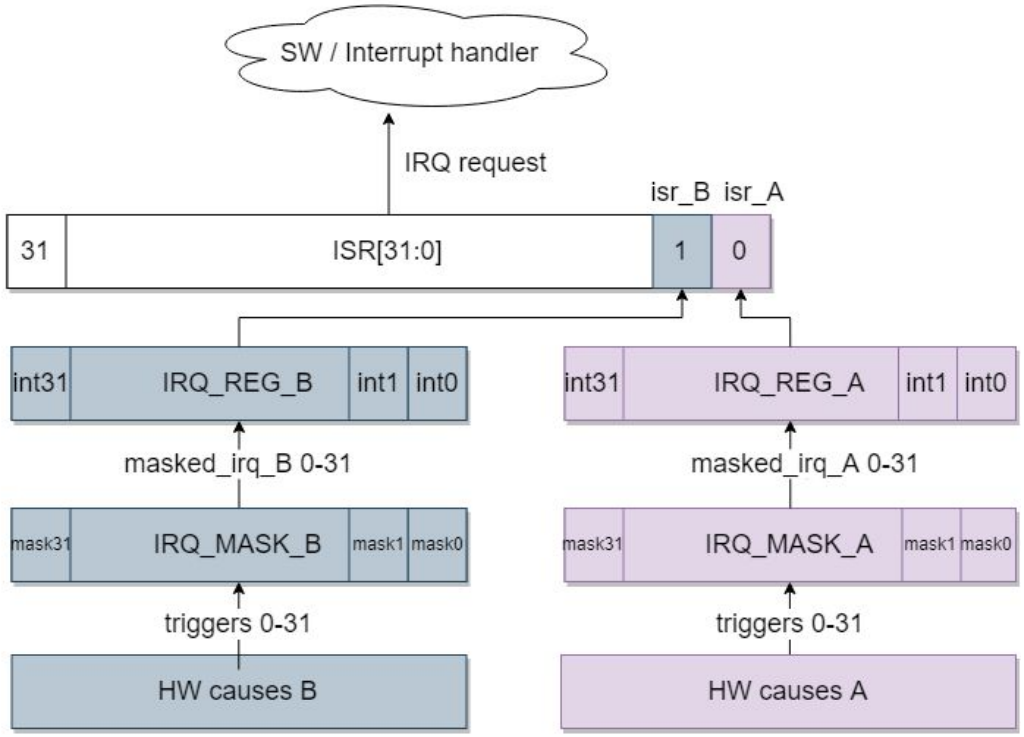
accellera
SYSTEMS INITIATIVE

Interrupt overview

- Essential part of every design
- Common ground for IRQ handling:
  - Usually grouped in blocks in design and share common paths and behavior
  - IRQ trigger is coming from design
  - Reflected in status registers with masking/unmasking/set/reset mechanism
- By taking those common parts the checking can be accelerated and partially automated

Explaining the concept

- Each IRQ has its corresponding register or a field in the register
- Groups or IRQs are mapped in ISR (Interrupt Status Register)
- Registers always have to reflect the correct values of the IRQs



