Clock Item Class

```
`include "uvm_macros.svh"

import uvm_pkg::*;

class ClkItem extends uvm_sequence_item;

   //operation to be performed - assigned according to the sequence called

  typedef enum {

    CLK_START, //start clock

    CLK_STOP, //stop clock

    CLK_WAIT, //gate clock

    CLK_FREQ //change frequency

  } op_type_t;


  //declaring clock variables random - corresponding to user defined variables

  rand op_type_t        op_type;

  rand int       clk_name   [];

  rand bit       init       [];

  rand time      period      [];

  rand time      phase_shift [];

  rand time      cyc_jitter  [];

  rand int                              duty_c [];

  rand int            t_end [];

  rand int        change_t_n [];

  rand time          new_tp [];

  rand time                        dull_t [];

  rand int           n_wait [];

  rand int          at_wait [];

  rand int              ppm [];

  rand int                          wait_list [];

  rand int                          freq_list [];
```

```
//class constructor
function new(string name = "ClkItem");
  super.new(name);
endfunction

//field macros
`uvm_object_utils_begin(ClkItem)
`uvm_field_array_int( wait_list, UVM_DEFAULT | UVM_NOPACK)
`uvm_field_array_int( freq_list, UVM_DEFAULT | UVM_NOPACK)
`uvm_field_enum(op_type_t, op_type, UVM_DEFAULT | UVM_NOPACK)
  `uvm_field_array_int( clk_name, UVM_DEFAULT | UVM_NOPACK)
  `uvm_field_array_int (              init, UVM_DEFAULT | UVM_NOPACK)
  `uvm_field_array_int (            period, UVM_DEFAULT | UVM_NOPACK | UVM_TIME)
`uvm_field_array_int (        phase_shift, UVM_DEFAULT | UVM_NOPACK | UVM_TIME)
`uvm_field_array_int (        cyc_jitter, UVM_DEFAULT | UVM_NOPACK | UVM_TIME)
`uvm_field_array_int (        dull_t, UVM_DEFAULT | UVM_NOPACK | UVM_TIME)
`uvm_field_array_int (          duty_c, UVM_DEFAULT | UVM_NOPACK)
`uvm_field_array_int (        t_end, UVM_DEFAULT | UVM_NOPACK)
`uvm_field_array_int (       change_t_n, UVM_DEFAULT | UVM_NOPACK)
`uvm_field_array_int (        new_tp, UVM_DEFAULT | UVM_NOPACK | UVM_TIME)
`uvm_field_array_int (       n_wait, UVM_DEFAULT | UVM_NOPACK)
`uvm_field_array_int (       at_wait, UVM_DEFAULT | UVM_NOPACK)
`uvm_field_array_int (       ppm, UVM_DEFAULT | UVM_NOPACK)
  `uvm_object_utils_end

virtual task displayAll();
  `uvm_info(get_full_name(), $sformatf("Got Transaction clk_name=%s", clk_name.size()), UVM_LOW);
endtask
```

```
endclass: ClkItem

//`timescale 1ns/1ps

class ClkAgentUserClass extends ClkItem;
  `uvm_object_utils(ClkAgentUserClass)
  int clk_list[];

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
  /*  set these variables if clock variations have to be specific, without being random    */
  /*  all (except c_width, clk_list_t, wait_clk_list and freq_clk_list) the parameters from here             will
be set to default values if not mentioned                                                                    */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

//no of clocks to be generated
  `define c_width 2

  //names of clock
  typedef enum {
  CLK_500MHz,
  CLK_250MHz
  } clk_list_t;

  //these parameters are corresponding to clk_list_t
  // initial values of the Clk output pins
  bit clk_init[] = {0, 0};
  //time period - in timeunit - default value = 1
  realtime t_period[] = {2, 5};
```

```
//start time of each clock in timeunit - default value = 0
realtime t_phase[] = {0.8, 1.5};
//duty cycle of each clock in percentage - defaut value = 50
real duty_cycle []= {50, 33.333};
//cycle to cycle jitter - in timeunit - default value = 0.01
realtime cyc_cyc_jitter[] = {0.010,0.025};
//ppm value - default value = 100
int ppm_val [] = {100,100}; //ppm value
//no of cycles after which clock to be stopped - (n=-1 implies dont stop) - default value = -1
int stop_cycle [] = {30,29};

//list of clks to change freq -  names should be acc to clk_list_t
int freq_clk_list[] = { CLK_500MHz , CLK_250MHz, CLK_250MHz};
//no of cycles after which freq is to be changed - default value = running cycle at that instant
int change_freq_n [] = {15, 3, 11};
//value of new time period - in timeunit - default value = 1
realtime new_t [] = {5, 3, 4};
//specify dull time - in timeunit - default value = 0.1
realtime dull_time [] = {0.8, 0.1, 0.1};

//list of clocks to be gated -  names should be acc to clk_list_t
int wait_clk_list[] = { CLK_250MHz, CLK_500MHz};
//pause clk after nth cycle - default value = running cycle at that instant
int wait_at [] = { 13, 20 };
//no of cycles for which clk has to be gated - default value = running cycle at that instant
int wait_n [] = { 7, 4};

/*                                        CONSTRAINTS
                                 */
```

```
//for clock name, user intervention not recommended
constraint clk_ind {clk_name.size() == `c_width;}
//for initial values
constraint clk_init_cons {init.size() == `c_width;
                         foreach(clk_init[i])
                           init[i] == clk_init[i];
                         //init[i]==0;
                                                                      }
//time period
constraint t_period_cons {//period.size() == `c_width;
                         period.size() == t_period.size();
                         foreach (period[i])
                           //period[i] inside {[1000:10000]};
                           period[i] == int'(t_period[i]*1000);
                        }
//initial offset/ start time
constraint t_phase_cons {//phase_shift.size() == `c_width;
                                                           phase_shift.size() == t_phase.size();
                        foreach (phase_shift[i])
                          //phase_shift[i] inside {[0:2000]};
                          phase_shift[i] == int'(t_phase[i]*1000);
                        }
//duty cycle
constraint duty_cons {//duty_c.size() == `c_width;
                                                   duty_c.size() == duty_cycle.size();
                     foreach(duty_c[i])
                       //duty_c[i] inside {[20000:80000]};
                       duty_c[i] == int'(duty_cycle[i]*1000);
```

```
                    }
//cyc to cyc jitter
constraint jitter_cons {//cyc_jitter.size() == `c_width;
                                                  cyc_jitter.size() == cyc_cyc_jitter.size();
                     foreach(cyc_jitter[i])
                       //cyc_jitter[i] inside {[10:25]};
