Automating sequence creation from a microarchitecture specification

Subramoni Parameswaran, Ravi Ram

2100 Logic drive, San Jose, CA 95124

Objectives

-Create stability and flexibility in the stimulus generation sequence classes

-Reduce footprint of testbench change required

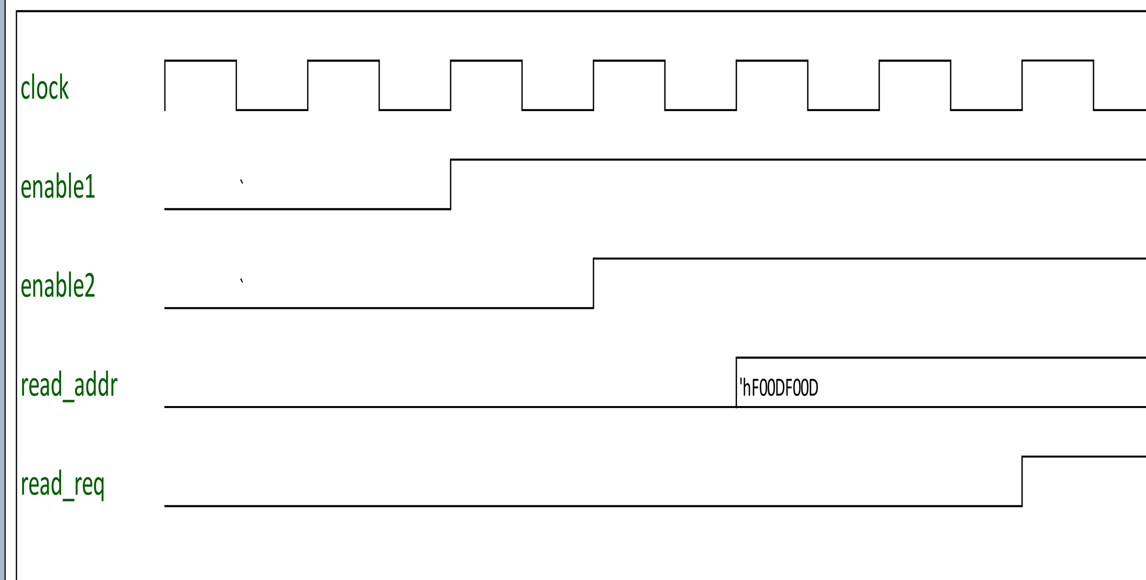
-Automate sequence/test creation

-Improve ability to absorb modified verification requirements

Microarchitecture Specification

-Read request signals
- *clock*
- *enable1*
- *enable2*
- *read_addr*
- *read_req*

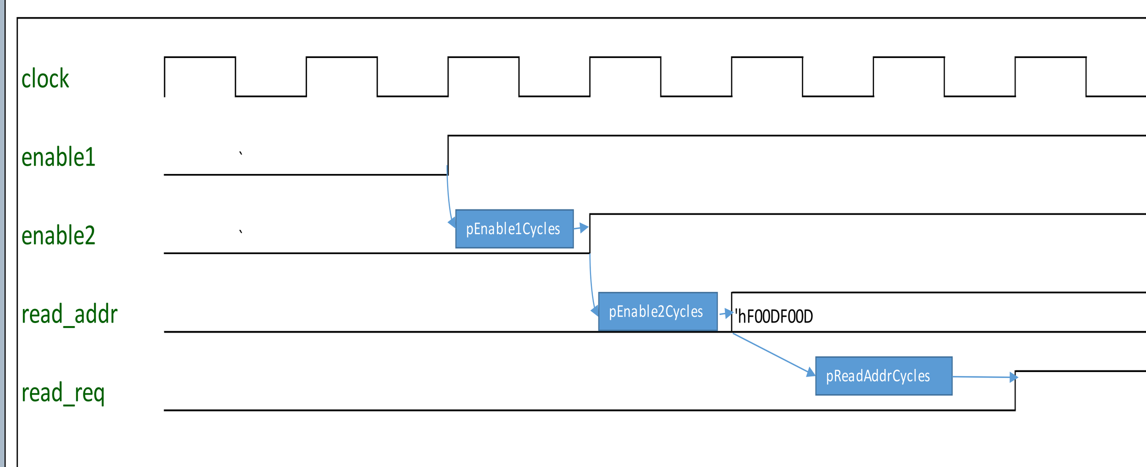
-Microarchitecture Assumptions/Specification
- *enable1* before/at *enable2*
- *enable2* before/at *read_addr*
- *read_addr* before/at *read_req*



