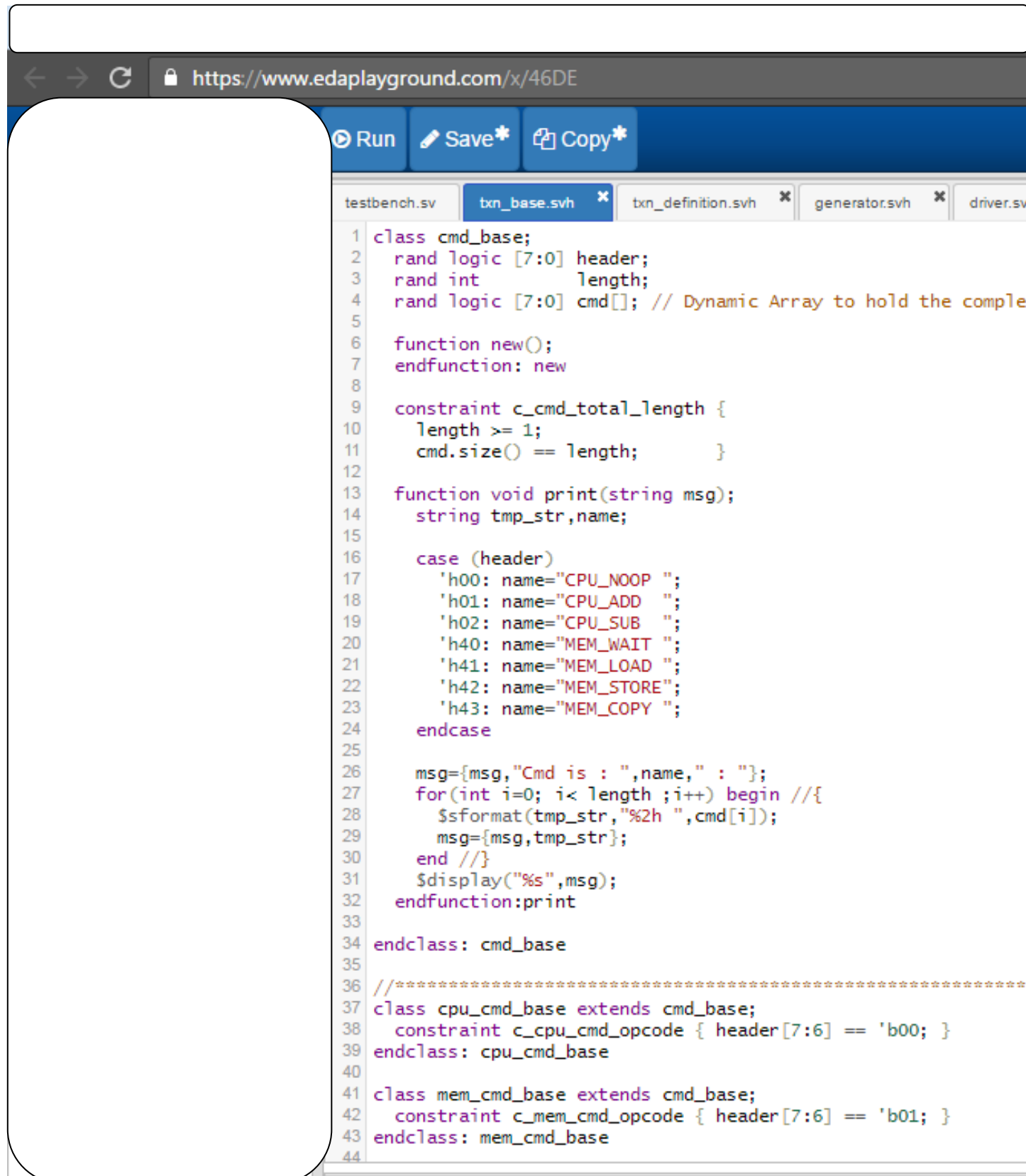
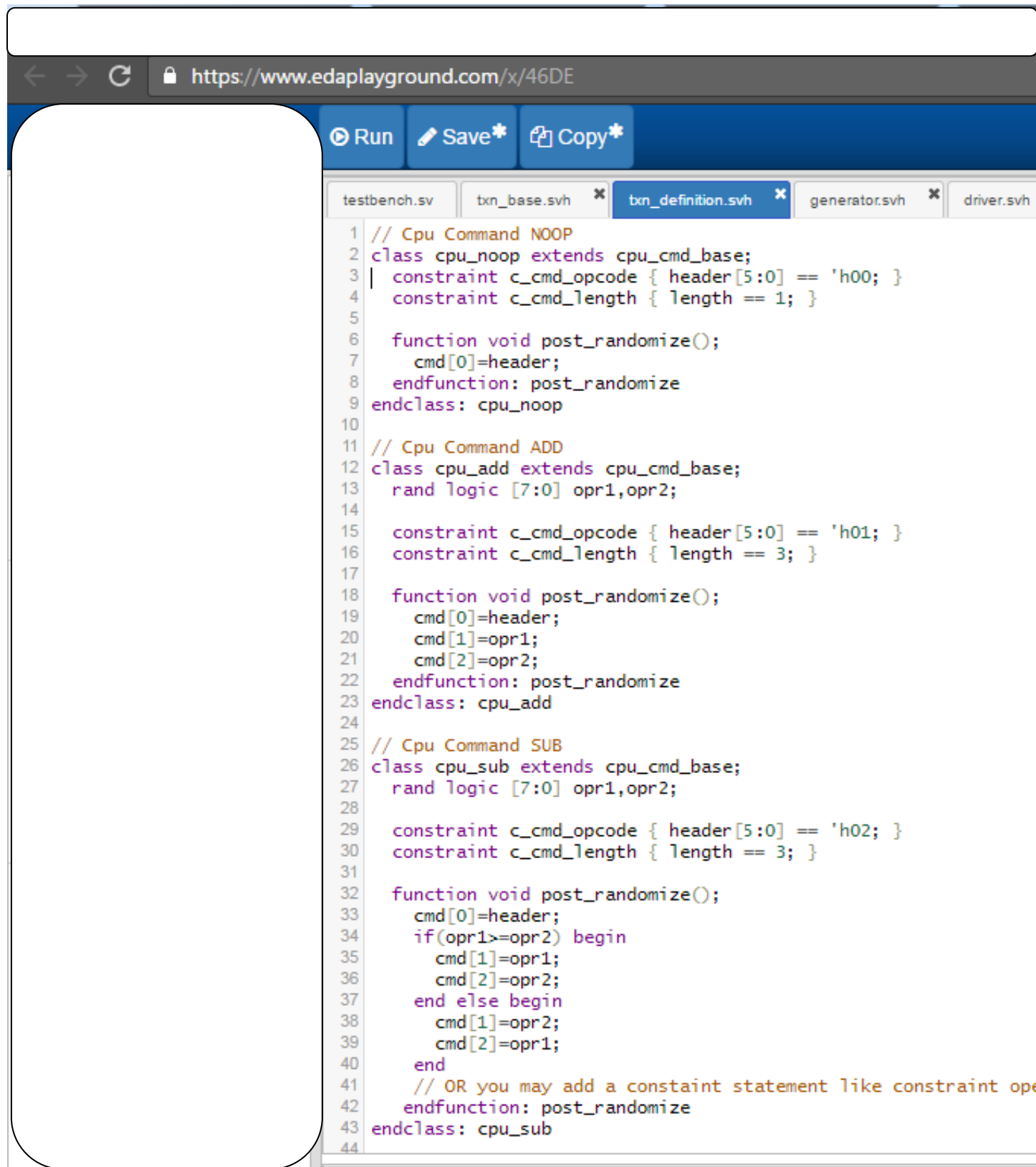


## 1. Txn Base Class



```
1 class cmd_base;
2   rand logic [7:0] header;
3   rand int      length;
4   rand logic [7:0] cmd[]; // Dynamic Array to hold the complete
5
6   function new();
7   endfunction: new
8
9   constraint c_cmd_total_length {
10    length >= 1;
11    cmd.size() == length;
12  }
13
14  function void print(string msg);
15    string tmp_str, name;
16
17    case (header)
18      'h00: name="CPU_NOOP ";
19      'h01: name="CPU_ADD  ";
20      'h02: name="CPU_SUB  ";
21      'h40: name="MEM_WAIT ";
22      'h41: name="MEM_LOAD ";
23      'h42: name="MEM_STORE";
24      'h43: name="MEM_COPY ";
25    endcase
26
27    msg={msg,"Cmd is : ",name," : "};
28    for(int i=0; i< length ;i++) begin //{
29      $sformat(tmp_str,"%2h ",cmd[i]);
30      msg={msg,tmp_str};
31    end //}
32    $display("%s",msg);