                       cyc_jitter[i] == int'(cyc_cyc_jitter[i]*1000);
                      }
//ppm value
constraint ppm_cons {//ppm.size() == `c_width;
                                                  ppm.size() == ppm_val.size();
                    foreach(ppm[i])
                     // ppm[i] inside {[10:1000]};
                      ppm[i] == ppm_val[i];
                   }
//no of cycles after which clock to be stopped
constraint stop_cons {//t_end.size() == `c_width;
                                                  t_end.size() == stop_cycle.size();
                    foreach(t_end[i])
                      //t_end[i] inside {[20:50]};
                      t_end[i] == stop_cycle[i];
                    }
//list of clocks to be gated
constraint wait_list_cons {wait_list.size() == wait_clk_list.size();
                    foreach(wait_list[i])
                      //wait_list[i] inside {[0:1]};
                      wait_list[i] == wait_clk_list[i];
                        }
//pause clk after nth cycle
```

```
constraint wait_at_cons {at_wait.size() == wait_clk_list.size();
                    foreach(at_wait[i])
                     // at_wait[i] inside {[0:10]};
                      at_wait[i] == wait_at[i];
                       }
//no of cycles for which clk has to be gated
constraint wait_n_cons {n_wait.size() == wait_clk_list.size();
                    foreach(n_wait[i])
                     // n_wait[i] inside {[1:10]};
                      n_wait[i] == wait_n[i];
                      }
//list of clocks for which freq has to be changed
constraint freq_list_cons {freq_list.size() == freq_clk_list.size();
                    foreach(freq_list[i])
                     // freq_list[i] inside {[0:1]};
                      freq_list[i] == freq_clk_list[i];
                        }
//value of new time period
constraint new_period_cons {new_tp.size() == freq_clk_list.size();
                         foreach (new_tp[i])
                          // new_tp[i] inside {[1000:10000]};
                           new_tp[i] == int'(new_t[i]*1000);
                          }
//no of cycles after which freq is changed
constraint change_freq_cons {change_t_n.size() == freq_clk_list.size();
                    foreach(change_t_n[i])
                     // change_t_n[i] inside {[1:10]};
                      change_t_n[i] == change_freq_n[i];
                         }
```

```
  //specify dull time
  constraint dull_cons {dull_t.size() == freq_clk_list.size();
                      foreach(dull_t[i])
                        // dull_t[i] inside {[100:2000]};
                        dull_t[i] == int'(dull_time[i]*1000);
                     }



  function new(string name = "ClkAgentUserClass");
    super.new(name);
    clk_list = new[`c_width];
    foreach(clk_list[i])
      clk_list[i] = i;
   // randomize_func();
  endfunction



endclass
```

Clock Sequence

```
// including transation file
//`include "clk_item.sv"
//`include "clk_item.sv"
//this is base seq class
//`include "clk_agent_pkg.sv"
class ClkBaseSequence extends uvm_sequence#(ClkAgentUserClass);
  `uvm_object_utils(ClkBaseSequence)
  ClkAgentUserClass it;
  static int inst_cnt;

  //class constructor
  function new(string name = "ClkBaseSequence");
    super.new(name);
  endfunction: new

  //body of sequence
  virtual task body();
    inst_cnt++;
    it = ClkAgentUserClass::type_id::create("it");
  endtask: body

endclass: ClkBaseSequence



//**********************************************************



//sequence to start the clk
class ClkStartSequence extends ClkBaseSequence;
  `uvm_object_utils(ClkStartSequence)

  //class constructor
  function new(string name = "ClkStartSequence");
    super.new(name);
  endfunction: new
  //body
  virtual task body();
    super.body();
    `uvm_info(get_type_name(),"starting item in start clk", UVM_LOW)
    start_item(it);
    //randomizing ClkItem with user defined parameters
    if (!it.randomize() with {
      op_type == CLK_START;
      clk_name.size() == clk_list.size();
      foreach(clk_list[i])
        clk_name[i] == clk_list[i];
    }) `uvm_error("CLK_START_SQNC", "\nRandomization failed\n")

    finish_item(it);
  endtask: body
endclass : ClkStartSequence


//************************************************************


//sequence to stop the clk
class ClkStopSequence extends ClkBaseSequence;
  `uvm_object_utils(ClkStopSequence)

  //class constructor
  function new(string name = "ClkStopSequence");
    super.new(name);
  endfunction: new

  //body
  virtual task body();
    super.body();
    `uvm_info(get_type_name(),"starting item in stop clk", UVM_LOW)
    start_item(it);

    //randomizing ClkItem with user defined parameters
    if (!it.randomize() with {
      op_type == CLK_STOP;
      foreach(clk_list[i])
        clk_name[i] == clk_list[i];
    }) `uvm_error("CLK_STOP_SQNC", "\nRandomization failed\n")

    finish_item(it);
  endtask: body

endclass: ClkStopSequence


//*************************************************************


//seq to gate the clk for specified number of cycles
class ClkWaitSequence extends ClkBaseSequence;
  `uvm_object_utils(ClkWaitSequence)

  //class constructor
  function new(string name = "ClkWaitSequence");
    super.new(name);
  endfunction: new

  //body
  virtual task body();
    super.body();
    `uvm_info(get_type_name(),"starting item in wait clk",UVM_LOW)
    start_item(it);
    //randomizing ClkItem with user defined parameters
    if (!it.randomize() with {
      op_type == CLK_WAIT;
      foreach(clk_list[i])
        clk_name[i] == clk_list[i];
    }) `uvm_error("CLK_START_SQNC", "\nRandomization failed\n")

    finish_item(it);
  endtask: body
endclass : ClkWaitSequence

//************************************************************


//seq to change freq of the clk
class ClkFreqSequence extends ClkBaseSequence;
  `uvm_object_utils(ClkFreqSequence)

  //class constructor
  function new(string name = "ClkFreqSequence");
```

```
    super.new(name);
  endfunction: new

  //body
  virtual task body();
    super.body();
    `uvm_info(get_type_name(),"starting item in change freq clk", UVM_LOW)
    start_item(it);
    //randomizing ClkItem with user defined parameters
    if (!it.randomize() with {
      op_type == CLK_FREQ;
      foreach(clk_list[i])
        clk_name[i] == clk_list[i];
    }) `uvm_error("CLK_FREQ_SQNC", "\nRandomization failed\n")