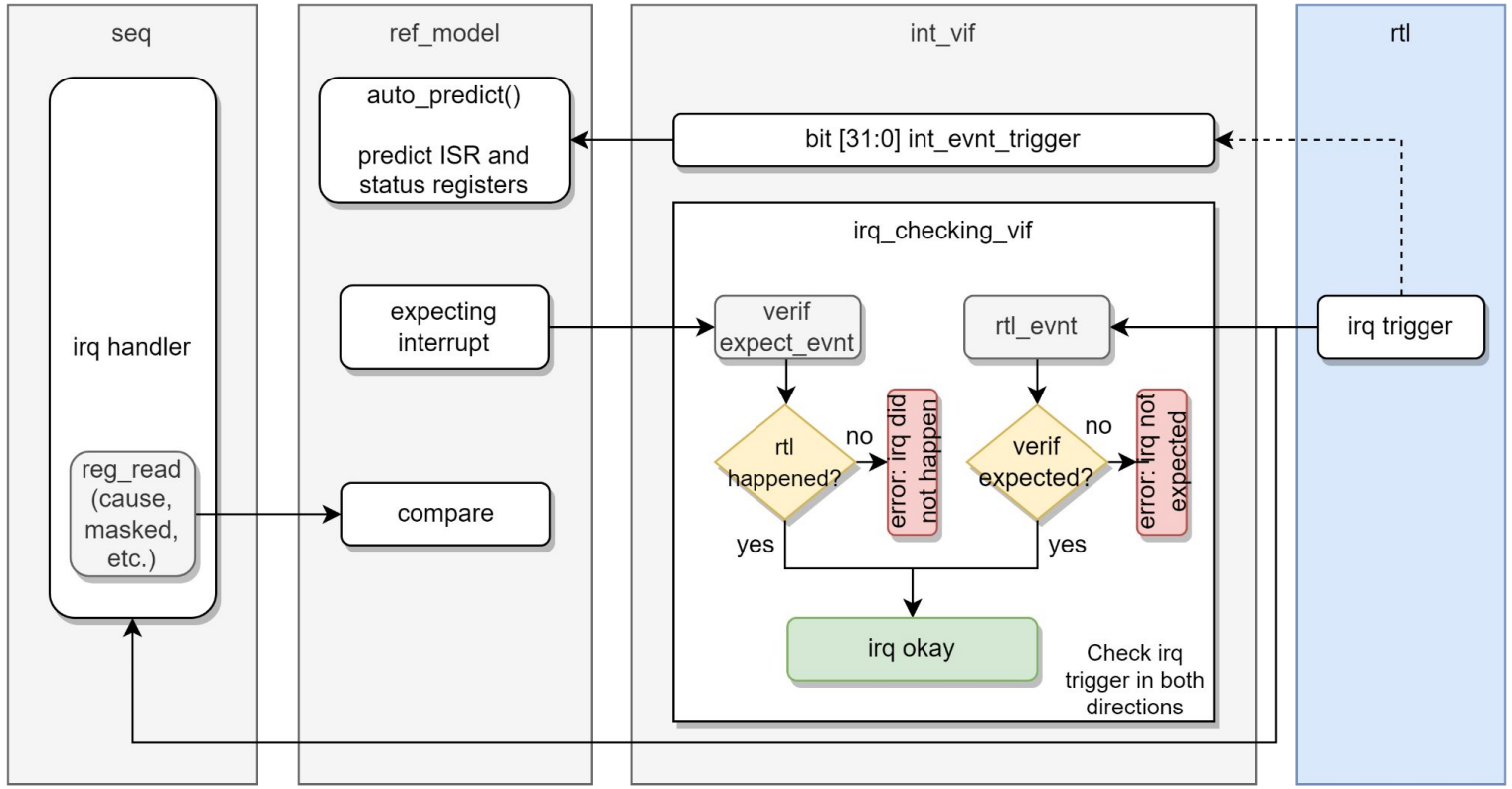
Explaining the concept

- Problem #1:
  - Verification often isn’t in sync with the RTL
- Solution #1:
  - Separate verification checkers from the register prediction
  - Create general task for automatic prediction
  - Prediction is based on the trigger coming from the design
  - **Goal:** *Check the aggregation of the interrupt*

Explaining the concept

- Problem #2:
  - Verifying the interrupt trigger
  - Checking must be done in both directions
- Solution #2:
  - Create interrupt checking interface
  - Interface is configurable and applicable to both edge and level triggered IRQs
  - **Goal:** *Verify the interrupt trigger*