33  endfunction:print
34
35 endclass: cmd_base
36
37 //*****
38 class cpu_cmd_base extends cmd_base;
39   constraint c_cpu_cmd_opcode { header[7:6] == 'b00; }
40 endclass: cpu_cmd_base
41
42 class mem_cmd_base extends cmd_base;
43   constraint c_mem_cmd_opcode { header[7:6] == 'b01; }
44 endclass: mem_cmd_base
```

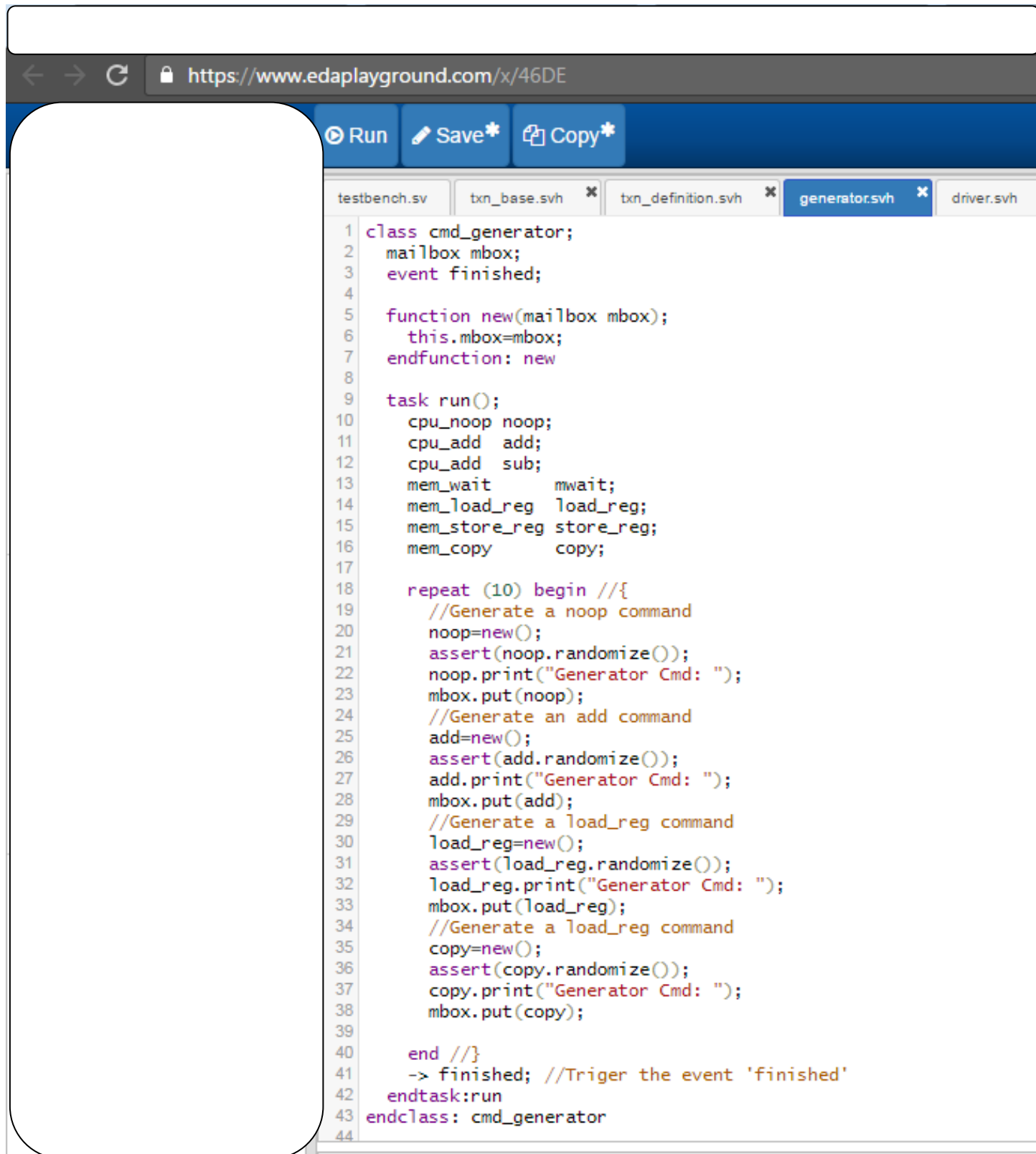
## 2. Transaction definition



The screenshot shows a web browser at <https://www.edaplayground.com/x/46DE> with a Verilog editor. The editor has tabs for `testbench.sv`, `txn_base.svh`, `txn_definition.svh` (active), `generator.svh`, and `driver.svh`. The code defines three transaction classes: `cpu_noop`, `cpu_add`, and `cpu_sub`, all extending `cpu_cmd_base`.

```
1 // Cpu Command NOOP
2 class cpu_noop extends cpu_cmd_base;
3 | constraint c_cmd_opcode { header[5:0] == 'h00; }
4   constraint c_cmd_length { length == 1; }
5