    finish_item(it);
  endtask: body
endclass : ClkFreqSequence


//***********************************************************
```

Clock Driver

```
`include "uvm_macros.svh"

import uvm_pkg::*;

//`include "clk_agent_pkg.sv"

class ClkDriver extends uvm_driver #(ClkAgentUserClass);

 `uvm_component_utils(ClkDriver)

 protected process proc_start_clk [];

 int n[];

 real period_[];

 int first = 0;

 bit clk_dummy[];

 bit wait_en []; //this is set to 1 if wait is enabled


 virtual ClkIf vif;

 int walk;

 event done;


 //class constructor

 function new(string name = "ClkDriver", uvm_component parent);

  super.new(name, parent);

 endfunction


 //build phase

 function void build_phase(uvm_phase phase);

  super.build_phase(phase);

  if(! uvm_config_db #(virtual ClkIf)::get(this,"","vif",vif))

   `uvm_error("NO_VIF",{"virtual interface must be set for: ",get_full_name(),".vif"});

  endfunction
 extern virtual task         stopClk (input ClkAgentUserClass it);

 extern virtual task         startClk (input ClkAgentUserClass it);

 extern virtual task         freqClk (input ClkAgentUserClass it);

 extern virtual task         waitClk (input ClkAgentUserClass it);

 extern virtual task         run_phase(uvm_phase phase);

 extern virtual task                     start_clock(input logic init_clk_val, input logic [31:0] clk_name,  time
phase_shift, realtime period, input time cyc_jitter, input int d_cyc, input int wait_n, input int wait_at,
input int ppm);

 extern virtual task                     generate_clk(input logic init_clk_val, input logic [31:0] clk_name, input
realtime period, input time cyc_jitter, input int d_cyc, input int ppm);


endclass: ClkDriver


 //this class takes transaction and starts clks simultaneously

 task ClkDriver::startClk(input ClkAgentUserClass it);

   `uvm_info(get_type_name(), $sformatf("it.clk_name.size()=%d", it.clk_name.size()), UVM_DEBUG)


   //starting clocks simultaneously

   for(int i=0; i< it.clk_name.size(); i++) begin

      int walk=i;

   fork

     start_clock(it.init[walk], walk, it.phase_shift[walk], it.period[walk]/1000.0, it.cyc_jitter[walk],
it.duty_c[walk], it.n_wait[walk], it.at_wait[walk], it.ppm[walk]);

   join_none

   end


 endtask: startClk




   //this task assign a process to individual clk
```

```
      task ClkDriver::start_clock(input logic init_clk_val,input logic [31:0] clk_name, input time
phase_shift,input realtime period, input time cyc_jitter, input int d_cyc, input int wait_n, input int
wait_at, input int ppm);

       real phase_shift1;


       //phase shift modelling

      phase_shift1 = phase_shift/1000.0;


       if (phase_shift1!=0) begin

       #(phase_shift1);

         end

       //updating current clock period

       period_[clk_name] = period;

       `uvm_info("in driver:", $sformatf("clk[%0d] starts now", clk_name), UVM_DEBUG)


       //setting initial value

        vif.clk[clk_name] = init_clk_val;

       vif.clk_ideal[clk_name] = init_clk_val;

       clk_dummy[clk_name] = init_clk_val;

       //starting process

       proc_start_clk[clk_name] = process::self();

       //generating clk

       generate_clk ( init_clk_val, clk_name, period, cyc_jitter,  d_cyc, ppm);



    endtask


       //this task generates the clk with the specified jitter and specified characteristics

    task ClkDriver::generate_clk (input logic init_clk_val, input logic [31:0] clk_name, input realtime
period, input time cyc_jitter, input int d_cyc, input int ppm);
```

```
  real cyc_jitter1;

  real t_on;

  real t_off;

  real t_on_ideal;

  real t_off_ideal;


                    real jitter_period;

  real prev_jitter=period;

  real diff;


  real max;

  real min;


 //calculating max and min time period (ppm implementation)

  max = period/(1 - ppm/1000000.0);

  min = period/(1 + ppm/1000000.0);


 //cycle to cycle jitter

  cyc_jitter1 = cyc_jitter/1000.0;


forever begin

   do begin

     //picking a time period within the range

     std::randomize(jitter_period) with {

   jitter_period inside {[min : max]};

      };

     //calculating deviation

     diff = jitter_period - prev_jitter;

     diff = (diff > 0) ? diff : -diff;
```

```
        end
          //ensuring cyc-to-cyc jitter condition is not violated
        while (diff > cyc_jitter1);
        prev_jitter = jitter_period;

       //calculating ON time and OFF time
       t_on = d_cyc*jitter_period/100000.0;
       t_off = jitter_period - t_on;
        t_on_ideal = d_cyc*period/100000.0;
       t_off_ideal = period - t_on_ideal;

       //starting clock according to initial value
        if(init_clk_val == 0) begin
          fork
            #(t_off) if(wait_en[clk_name]==0) vif.clk[clk_name] = 1'b1;
            #(t_off_ideal) if(wait_en[clk_name]==0) vif.clk_ideal[clk_name] = 1'b1;
            #(t_off_ideal) if(wait_en[clk_name]==0) clk_dummy[clk_name] = 1'b1;
          join
          `uvm_info("in driver:", $sformatf("clk[%0d]=%0d with var_on_period=%f var_off_period=%f",
clk_name, vif.clk[clk_name], t_on, t_off), UVM_DEBUG)
          fork
           #(t_on) vif.clk[clk_name] = 1'b0;
           #(t_on_ideal) vif.clk_ideal[clk_name] = 1'b0;
           #(t_on_ideal) clk_dummy[clk_name] = 1'b0;
          join
          `uvm_info("in driver: ideal", $sformatf("clk[%0d]=%0d with var_on_period=%f
var_off_period=%f", clk_name, vif.clk_ideal[clk_name], t_on_ideal, t_off_ideal), UVM_DEBUG)
        end
        else begin
          fork
          #(t_on) vif.clk[clk_name] = 1'b0;
            #(t_on_ideal) vif.clk_ideal[clk_name] = 1'b0;
            #(t_on_ideal) clk_dummy[clk_name] = 1'b0;
          join
        `uvm_info("in driver:", $sformatf("clk[%0d]=%0d with var_on_period=%f var_off_period=%f",
clk_name, vif.clk[clk_name], t_on, t_off), UVM_DEBUG)
          fork
          #(t_off) vif.clk[clk_name] = 1'b1;
            #(t_off_ideal) vif.clk_ideal[clk_name] = 1'b1;
            #(t_off_ideal) clk_dummy[clk_name] = 1'b1;
          join
        `uvm_info("in driver:", $sformatf("clk[%0d]=%0d with var_on_period=%f var_off_period=%f",
clk_name, vif.clk[clk_name], t_on, t_off), UVM_DEBUG)
        end
       n[clk_name] = n[clk_name] + 1;
     end
   endtask


       //task to stop the clk
   task ClkDriver::stopClk (input ClkAgentUserClass it);
     for(int j=0; j< it.clk_name.size(); j++) begin
       int walk = j;
       fork