Flow of checking



Handling status and control registers

- Prediction using macros

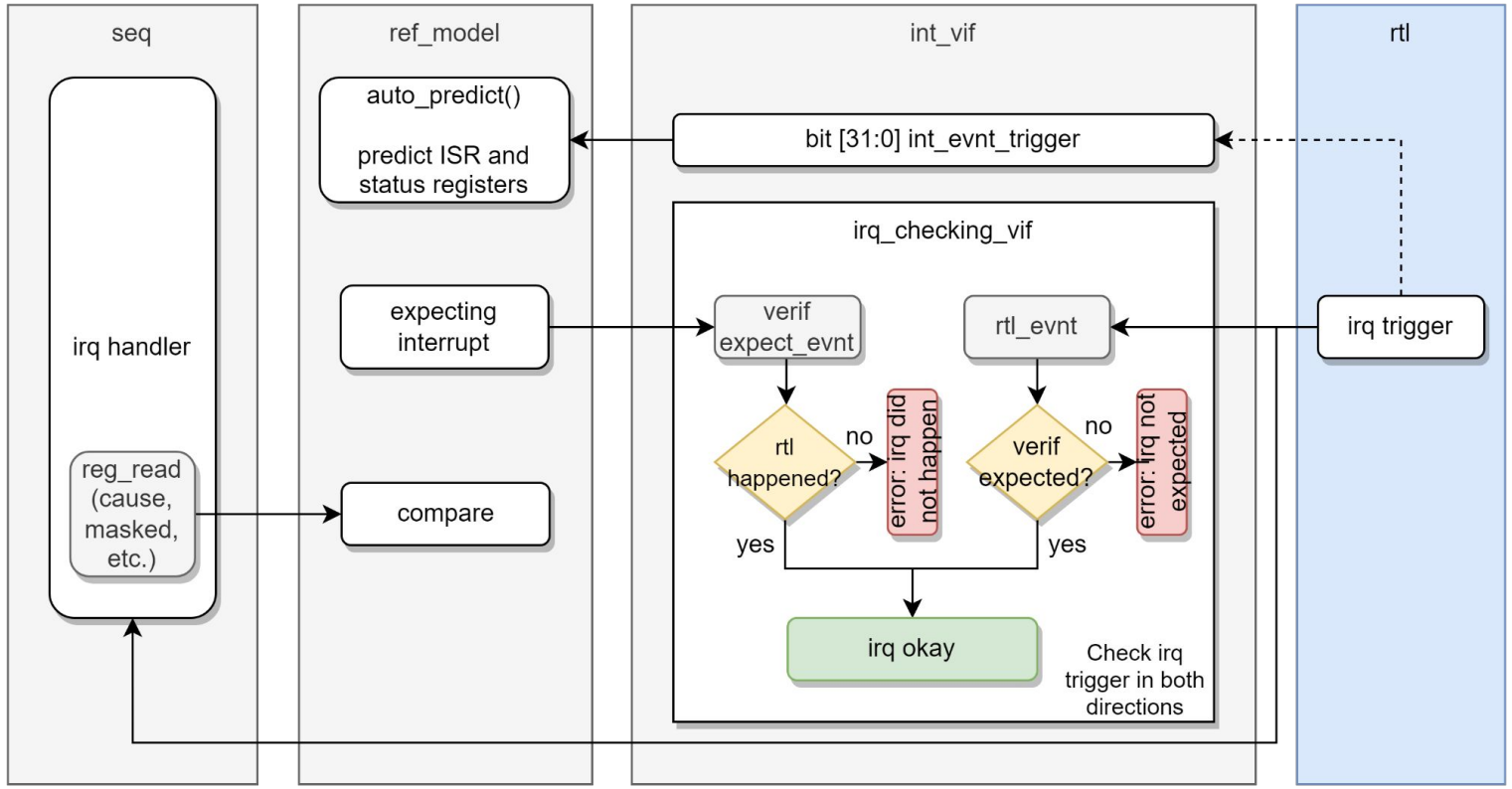
```
`define IRQ_AUTO_PREDICT(isr_reg_name, isr_field, irq_reg_name, irq_field,
reg_mask_name, reg_mask)\
forever begin\
    int position = reg_model.IRQ_BLOCK.irq_reg_name.irq_event.get_lsb_pos();\
    @(posedge int_vif.int_event_trigger[position]);\
    if(!reg_model.IRQ_BLOCK.reg_mask_name.reg_mask.get_mirrored_value()) begin\
        `uvm_info(get_full_name(), $sformatf("Predicting ISR reg due to %s",
reg_model.IRQ_BLOCK.irq_reg_name.irq_event.get_name()), UVM_HIGH)\
        assert(reg_model.IRQ_BLOCK.isr_reg_name.isr_field.predict(.value(1)));\
        assert(reg_model.IRQ_BLOCK.irq_reg_name.irq_event.predict(.value(1)));\
    end\
end\
```

