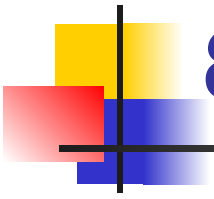
UVM

UVM Generator

uvmgen

```
UVM Version : 2
Complete Env.(1) OR Individual Template(2)? : 1
Want to create your own methods[Instead of uvm shorthand macros]? : n
RAL env? : 0
Env. Name : environment
Agents? : y
Name of master agent:  : ethernet
Name of sequencer in ethernet master agent:  : eth_sequencer
Name of driver in ethernet master agent:  : eth_driver
Name of monitor in ethernet master agent:  : eth_monitor
Name of interface related to ethernet master agent:  : eth_intf
Name of transaction in ethernet master agent:  : eth_data
BU class for this transaction? : n
Name of slave agent:  : slave
Name of sequencer in slave slave agent:  : slave_sequencer
Name of driver in slave slave agent:  : slave_driver
Name of monitor in slave slave agent:  : slave_monitor
Name of physical interface related to slave slave agent:  : slave_intf
Name of transaction related to slave slave agent:  : eth_data
Driver information for the slave agent slave :  :
Driver Type : Driver, PULL DRIVER (uvm_driver)
Driver information for the master agent ethernet :  :
Driver Type : Driver, PULL DRIVER (uvm_driver)
Scoreboard? : y
Name of Scoreboard Class : scoreboard
```



8x8 Router UVM testbench

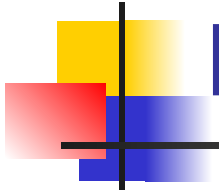
/home/bgreene/ELEN613/RT_uvm

Source code

- proj/src

Compile + Run directory

- Proj/run



Ethernet Agent

Verilog files

- ethernet_eth_data.sv
- ethernet_eth_driver.sv
- ethernet_eth_intf.sv
- ethernet_eth_monitor.sv
- ethernet_eth_sequencer.sv
- ethernet_sequence_library.sv

Ethernet_eth_data

```
class eth_data extends uvm_sequence_item;

  typedef enum {READ, WRITE } kinds_e;
  rand kinds_e kind;
  typedef enum {IS_OK, ERROR} status_e;
  rand status_e status;
  rand byte sa;

  // ToDo: Add constraint blocks to prevent error injection
  // ToDo: Add relevant class properties to define all transactions
  // ToDo: Modify/add symbolic transaction identifiers to match
   rand bit idle;
   rand bit [3:0] delay;
   rand bit [2:0] src,dst;
   rand bit [31:0] payload;

  constraint eth_data_valid {
    // ToDo: Define constraint to make descriptor valid
    delay == 10;
    idle == 0;
    status == IS_OK;
  }
  `uvm_object_utils_begin(eth_data)

    // ToDo: add properties using macros here

    `uvm_field_enum(kinds_e,kind,UVM_ALL_ON)
    `uvm_field_enum(status_e,status, UVM_ALL_ON)
    `uvm_field_int(idle, UVM_ALL_ON)
    `uvm_field_int(delay, UVM_ALL_ON)
    `uvm_field_int(src, UVM_ALL_ON)
    `uvm_field_int(dst, UVM_ALL_ON)
    `uvm_field_int(payload, UVM_ALL_ON)
  `uvm_object_utils_end

  extern function new(string name = "Trans");
endclass: eth_data


function eth_data::new(string name = "Trans");
  super.new(name);
endfunction: new
```