          //wait for the instance of stopping the clk
          wait (n[walk] == it.t_end[walk]) begin
           //if process is running kill it and assign it to null
           if(proc_start_clk[walk] != null) begin
```

```
            proc_start_clk[walk].kill();
            proc_start_clk[walk] = null;
            `uvm_info(get_type_name(),$sformatf("clk[%0d] stopped", walk),UVM_LOW)
            //if clock is stopped, trigger event 'done'
            ->done;
          end
          end
        join_none
      end
    endtask


    //task to pause the clk for specified number of cycles
    task ClkDriver::waitClk (input ClkAgentUserClass it);
      foreach(it.wait_list[j]) begin
        int walk = j;
        fork
          //wait for the instance of gating the clk
          wait (n[it.wait_list[walk]] == it.at_wait[walk]) begin
           //if process is running, suspend the process of clk for the specified number of cycles and resume
it later
           if(proc_start_clk[it.wait_list[walk]] != null) begin
            proc_start_clk[it.wait_list[walk]].suspend();
            wait_en[it.wait_list[walk]] = 1;
            `uvm_info(get_type_name(),$sformatf("clk[%0d] paused", it.wait_list[walk]), UVM_LOW)
            //wait for mentioned number of cycles
            #(it.n_wait[walk]*period_[it.wait_list[walk]]);
            proc_start_clk[it.wait_list[walk]].resume();
            wait_en[it.wait_list[walk]] = 0;
            `uvm_info(get_type_name(), $sformatf("clk[%0d] resumed", it.wait_list[walk]), UVM_LOW)
        end

        end

      join_none

    end

  endtask


      //task to change freq

  task ClkDriver::freqClk (input ClkAgentUserClass it);

    foreach(it.freq_list[k]) begin

      int walk = k;

      fork

        //wait for the instant to change freq

        wait (n[it.freq_list[walk]] == it.change_t_n[walk]) begin

          //kill the process if its running and create a new one

          if(proc_start_clk[it.freq_list[walk]] != null) begin

             proc_start_clk[it.freq_list[walk]].kill();

            proc_start_clk[it.freq_list[walk]] = process::self();

            `uvm_info(get_type_name(),$sformatf("clk[%0d] freq changed", it.freq_list[walk]), UVM_LOW)

            //wait till dull time

            #((it.dull_t[it.freq_list[walk]])/1000.0);

            `uvm_info(get_type_name(),$sformatf("clk[%0d] dull time finished", it.freq_list[walk]),
UVM_LOW)

            //update current time period

            period_[it.freq_list[walk]] = it.new_tp[walk]/1000.0;

            //generate clk with updated freq value

            generate_clk(it.init[it.freq_list[walk]], it.freq_list[walk], it.new_tp[walk]/1000.0,
it.cyc_jitter[it.freq_list[walk]], it.duty_c[it.freq_list[walk]], it.ppm[it.freq_list[walk]]);

          end

        end
      join_none
    end
  endtask



 //driver run phase
task ClkDriver::run_phase(uvm_phase phase);
  ClkAgentUserClass it;
  forever begin
    seq_item_port.get_next_item(it);
   // it.print();
    //initializing values to default values if not specified by user

    //intial value
    if(it.op_type == 0 || it.op_type == 3) begin
      if(it.init.size() < it.clk_name.size()) begin
      int prev = it.init.size();
        it.init = new[it.clk_name.size()] (it.init);
        for(int i=prev ; i< it.clk_name.size(); i++)
        begin
          it.init[i] = `init_def;
            end
    end
      //start time
      if(it.op_type == 0 )
        if(it.phase_shift.size() < it.clk_name.size()) begin
      int prev = it.phase_shift.size();
          it.phase_shift = new[it.clk_name.size()] (it.phase_shift);
          for(int i=prev ; i< it.clk_name.size(); i++)
```

```
     begin
       it.phase_shift[i] = `phase_shift_def;
         end
 end
 end
 //time period
 if( it.op_type == 0 || it.op_type == 2) begin
   if(it.period.size() < it.clk_name.size()) begin
   int prev = it.period.size();
     it.period = new[it.clk_name.size()] (it.period);
     for(int i=prev ; i< it.clk_name.size(); i++)
     begin
       it.period[i] = `period_def;
         end
 end
end
 //duty cycle
 if(it.op_type == 0 || it.op_type == 3) begin
   if(it.duty_c.size() < it.clk_name.size()) begin
   int prev = it.duty_c.size();
     it.duty_c = new[it.clk_name.size()] (it.duty_c);
     for(int i=prev ; i< it.clk_name.size(); i++)
     begin
       it.duty_c[i] = `duty_c_def;
         end
 end
end
 //cycle to cycle jitter
if(it.op_type == 0 || it.op_type == 3) begin
```

```
  if(it.cyc_jitter.size() < it.clk_name.size()) begin
    int prev = it.cyc_jitter.size();
    it.cyc_jitter = new[it.clk_name.size()] (it.cyc_jitter);
    for(int i=prev ; i< it.clk_name.size(); i++)
      begin
        it.cyc_jitter[i] = `cyc_jitter_def;
          end
  end
end
  //ppm value
  if(it.op_type == 0 || it.op_type == 3) begin
  if(it.ppm.size() < it.clk_name.size()) begin
    int prev = it.ppm.size();
    it.ppm = new[it.clk_name.size()] (it.ppm);
    for(int i=prev ; i< it.clk_name.size(); i++)
      begin
        it.ppm[i] = `ppm_def;
          end
  end
  end
  //stop time
  if(it.op_type == 1) begin
  if(it.t_end.size() < it.clk_name.size()) begin
    int prev = it.t_end.size();
    it.t_end = new[it.clk_name.size()] (it.t_end);
    for(int i=prev ; i< it.clk_name.size(); i++)
      begin
        it.t_end[i] = `t_end_def;
          end
```

```
end
end
//change frequency instance
if(it.op_type == 3) begin
if(it.change_t_n.size() < it.freq_list.size()) begin
 int prev = it.change_t_n.size();
 it.change_t_n = new[it.freq_list.size()] (it.change_t_n);
 for(int i=prev ; i< it.freq_list.size(); i++)
   begin
     it.change_t_n[i] = n[it.freq_list[i]];
       end
end
end
//dull time
if(it.op_type == 3) begin
if(it.dull_t.size() < it.freq_list.size()) begin
 int prev = it.dull_t.size();
 it.dull_t = new[it.freq_list.size()] (it.dull_t);
 for(int i=prev ; i< it.freq_list.size(); i++)
   begin
     it.dull_t[i] = `dull_t_def;
       end
end
end
//new time period
if(it.op_type == 3) begin
if(it.new_tp.size() < it.freq_list.size()) begin
 int prev = it.new_tp.size();
 it.new_tp = new[it.freq_list.size()] (it.new_tp);
```

```
  for(int i=prev ; i< it.freq_list.size(); i++)
    begin
      it.new_tp[i] = `new_tp_def;
        end
end
end
//no of cycles to gate clock
if(it.op_type == 2) begin
if(it.n_wait.size() < it.wait_list.size()) begin
        int prev = it.n_wait.size();
  it.n_wait = new[it.wait_list.size()] (it.n_wait);