Handling status and control registers

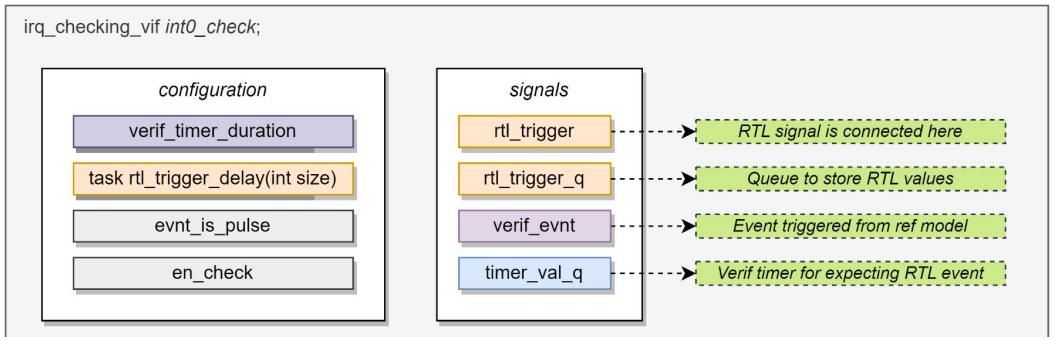
- auto_predict()

```
task auto_predict();
    fork
        //IRQ_REG_A
        `IRQ_AUTO_PREDICT(ISR, isr_A, IRQ_REG_A, int0, IRQ_MASK_A, mask0)
        . . .
        `IRQ_AUTO_PREDICT(ISR, isr_A, IRQ_REG_A, int31, IRQ_MASK_A, mask31)
        //IRQ_REG_B
        `IRQ_AUTO_PREDICT(ISR, isr_B, IRQ_REG_B, int0, IRQ_MASK_B, mask0)
        . . .
        `IRQ_AUTO_PREDICT(ISR, isr_B, IRQ_REG_B, int31, IRQ_MASK_B, mask31)
    join_none
endtask
```