Randomization in the sequences –(1)

Introduce random delays between signal transitions in the sequence

Use (min, max) parameter ranges to constrain delays




```
class read_seq extends uvm_sequence #(read_txfer);
…………………
rand uint pEnable1Cycles;  // Specific number of cycles where only enable1 will be high
uint pEnable1CyclesMin = 0, pEnable1CyclesMax = 13;     // Range of cycles where only enable1 can be high
…………………
constraint cEnable1Cycles {
pEnable1Cycles >= pEnable1CyclesMin;
pEnable1Cycles <= pEnable1CyclesMax;
}

virtual task body():
repeat(pEnable1Cycles)
`uvm_do_with(req, {enable1 == 1'b1;});
repeat(pEnable2Cycles)
`uvm_do_with(req, {enable1 == 1'b1; enable2 == 1'b1;});
repeat(pReadAddrCycles)
`uvm_do_with(req, {enable1 == 1'b1; enable2 == 1'b1; read_addr == 'hF00DF00D;});
`uvm_do_with(req, {enable1 == 1'b1; enable2 == 1'b1; read_addr == 'hF00DF00D; read == 1'b1;});
endtask
…………………
endclass
```

Randomization driven by test –(2)

Move sequence configuration into a sequence configuration class that can be modified by the test writer at run time

Same sequence can be reused by multiple directed tests, reducing changes required in underlying sequences

Use the factory to override the sequence configuration class type in some or all tests if desired

```
class read_seq_cfg extends uvm_object;
rand uint pEnable1Cycles;                           // Specific number of cycles where only enable1 will be high
uint pEnable1CyclesMin, pEnable1CyclesMax;     // Range of cycles where only enable1 can be high
……
function new(string name="read_seq_cfg",
  int unsigned pEnable1CyclesMin = 0,           // Set default minimum, override at run time if needed
  int unsigned pEnable1CyclesMax = 13,          // Set default maximum, override at run time if needed
  ….
  );
  this.pEnable1CyclesMin = pEnable1CyclesMin;
  this.pEnable1CyclesMax = pEnable1CyclesMax;
endfunction

constraint cEnable1Cycles {
  pEnable1Cycles >= pEnable1CyclesMin;
  pEnable1Cycles <= pEnable1CyclesMax;
}
endclass
```

```
class read_long_enables_test extends uvm_test;
rand read_seq_cfg m_read_seq_cfg;
virtual function void build_phase(uvm_phase phase):
…..
 m_read_seq_cfg = read_seq_cfg::type_id::create("read_seq_cfg");
 if(!m_read_seq_cfg.randomize()) `uvm_error(get_type_name(), "Unable to randomize read_seq_cfg");
 read_seq_cfg.pEnable1CyclesMin = 12;
 read_seq_cfg.pEnable1CyclesMax = 13;
 uvm_config_object::set(this, "*", "read_seq_cfg", m_read_seq_cfg);
endfunction
endclass
```