  for(int i=prev ; i< it.wait_list.size(); i++)
    begin
      it.n_wait[i] = `n_wait_def;
        end
end
end
//instant to gate clk
if(it.op_type == 2) begin
if(it.at_wait.size() < it.wait_list.size()) begin
        int prev = it.at_wait.size();
  it.at_wait = new[it.wait_list.size()] (it.at_wait);
  for(int i=prev ; i< it.wait_list.size(); i++)
    begin
      it.at_wait[i] = n[it.wait_list[i]];
        end
end
end
```

```
//initially create processes, current period and number of cycles for each clk
if(first == 0) begin
  proc_start_clk = new [it.clk_name.size()];
  n = new[it.clk_name.size()];
  period_ = new[it.clk_name.size()];
  clk_dummy = new[it.clk_name.size()];
  wait_en = new[it.clk_name.size()];
end

//call task according to the operation to be performed
case(it.op_type)
  0 : startClk (it);
  1 : stopClk (it);
  2 : waitClk (it);
  3 : freqClk (it);
  default : `uvm_error(get_type_name,"operation not found")
endcase

//increment to track number of sequences driven
first = first + 1;
  //if last sequence is driven, wait for all clocks to stop
  if(first == it.op_type.num())
      begin
        //wait till all clocks are stopped
                        repeat(it.clk_name.size()) begin
                                    @(done);
                                                end
                                    end
seq_item_port.item_done();
```

```
    end
  endtask: run_phase
```

Clock Agent

```
//`include "clk_driver.sv"
//`include "clk_monitor.sv"
 `include "uvm_macros.svh"
import uvm_pkg::*;
//`include "clk_agent_pkg.sv"
//agent class where operations are done acc to ACTIVE/ PASSIVE
class ClkAgent extends uvm_agent;
  `uvm_component_param_utils(ClkAgent)

  uvm_sequencer #(ClkAgentUserClass) sqcr;
  ClkDriver drv;
  //ClkMonitor mon;

  //uvm_analysis_port #(ClkAgentUserClass) aport;

  //class constructor
  function new(string name = "ClkAgent", uvm_component parent = null);
    super.new(name, parent);
  endfunction

  extern virtual function void build_phase  (uvm_phase phase);
  extern virtual function void connect_phase(uvm_phase phase);

endclass: ClkAgent
```

```
 //build phase - creating sequencer, monitor and driver acc to state of agent
 function void ClkAgent::build_phase(uvm_phase phase);
   //aport = new("aport", this);
   //uvm_config_db#(int)::get(this,"", "is_active", is_active);
   //if agent is active, creating sequencer, monitor and driver
            //if(is_active == UVM_ACTIVE) begin
   sqcr = uvm_sequencer #(ClkAgentUserClass)::type_id::create("sequencer", this);
            drv = ClkDriver::type_id::create(   "driver", this);
   //end
   //if agent is passive,create only monitor
   // mon = ClkMonitor::type_id::create("mon", this);
   `uvm_info(get_full_name(), "Agent Build Stage Complete", UVM_LOW)
 endfunction: build_phase



 function void ClkAgent::connect_phase(uvm_phase phase);
   //if agent is active, connect driver and sequencer
   //if(is_active == UVM_ACTIVE) begin
                        drv.seq_item_port.connect(sqcr.seq_item_export);
      uvm_report_info(get_full_name(), "connect_phase, Connected driver to sequencer");
    //end
 endfunction: connect_phase
```

Memory item

```
`include "uvm_macros.svh"
import uvm_pkg::*;
```

```
// This is the base transaction object that will be used
// in the environment to initiate new transactions and
// capture transactions at DUT interface
class reg_item extends uvm_sequence_item;
  rand bit [`ADDR_WIDTH-1:0]    addr;
  rand bit [`DATA_WIDTH-1:0]    wdata;
  rand bit                                              wr;
  bit [`DATA_WIDTH-1:0]                 rdata;

  // Use utility macros to implement standard functions
  // like print, copy, clone, etc
  `uvm_object_utils_begin(reg_item)
            `uvm_field_int (addr, UVM_DEFAULT)
            `uvm_field_int (wdata, UVM_DEFAULT)
            `uvm_field_int (rdata, UVM_DEFAULT)
            `uvm_field_int (wr, UVM_DEFAULT)
  `uvm_object_utils_end

  virtual function string convert2str();
    return $sformatf("addr=0x%0h wr=0x%0h wdata=0x%0h rdata=0x%0h", addr, wr, wdata, rdata);
  endfunction

  function new(string name = "reg_item");
    super.new(name);
  endfunction
endclass
```

Memory Sequence

```
//import mem_pkg::*;
```

```
//`include "clk_agent_pkg.sv"
`include "uvm_macros.svh"
import uvm_pkg::*;
//`include "mem_item.sv"
class gen_item_seq extends uvm_sequence#(reg_item);
  `uvm_object_utils(gen_item_seq)
  function new(string name="gen_item_seq");
    super.new(name);
  endfunction

  rand int num;   // Config total number of items to be sent

  constraint c1 { soft num inside {[2:5]}; }

  virtual task body();
    for (int i = 0; i < num; i ++) begin
            reg_item m_item = reg_item::type_id::create("m_item");
            start_item(m_item);
            m_item.randomize();
      `uvm_info("SEQ", $sformatf("Generate new item: %s", m_item.convert2str()), UVM_LOW)
            finish_item(m_item);
    end
    `uvm_info("SEQ", $sformatf("Done generation of %0d items", num), UVM_LOW)
  endtask
endclass

//`include "clk_agent_pkg.sv"
`include "uvm_macros.svh"
import uvm_pkg::*;
```

```
class driver extends uvm_driver #(reg_item);
  `uvm_component_utils(driver)
  function new(string name = "driver", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  virtual ClkIf vif;

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    if (!uvm_config_db#(virtual ClkIf)::get(this, "", "vif", vif))
      `uvm_fatal("DRV", "Could not get vif")
  endfunction

  virtual task run_phase(uvm_phase phase);
    super.run_phase(phase);
    forever begin
      reg_item m_item;
      `uvm_info("DRV", $sformatf("Wait for item from sequencer"), UVM_LOW)
      seq_item_port.get_next_item(m_item);
      drive_item(m_item);
      seq_item_port.item_done();
    end
  endtask

  virtual task drive_item(reg_item m_item);
      vif.sel <= 1;
      vif.addr        <= m_item.addr;
      vif.wr          <= m_item.wr;
      vif.wdata <= m_item.wdata;
    @(posedge vif.clk[0]);
      while (!vif.ready)  begin
        `uvm_info("DRV", "Wait until ready is high", UVM_LOW)
        @(posedge vif.clk[0]);
      end

      vif.sel <= 0;
  endtask
endclass
```

Memory Agent

```
//import mem_pkg::*;
//`include "clk_agent_pkg.sv"
`include "uvm_macros.svh"
import uvm_pkg::*;
//`include "mem_item.sv"

//`include "mem_driver.sv"
class agent extends uvm_agent;