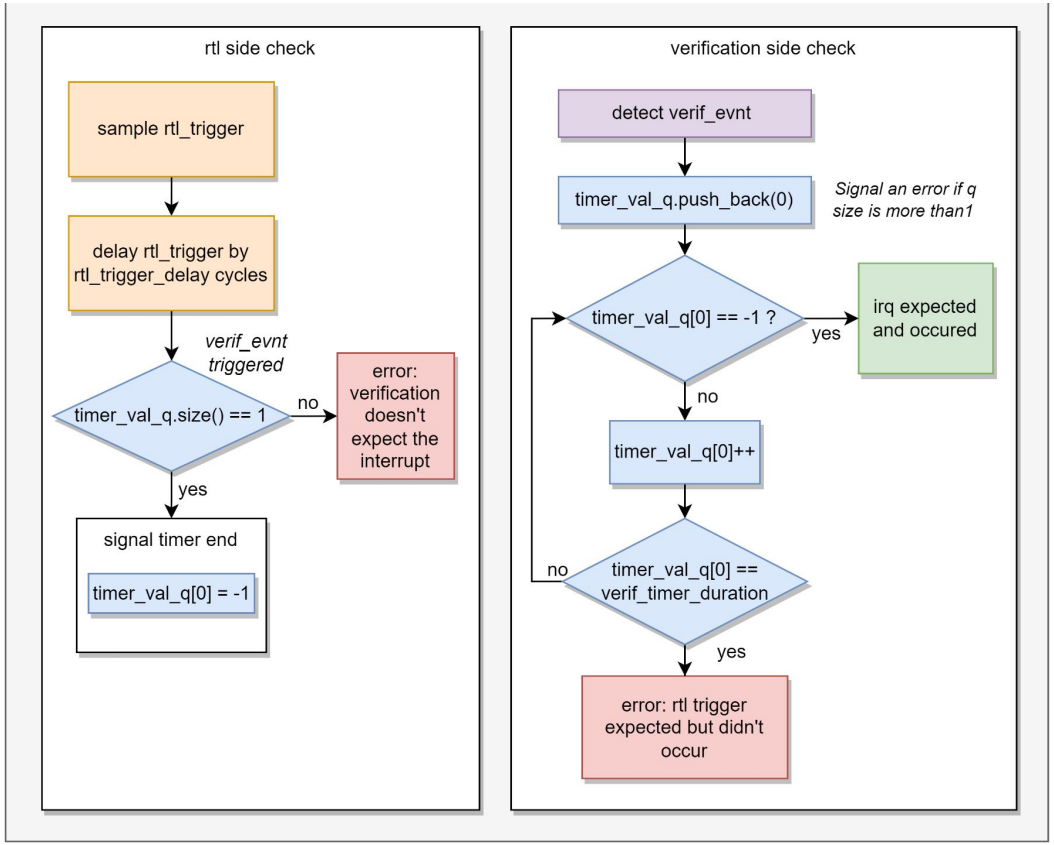
Flow of checking



How does irq_checking_vif work?



- Verification must be triggered first
- Two scenarios:
  - RTL trigger happens before verification
  - Verification happens before RTL



Checking the RTL trigger

- *rtl_trigger_delay(int size)*

```
task rtl_trigger_delay(int size);
   if (size == 0) `uvm_error("%m", "Size cannot be zero")
   repeat(size) rtl_trigger_q.push_back(0);
endtask
```

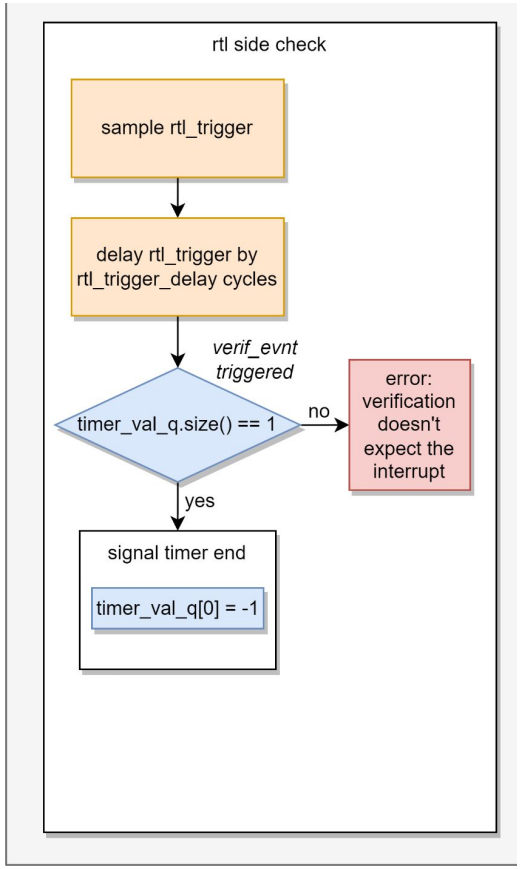
Checking the RTL trigger

- Sampling and delaying is done in an always block

```
always @(posedge clk) begin
    void'(rtl_trigger_q.pop_front());
    void'(rtl_trigger_q.push_back(rtl_trigger));
end
```

Checking the RTL trigger

- Wait for *rtl_trigger_q[0]* to change from 0 to 1
- Call a task to check the occurrence of *verif_evnt*
- Check the status of queue *timer_val_q*
- If *verif_evnt* was not triggered:
  - ERROR: Verification doesn’t expect the interrupt
- If the *verif_evnt* was already expected:
  - *timer_val_q[0]* will be set to -1