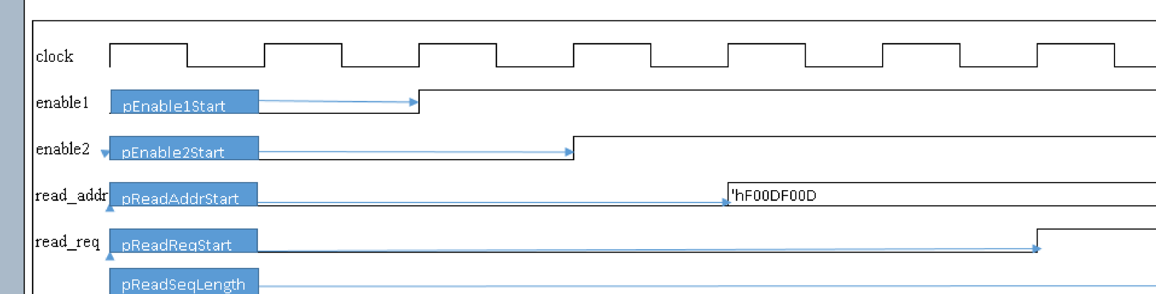
```
class read_seq extends uvm_sequence #(read_txfer);
…………………
virtual task body():
 parent = get_sequencer();                    // Get the uvm_component to eventually get linked config object
 if (parent == null) `uvm_fatal("nullParent", …);
 if (m_read_seq_cfg == null) begin
  if (!uvm_config_db#(read_seq_cfg)::get(parent, "", "read_seq_cfg",
     m_read_seq_cfg))   // Get the configuration for the sequence
   `uvm_fatal("nullReadReqSeqCfg", ….);
 end
 repeat(m_read_seq_cfg.pEnable1Cycles)    `uvm_do_with(req, {enable1 == 1'b1;});
 repeat(m_read_seq_cfg.pEnable2Cycles)    `uvm_do_with(req, {enable1 == 1'b1; enable2 == 1'b1;});
 repeat(m_read_seq_cfg.pReadAddrCycles) `uvm_do_with(req, {enable1 == 1'b1; enable2 == 1'b1;

 read_addr == 'hF00DF00D;});
 `uvm_do_with(req, {enable1 == 1'b1; enable2 == 1'b1; read_addr == 'hF00DF00D; read == 1'b1;});
endtask
……
endclass
```

Ultimate sequence flexibility-(3)

Redefine the sequence configuration parameters as an executable microarchitecture specification for the entire sequence

Define and use a signal generator (sig_gen) class that can be shared by all sequences to automate the creation of sequences



Signal parameter	Minimum bound from start of sequence	Maximum bound from start of sequence
pEnable1Start	0	2
pEnable2Start	0	3
pReadAddrStart	0	4
pReadReqStart	5	6
pReadSeqLength	7	9

```
class sig_gen #(type T=bit);
…….
 function new(string       name,
              int unsigned initialLength,
              T           initialVal='d0,
              int unsigned finalLength,
              T           finalVal=~initialVal);
    this.name          = name;
    this.initialLength = initialLength;
    this.initialVal    = initialVal;
    this.finalLength   = finalLength;
    this.finalVal      = finalVal;
 endfunction // new

 function T getVal();
    T returnVal;
  if (count < initialLength) return initialVal;
  else if ((count >initialLength) && (count < finalLength)) return finalVal;
  count++;
 endfunction
endclass
```

```
class read_seq extends uvm_sequence #(read_txfer);
…………………
virtual task body():
 // local variables (l_ variables) to compute the cycle-by-cycle values of the signals that we want to drive
 sig_gen#(bit) l_enable1, l_enable2, l_read_req;

 l_enable1 = new("l_enable1",
 .initialLength(m_read_seq_cfg.pEnable1Start),
                .initialValue(1'b0), .finalValue(1'b1),
                .finalLength(m_read_seq_cfg.pReadSeqLength -
 m_read_seq_cfg.pEnable1Start));
 …. // similar definitions for l_read_req and l_read_addr

 for (int i = 0; i < m_read_seq_cfg.pReadSeqLength; i++) begin
   `uvm_do_with(req, {enable1 == l_enable1.getVal();
                      enable2 == l_enable2.getVal();
                      read_addr == l_read_addr.getVal();
                      read_req == l_read_req.getVal();});
 end
endtask
……
endclass
```

Conclusion

Final approach used successfully as part of shared VIP

Stable and flexible sequences can be created using UVM configuration data base

Test writer can create targeted random tests without changing underlying sequences

Sequences/signals can share code using signal generator class

Executable microarchitecture specification and shared signal generator class can be used to generate sequences

www.PosterPresentations.com