  `uvm_component_utils(agent)
function new(string name="agent", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  driver              d0;                  // Driver handle
  //monitor                    m0;                  // Monitor handle
  uvm_sequencer #(reg_item)    s0;                  // Sequencer Handle
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    s0 = uvm_sequencer#(reg_item)::type_id::create("s0", this);
    d0 = driver::type_id::create("d0", this);
   // m0 = monitor::type_id::create("m0", this);
  endfunction

  virtual function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    d0.seq_item_port.connect(s0.seq_item_export);
  endfunction

endclass
```

Environment Class

```
//import mem_pkg::*;
`include "uvm_macros.svh"
import uvm_pkg::*;
//import clk_agent_pkg::*;
//`include "clk_agent_pkg.sv"
//`include "mem_agent.sv"
//`include "clk_agent.sv"
class env extends uvm_env;
  `uvm_component_utils(env)
  function new(string name="env", uvm_component parent=null);
    super.new(name, parent);
  endfunction

  agent           a0;                // Agent handle
```

```
 //scoreboard   sb0;                    // Scoreboard handle
  ClkAgent    c_agent;
 virtual function void build_phase(uvm_phase phase);
   super.build_phase(phase);
   a0 = agent::type_id::create("a0", this);
   c_agent=    ClkAgent::type_id::create("c_agent", this);
  // uvm_config_db#(int)::set(this, "c_agent", "is_active", UVM_ACTIVE);
   //  sb0 = scoreboard::type_id::create("sb0", this);
 endfunction

 //virtual function void connect_phase(uvm_phase phase);
 //  super.connect_phase(phase);
 //  a0.m0.mon_analysis_port.connect(sb0.m_analysis_imp);
//  endfunction
Endclass
```

Test Class

```
//`include"mem_seq.sv"
//`include"mem_env.sv"
//`include"clk_sequence.sv"
`include "uvm_macros.svh"
import uvm_pkg::*;
//import mem_pkg::*;
import clk_agent_pkg::*;
class test extends uvm_test;
 `uvm_component_utils(test)
 function new(string name = "test", uvm_component parent=null);
   super.new(name, parent);
 endfunction
```

```
  gen_item_seq seq;
   uvm_table_printer printer;
    ClkStartSequence seq1;
    ClkStopSequence seq2;
    ClkFreqSequence seq3;
    ClkWaitSequence seq4;
  env e0;
// virtual ClkIf vif;

  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    e0 = env::type_id::create("e0", this);
     printer = new();
    printer.knobs.depth = 5;
    //if (!uvm_config_db#(virtual reg_if)::get(this, "", "reg_vif", vif))
     // `uvm_fatal("TEST", "Did not get vif")

     // uvm_config_db#(virtual reg_if)::set(this, "e0.a0.*", "reg_vif", vif);
  endfunction

  virtual task run_phase(uvm_phase phase);

    //creating sequences
    seq1 = ClkStartSequence::type_id::create("seq1");
    seq2 = ClkStopSequence::type_id::create("seq2");
    seq3 = ClkFreqSequence::type_id::create("seq3");
    seq4 = ClkWaitSequence::type_id::create("seq4");
    seq = gen_item_seq::type_id::create("seq");
    phase.raise_objection(this);
    run_sequences(seq1, seq2, seq3, seq4);
   // apply_reset();
    seq.randomize() with {num inside {[20:30]}; };
    seq.start(e0.a0.s0);
    #200;
    phase.drop_objection(this);
  endtask

  virtual function void end_of_elaboration_phase(uvm_phase phase);
    super.end_of_elaboration_phase(phase);
    `uvm_info(get_type_name(), $sformatf("Printing Test Topology: \n%s", this.sprint(printer)),
UVM_LOW)
  endfunction

  virtual task run_sequences(input ClkStartSequence seq1, input   ClkStopSequence seq2, input
ClkFreqSequence seq3, input  ClkWaitSequence seq4);
    fork
      seq1.start(e0.c_agent.sqcr);
    #20 seq3.start(e0.c_agent.sqcr);
    #10 seq4.start(e0.c_agent.sqcr);
   // seq2.start(e0.c_agent.sqcr);
    join
  endtask

  //virtual task apply_reset();
   // vif.rstn <= 0;
   // repeat(5) @ (posedge vif.clk[0]);
   // vif.rstn <= 1;
   // repeat(10) @ (posedge vif.clk[0]);
  //endtask

Endclass

Package
package clk_agent_pkg;
`define ADDR_WIDTH 8
`define DATA_WIDTH 16
`define DEPTH 256
`define ppm_def 100
`define init_def 0
`define phase_shift_def 0
`define period_def 1000
`define duty_c_def 50000
`define cyc_jitter_def 10
`define t_end_def -1
`define dull_t_def 100
`define new_tp_def 1000
`define n_wait_def 1
`include "mem_item.sv"
`include "mem_seq.sv"
`include "mem_driver.sv"
`include "mem_agent.sv"
`include "clk_item.sv"
`include "clk_sequence.sv"
`include "clk_driver.sv"
`include "clk_agent.sv"
//`include "def_file.sv"
`include "mem_env.sv"
```

```
//`include "mem_test.sv"
Endpackage


Testbench Top
// Code your testbench here
// or browse Examples
`include "uvm_macros.svh"
import uvm_pkg::*;
//import mem_pkg::*;
//`include "def_file.sv"
`include "clk_agent_pkg.sv"
//import mem_pkg::*;
`include "mem_test.sv"
interface ClkIf #(parameter N=1)();
  logic clk [N-1:0]; //N-1 clock signals
  logic clk_ideal [N-1 : 0];
  logic rstn;
  logic [7:0] addr;
  logic [15:0] wdata;
  logic [15:0] rdata;
  logic              wr;
  logic              sel;
  logic              ready;
endinterface
module tb;
 // reg clk;

//  always #10 clk =~ clk;
//  reg_if _if (clk);
```

```
  ClkIf vif();
 reg_ctrl u0 (.dif(vif));

 //test t0;

 initial begin
   //clk <= 0;
    uvm_config_db #(virtual ClkIf)::set(uvm_root::get(),"*","vif",vif);
   run_test("test");
 end

 // System tasks to dump VCD waveform file
 initial begin
   $dumpvars;
   $dumpfile ("dump.vcd");
 end
endmodule
```

Output

```
[2021-06-12 03:00:02 EDT] xrun -Q -unbuffered '-timescale' '1ns/1ns' '-sysv' '-access'
'+rw' -uvmnocdnsextra -uvmhome $UVM_HOME $UVM_HOME/src/uvm_macros.svh design.sv
testbench.sv
TOOL:  xrun   20.09-s003: Started on Jun 12, 2021 at 03:00:02 EDT
xrun: 20.09-s003: (c) Copyright 1995-2020 Cadence Design Systems, Inc.
        m_item.randomize();
                      |
xmvlog: *W,FUNTSK (mem_seq.sv,20|20): function called as a task without void'().
(`include file: mem_seq.sv line 20, `include file: clk_agent_pkg.sv line 16, file:
testbench.sv line 7)
           std::randomize(jitter_period) with {
                          |
xmvlog: *W,FUNTSK (clk_driver.sv,105|25): function called as a task without void'().
(`include file: clk_driver.sv line 105, `include file: clk_agent_pkg.sv line 21, file:
testbench.sv line 7)
package clk_agent_pkg;
                    |
xmvlog: *W,TSNSPK (clk_agent_pkg.sv,1|20): `timescale is not specified for the
package.  The default timescale of 1ns/1ns will be assumed for this package.