Checking the RTL trigger

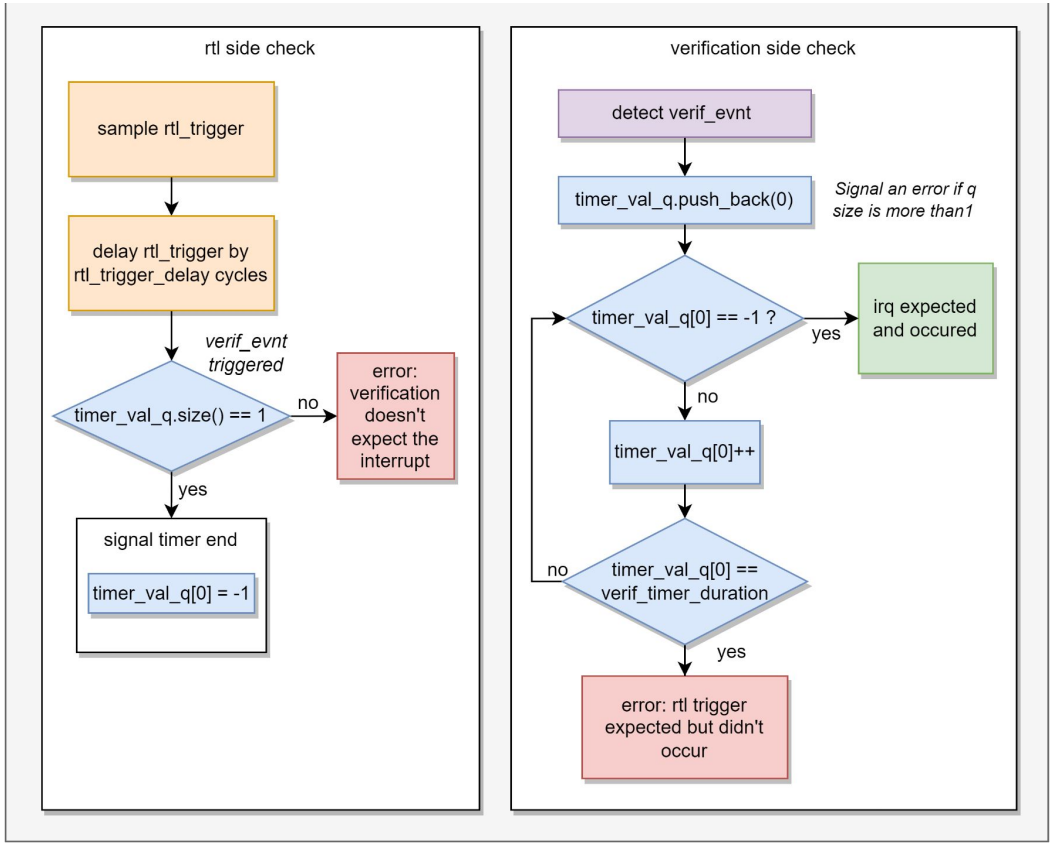
```
always @(posedge clk) //for pulses __|**|__
     if (evnt_is_pulse && rtl_trigger_q[0])
          check_if_evnt_expected();


always @(posedge rtl_trigger_q[0] iff !evnt_is_pulse ) //for level __|******
          check_if_evnt_expected();


function void check_if_evnt_expected();
     `uvm_info($sformatf("%m"), "rtl event triggered", UVM_HIGH)
     if (en_check)
          if (timer_val_q.size == 1) timer_val_q[0] = -1;
          else `uvm_error($sformatf("%m"), "RTL event occurred without it being expected.")
endfunction
```

Checking verification event

- Initialize the timer_val_q value
- Start the timer
- verif_timer_duration sets the time window for RTL event