generated already

enables automation expands different tasks & functions

Ethernet_eth_driver

```
task eth_driver::reset_phase(uvm_phase phase);
    super.reset_phase(phase);
    // ToDo: Reset output signals
     drv_if.mck.frame_n <= '1;
     drv_if.mck.valid_n <= '1;
     drv_if.mck.di <= 'x;
     repeat (1) @(drv_if.mck);
endtask: reset_phase

task eth_driver::run_phase(uvm_phase phase);
   super.run_phase(phase);
   repeat (50) @(drv_if.mck);
fork
      tx_driver(0); <-just 1 channel
   join
endtask: run_phase

task eth_driver::tx_driver(int i);
 forever begin
      eth_data tr;
      // ToDo: Set output signals to their idle state
      this.drv_if.master.async_en      <= 0;
      `uvm_info("environment_DRIVER", "Starting transaction...",UVM_LOW)
      seq_item_port.get_next_item(tr);
       if (tr.dst==7 && tr.src==7) send1pkt(tr.src,tr); <-
      seq_item_port.item_done();
      `uvm_info("environment_DRIVER", "Completed transaction...",UVM_LOW)
      `uvm_info("environment_DRIVER", tr.sprint(),UVM_HIGH)
      `uvm_do_callbacks(eth_driver,eth_driver_callbacks,
                  post_tx(this, tr))
   end
endtask : tx_driver
```

```
task eth_driver::send1pkt(int i,eth_data tr);
   if (i!== tr.src) $display("ERROR %d != %d",i,tr.src);
   if (tr.idle)
     begin
       repeat(tr.delay)  @(drv_if.mck);
       return;
     end
   $display($time, ": Sending packet src=%1d:dst=%1d",tr.src,tr.dst);
   repeat(5) @(drv_if.mck);
   drv_if.mck.frame_n[tr.src] <= 1'b0;
   drv_if.mck.di[tr.src] <= tr.dst[0];
   @(drv_if.mck)  drv_if.mck.di[tr.src] <= tr.dst[1];
   @(drv_if.mck)  drv_if.mck.di[tr.src] <= tr.dst[2];
   @(drv_if.mck)  drv_if.mck.di[tr.src] <= 1'b0;
   // Padding
   repeat(1) @(drv_if.mck) ;
   for (int i=0;i<32;i=i+1) begin
     drv_if.mck.valid_n[tr.src] <= 1'b0;
     drv_if.mck.di[tr.src] <= tr.payload[i];
     drv_if.mck.frame_n[tr.src] <= i==31;
     @(drv_if.mck);
   end
   drv_if.mck.valid_n[tr.src] <= 1'b1;
   drv_if.mck.di[tr.src] <= 1'bx;
   repeat (5)  @(drv_if.mck);

endtask:send1pkt
```

Ethernet_eth_monitor

```
task eth_monitor::tx_monitor();
  forever begin
    eth_data tr;
    // ToDo: Wait for start of transaction

    `uvm_do_callbacks(eth_monitor,eth_monitor_callbacks,
                pre_trans(this, tr))
    `uvm_info("environment_MONITOR", "Starting transaction...",UVM_LOW)
    // ToDo: Observe first half of transaction
    rcv1pkt(7,tr);
    `uvm_do_callbacks(eth_monitor,eth_monitor_callbacks,
                pre_ack(this, tr))
    // ToDo: React to observed transaction with ACK/NAK
    `uvm_info("environment_MONITOR", "Completed transaction...",UVM_LOW)
    `uvm_info("environment_MONITOR", tr.sprint(),UVM_LOW)
    `uvm_do_callbacks(eth_monitor,eth_monitor_callbacks,
                post_trans(this, tr))
    mon_analysis_port.write(tr);
  end
endtask: tx_monitor
```

```
task eth_monitor::run_phase(uvm_phase phase);
  super.run_phase(phase);
 // phase.raise_objection(this,""); //Raise/drop objections in sequence file
  fork
    tx_monitor();
  join
 // phase.drop_objection(this);

task automatic eth_monitor::rcv1pkt(int i, ref eth_data tr);
   eth_data tmp;
    tmp = new();
    tmp.dst=i;
    while (mon_if.pck.frame_n[i]!=='0) @(mon_if.pck) ;
    while (mon_if.pck.valid_n[i]!=='0) @(mon_if.pck) ;
    for (int j=0;j<32;j=j+1) begin
tmp.payload[j] <= mon_if.pck.di[i];
      @(mon_if.pck);
    end
    tr = tmp;
endtask:rcv1pkt
```