(`include file: clk_agent_pkg.sv line 26, file: testbench.sv line 7)
    seq.randomize() with {num inside {[20:30]}; };
                |
xmvlog: *W,FUNTSK (mem_test.sv,44|16): function called as a task without void'().
(`include file: mem_test.sv line 44, file: testbench.sv line 9)
        Top level design units:
                uvm_pkg
                clk_agent_pkg
                $unit_0x67f934e9
                tb
xmelab: *W,DSEMEL: This SystemVerilog design will be simulated as per IEEE 1800-2009
SystemVerilog simulation semantics. Use -disable_sem2009 option for turning off SV
2009 simulation semantics.
Loading snapshot worklib.tb:sv .................... Done
SVSEED default: 1
xmsim: *W,RNDXCELON: A newer version of the SystemVerilog constraint solver is being
used which has better support for array-solving, new solve-order mechanism, and seed
stability enhancements..
xmsim: *W,DSEM2009: This SystemVerilog design is simulated as per IEEE 1800-2009
SystemVerilog simulation semantics. Use -disable_sem2009 option for turning off SV
2009 simulation semantics.
xcelium> source /xcelium20.09/tools/xcelium/files/xmsimrc
xcelium> source /xcelium20.09/tools//methodology/UVM/CDNS-1.2/sv/files/tcl/uvm_sim.tcl
xcelium> run
UVM_INFO /xcelium20.09/tools//methodology/UVM/CDNS-1.2/sv/src/base/uvm_root.svh(412) @
0: reporter [UVM/RELNOTES]
----------------------------------------------------------------
CDNS-UVM-1.2 (20.09-s003)
(C) 2007-2014 Mentor Graphics Corporation
(C) 2007-2014 Cadence Design Systems, Inc.
(C) 2006-2014 Synopsys, Inc.
(C) 2011-2013 Cypress Semiconductor Corp.
(C) 2013-2014 NVIDIA Corporation
----------------------------------------------------------------

  ***********       IMPORTANT RELEASE NOTES         ************

  You are using a version of the UVM library that has been compiled
  with `UVM_NO_DEPRECATED undefined.
  See http://www.eda.org/svdb/view.php?id=3313 for more details.

  You are using a version of the UVM library that has been compiled
  with `UVM_OBJECT_DO_NOT_NEED_CONSTRUCTOR undefined.
  See http://www.eda.org/svdb/view.php?id=3770 for more details.
```

```
      (Specify +UVM_NO_RELNOTES to turn off this notice)

UVM_INFO @ 0: reporter [RNTST] Running test test...
xmsim: *W,PRPASZ: Packed array at
"worklib.uvm_pkg::uvm_string_to_bits.uvm_string_to_bits" of 115200 bits exceeds limit
of 4096 - not probed
 Use 'probe -create -packed 115200
worklib.uvm_pkg::uvm_string_to_bits.uvm_string_to_bits' or 'setenv SHM_PACKED_LIMIT
115200' to adjust limit.
xmsim: *W,PRPASZ: Packed array at "worklib.uvm_pkg::uvm_bits_to_string.str" of 115200
bits exceeds limit of 4096 - not probed
 Use 'probe -create -packed 115200 worklib.uvm_pkg::uvm_bits_to_string.str' or 'setenv
SHM_PACKED_LIMIT 115200' to adjust limit.
UVM_INFO clk_agent.sv(37) @ 0: uvm_test_top.e0.c_agent [uvm_test_top.e0.c_agent] Agent
Build Stage Complete
UVM_INFO @ 0: uvm_test_top.e0.c_agent [uvm_test_top.e0.c_agent] connect_phase,
Connected driver to sequencer
UVM_INFO mem_test.sv(52) @ 0: uvm_test_top [test] Printing Test Topology:
--------------------------------------------------------------
Name                        Type                     Size  Value
--------------------------------------------------------------
uvm_test_top                test                     -     @1883
  e0                        env                      -     @1948
    a0                      agent                    -     @1986
      d0                    driver                   -     @2707
        rsp_port            uvm_analysis_port        -     @2807
        seq_item_port       uvm_seq_item_pull_port   -     @2757
      s0                    uvm_sequencer            -     @2048
        rsp_export          uvm_analysis_export      -     @2112
        seq_item_export     uvm_seq_item_pull_imp    -     @2674
        arbitration_queue   array                    0     -
        lock_queue          array                    0     -
        num_last_reqs       integral                 32    'd1
        num_last_rsps       integral                 32    'd1
    c_agent                 uvm_agent                -     @2017
      driver                ClkDriver                -     @3517
        rsp_port            uvm_analysis_port        -     @3624
        seq_item_port       uvm_seq_item_pull_port   -     @3574
      sequencer             uvm_sequencer            -     @2862
        rsp_export          uvm_analysis_export      -     @2922
        seq_item_export     uvm_seq_item_pull_imp    -     @3484
        arbitration_queue   array                    0     -
        lock_queue          array                    0     -
        num_last_reqs       integral                 32    'd1
        num_last_rsps       integral                 32    'd1
--------------------------------------------------------------

UVM_INFO mem_driver.sv(22) @ 0: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO clk_sequence.sv(41) @ 0: uvm_test_top.e0.c_agent.sequencer@@seq1
[ClkStartSequence] starting item in start clk
UVM_INFO clk_sequence.sv(102) @ 10: uvm_test_top.e0.c_agent.sequencer@@seq4
[ClkWaitSequence] starting item in wait clk
UVM_INFO clk_sequence.sv(130) @ 20: uvm_test_top.e0.c_agent.sequencer@@seq3
[ClkFreqSequence] starting item in change freq clk
UVM_INFO clk_driver.sv(210) @ 20: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[1]
freq changed
UVM_INFO clk_driver.sv(213) @ 20: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[1]
dull time finished
UVM_INFO mem_seq.sv(21) @ 20: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x84 wr=0x1 wdata=0x66ea rdata=0x0
UVM_INFO mem_driver.sv(22) @ 22: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 22: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
```

```
addr=0xb2 wr=0x0 wdata=0x1311 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 24: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 24: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0xc wr=0x0 wdata=0x3d7 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 26: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 26: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x94 wr=0x1 wdata=0xa0c4 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 28: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 28: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x49 wr=0x1 wdata=0xf5ab rdata=0x0