Checking verification event

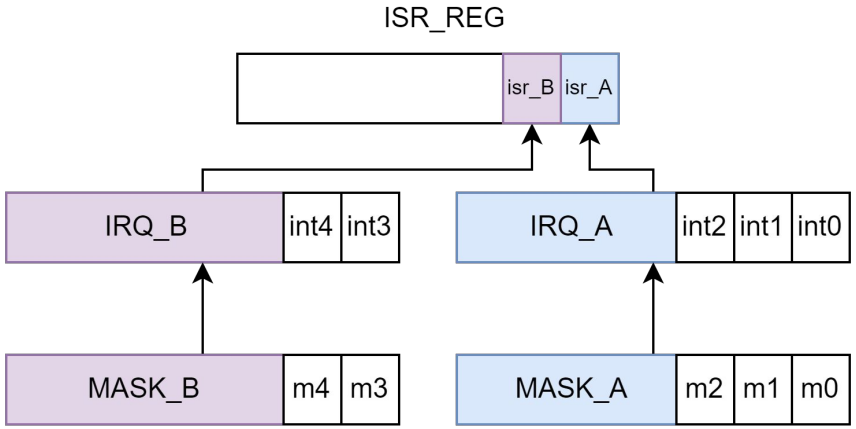
```
always @(verif_evnt)
    if (en_check && (evnt_is_pulse || rtl_trigger_q[0] === 0)) begin
        if(!timer_val_q.size()) timer_val_q.push_back(0);
        else `uvm_error($sformatf("%m"), "verif event expected more than once")
     `uvm_info($sformatf("%m"), "verif event triggered", UVM_HIGH)
     if (timer_val_q.size == 1)
            fork
               check_expected_ev_happens(); //check if rtl signal happens
            join_none
    end
```

Checking verification event

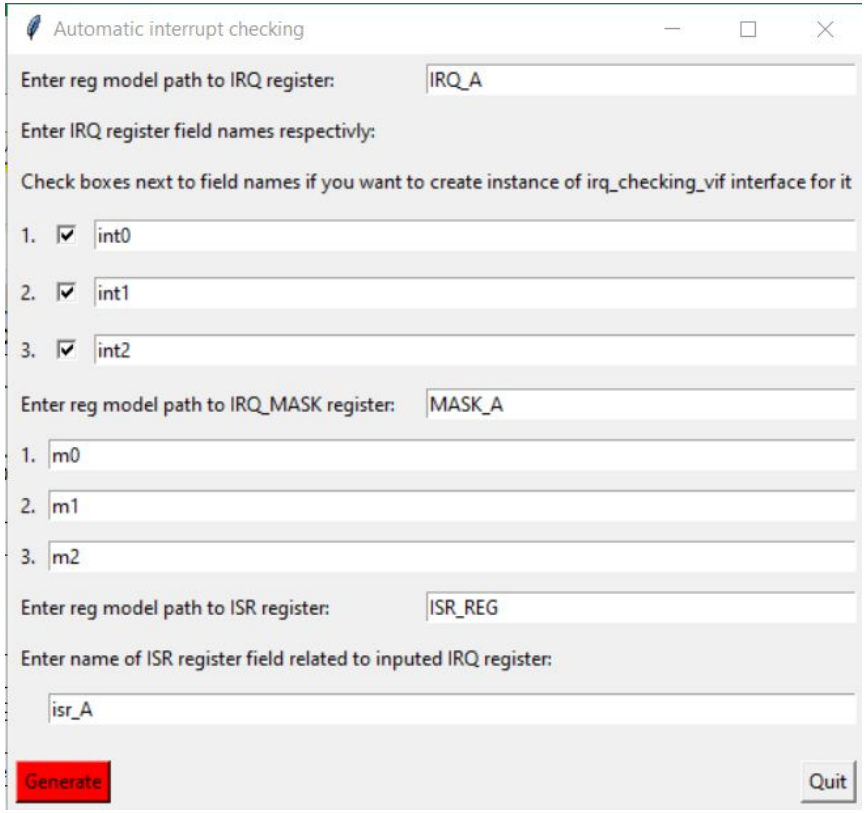
```
task check_expected_ev_happens();
    while (timer_val_q.size > 0) begin
        @(posedge clk);
        if (timer_val_q[0] == -1) begin
            void'(timer_val_q.pop_front());
            `uvm_info($sformatf("%m"), "RTL event expected and occured.", UVM_HIGH)
        end
        if (timer_val_q[0] == verif_timer_duration) begin
            void'(timer_val_q.pop_front());
            `uvm_error($sformatf("%m"), "RTL event didn't occur although expected in
verification.")
        end else
             timer_val_q[0] += 1;
    end // while
endtask // check_expect_happens
```

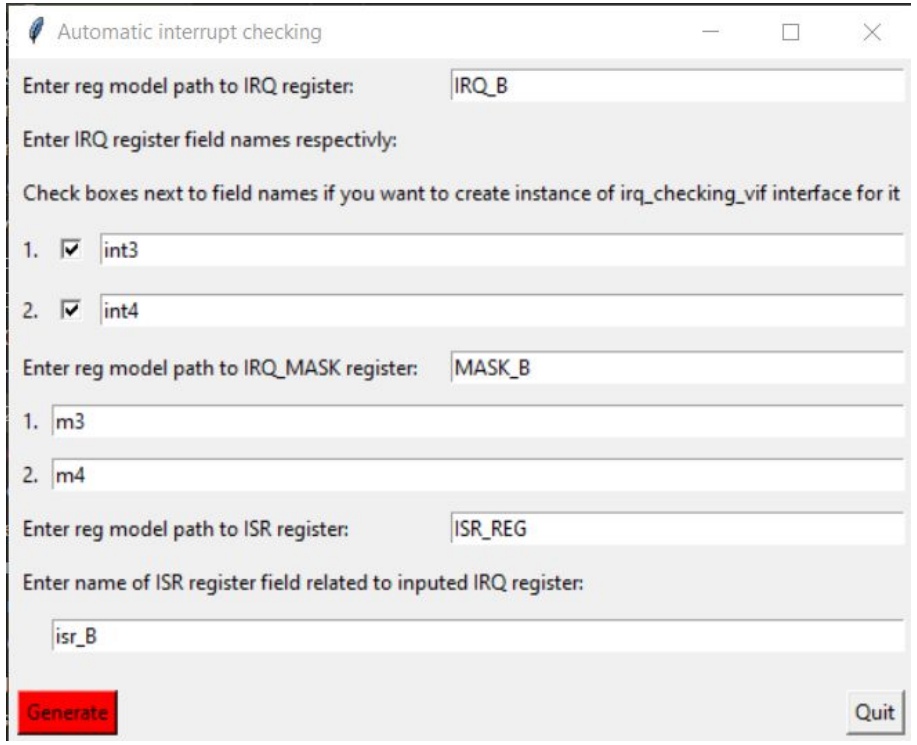
Automating with Python

- Take advantage of repeating IRQ behavior and standard verification procedure
- Create script based on main building blocks of code



Automating with Python





Automating with Python

```
fork
   `IRQ_AUTO_PREDICT(ISR_REG, isr_A, IRQ_A, int0, MASK_A, m0) //IRQ_A
   `IRQ_AUTO_PREDICT(ISR_REG, isr_A, IRQ_A, int1, MASK_A, m1) //IRQ_A
   `IRQ_AUTO_PREDICT(ISR_REG, isr_A, IRQ_A, int2, MASK_A, m2) //IRQ_A
   `IRQ_AUTO_PREDICT(ISR_REG, isr_B, IRQ_B, int3, MASK_B, m3) //IRQ_B
   `IRQ_AUTO_PREDICT(ISR_REG, isr_B, IRQ_B, int4, MASK_B, m4) //IRQ_B
join_none

irq_checking_vif trigger_check_IRQ_A_int0;
irq_checking_vif trigger_check_IRQ_A_int1;
irq_checking_vif trigger_check_IRQ_A_int2;
irq_checking_vif trigger_check_IRQ_B_int3;
irq_checking_vif trigger_check_IRQ_B_int4;
```

Automating with Python

- Upgrading the script:
  - Remove gui
  - Generate everything based on Excel Spreadsheet
  - Add additional building blocks of code for your specific project

Conclusion – Why use auto prediction?

- Checking is centralized in one location
- An independent standardized mechanism is created
- Debug and implementation process easier and faster

Conclusion – Why use irq_checking_vif?

- Checking in both directions
- Precise checking because the detection window can be narrow
- Configurable and applicable to all interrupts (pulses and levels)

Results

- Increased code reusability
- Partially automated checking
- Decreased the time needed in order to fully verify the interrupts
- Reusable on the same project, but also on company level

Questions?

Thank you!

aleksandrad@thevtool.com

nemanjas@thevtool.com

yoavf@chain-reaction.io

itayh@chain-reaction.io