UVM_INFO mem_driver.sv(22) @ 30: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 30: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x2c wr=0x0 wdata=0x778c rdata=0x0
UVM_INFO clk_driver.sv(210) @ 31: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[0]
freq changed
UVM_INFO clk_driver.sv(213) @ 32: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[0]
dull time finished
UVM_INFO mem_driver.sv(22) @ 35: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 35: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x3c wr=0x0 wdata=0xb167 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 41: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 41: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x79 wr=0x1 wdata=0xa33b rdata=0x0
UVM_INFO clk_driver.sv(210) @ 44: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[1]
freq changed
UVM_INFO clk_driver.sv(213) @ 44: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[1]
dull time finished
UVM_INFO mem_driver.sv(22) @ 46: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 46: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0xe3 wr=0x1 wdata=0xcddc rdata=0x0
UVM_INFO mem_driver.sv(22) @ 52: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO clk_driver.sv(187) @ 52: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[1]
paused
UVM_INFO mem_seq.sv(21) @ 52: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x7b wr=0x0 wdata=0xa4a4 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 59: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 59: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x40 wr=0x1 wdata=0xb438 rdata=0x0
UVM_INFO clk_driver.sv(187) @ 62: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[0]
paused
UVM_INFO clk_driver.sv(192) @ 80: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[1]
resumed
UVM_INFO clk_driver.sv(192) @ 82: uvm_test_top.e0.c_agent.driver [ClkDriver] clk[0]
resumed
UVM_INFO mem_driver.sv(22) @ 88: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 88: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x32 wr=0x1 wdata=0xfaf3 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 94: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 94: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x52 wr=0x1 wdata=0x6ea8 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 99: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 99: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x9e wr=0x0 wdata=0x26fd rdata=0x0
UVM_INFO mem_driver.sv(22) @ 105: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
```

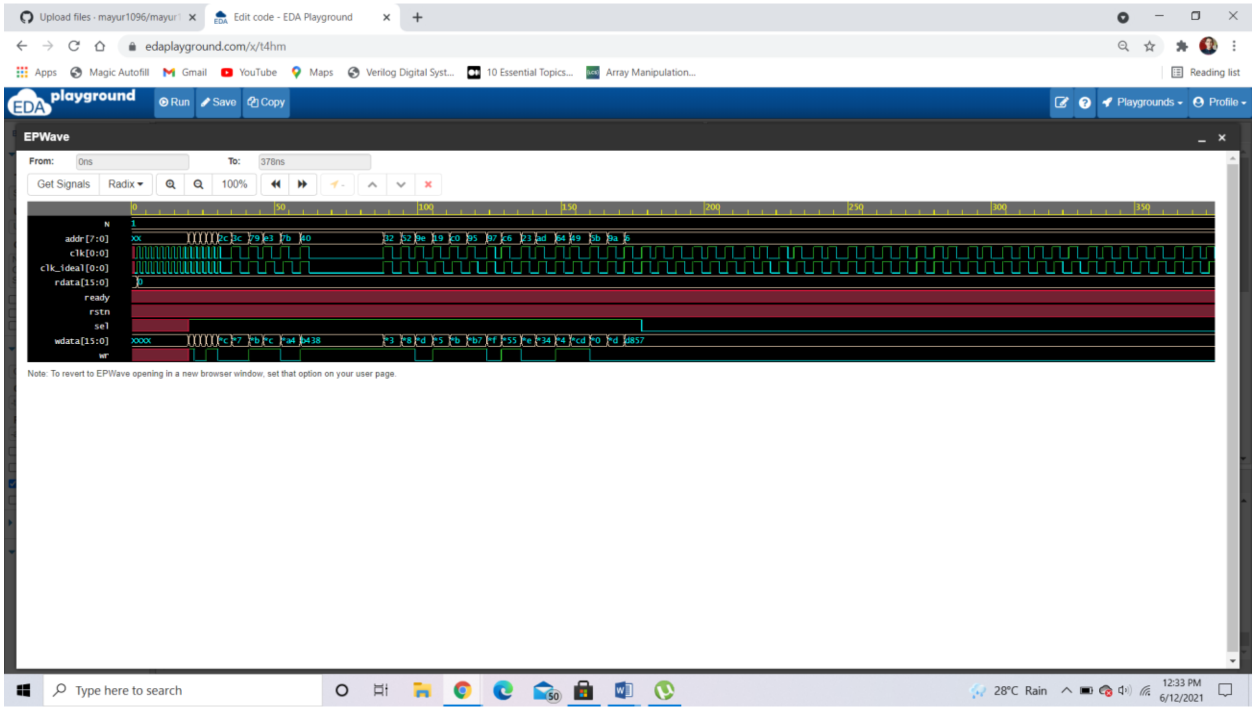
```
sequencer
UVM_INFO mem_seq.sv(21) @ 105: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x19 wr=0x1 wdata=0x7fa5 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 111: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 111: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0xc0 wr=0x1 wdata=0x9c1b rdata=0x0
UVM_INFO mem_driver.sv(22) @ 117: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 117: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x95 wr=0x1 wdata=0x64b7 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 124: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 124: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x97 wr=0x0 wdata=0x661f rdata=0x0
UVM_INFO mem_driver.sv(22) @ 129: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 129: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0xc6 wr=0x1 wdata=0xa055 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 136: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 136: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x23 wr=0x0 wdata=0x5de rdata=0x0
UVM_INFO mem_driver.sv(22) @ 141: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 141: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0xad wr=0x0 wdata=0xa434 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 148: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 148: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x64 wr=0x1 wdata=0xfa84 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 153: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 153: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x49 wr=0x1 wdata=0x7dcd rdata=0x0
UVM_INFO mem_driver.sv(22) @ 160: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 160: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x5b wr=0x0 wdata=0xb910 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 166: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 166: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x9a wr=0x0 wdata=0xac4d rdata=0x0
UVM_INFO mem_driver.sv(22) @ 172: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(21) @ 172: uvm_test_top.e0.a0.s0@@seq [SEQ] Generate new item:
addr=0x6 wr=0x0 wdata=0xd857 rdata=0x0
UVM_INFO mem_driver.sv(22) @ 178: uvm_test_top.e0.a0.d0 [DRV] Wait for item from
sequencer
UVM_INFO mem_seq.sv(24) @ 178: uvm_test_top.e0.a0.s0@@seq [SEQ] Done generation of 26
items
UVM_INFO /xcelium20.09/tools//methodology/UVM/CDNS-
1.2/sv/src/base/uvm_objection.svh(1271) @ 378: reporter [TEST_DONE] 'run' phase is
ready to proceed to the 'extract' phase
UVM_INFO /xcelium20.09/tools//methodology/UVM/CDNS-
1.2/sv/src/base/uvm_report_server.svh(847) @ 378: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :   73
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[ClkDriver]    10
[ClkFreqSequence]     1
```

```
[ClkStartSequence]     1
[ClkWaitSequence]     1
[DRV]     27
[RNTST]     1
[SEQ]     27
[TEST_DONE]     1
[UVM/RELNOTES]     1
[test]     1
[uvm_test_top.e0.c_agent]     2

Simulation complete via $finish(1) at time 378 NS + 58
/xcelium20.09/tools/methodology/UVM/CDNS-1.2/sv/src/base/uvm_root.svh:543     $finish;
xcelium> exit
TOOL:  xrun   20.09-s003: Exiting on Jun 12, 2021 at 03:00:11 EDT  (total: 00:00:09)
Finding VCD file...
./dump.vcd
[2021-06-12 03:00:11 EDT] Opening EPWave...
Done
```

waveform



Note: To revert to EPWave opening in a new browser window, set that option on your user page.