6   function void post_randomize();
7       cmd[0]=header;
8   endfunction: post_randomize
9 endclass: cpu_noop
10
11 // Cpu Command ADD
12 class cpu_add extends cpu_cmd_base;
13   rand logic [7:0] opr1,opr2;
14
15   constraint c_cmd_opcode { header[5:0] == 'h01; }
16   constraint c_cmd_length { length == 3; }
17
18   function void post_randomize();
19       cmd[0]=header;
20       cmd[1]=opr1;
21       cmd[2]=opr2;
22   endfunction: post_randomize
23 endclass: cpu_add
24
25 // Cpu Command SUB
26 class cpu_sub extends cpu_cmd_base;
27   rand logic [7:0] opr1,opr2;
28
29   constraint c_cmd_opcode { header[5:0] == 'h02; }
30   constraint c_cmd_length { length == 3; }
31
32   function void post_randomize();
33       cmd[0]=header;
34       if(opr1>=opr2) begin
35           cmd[1]=opr1;
36           cmd[2]=opr2;
37       end else begin
38           cmd[1]=opr2;
39           cmd[2]=opr1;
40       end
41       // OR you may add a constaint statement like constraint opr1 < opr2;
42   endfunction: post_randomize
43 endclass: cpu_sub
44
```

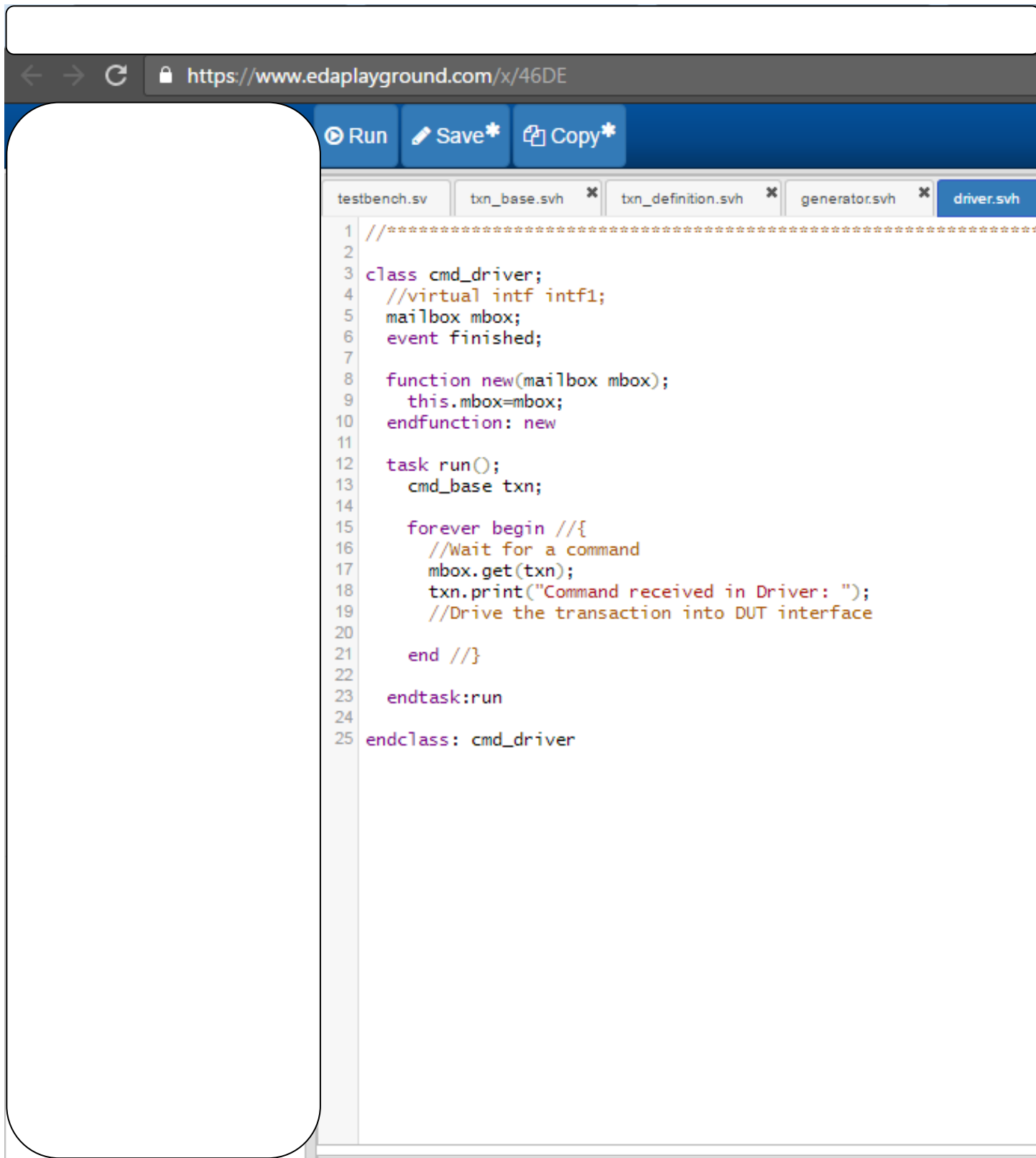
### 3. Generator



The screenshot shows the Edaplayground web interface at the URL [https://www.edaplightground.com/x/46DE](https://www.edaplayground.com/x/46DE). The interface includes a toolbar with 'Run', 'Save\*', and 'Copy\*' buttons. Below the toolbar, there are tabs for different files: 'testbench.sv', 'txn\_base.svh', 'txn\_definition.svh', 'generator.svh' (which is the active file), and 'driver.svh'. The 'generator.svh' file contains the following Verilog code:

```
1 class cmd_generator;
2     mailbox mbox;
3     event finished;
4
5     function new(mailbox mbox);
6         this.mbox=mbox;
7     endfunction: new
8
9     task run();
10        cpu_noop noop;
11        cpu_add add;
12        cpu_add sub;
13        mem_wait mwait;
14        mem_load_reg load_reg;
15        mem_store_reg store_reg;
16        mem_copy copy;
17
18        repeat (10) begin //{
19            //Generate a noop command
20            noop=new();
21            assert(noop.randomize());
22            noop.print("Generator Cmd: ");
23            mbox.put(noop);
24            //Generate an add command
25            add=new();
26            assert(add.randomize());
27            add.print("Generator Cmd: ");
28            mbox.put(add);
29            //Generate a load_reg command
30            load_reg=new();
31            assert(load_reg.randomize());
32            load_reg.print("Generator Cmd: ");
33            mbox.put(load_reg);
34            //Generate a load_reg command
35            copy=new();
36            assert(copy.randomize());
37            copy.print("Generator Cmd: ");
38            mbox.put(copy);
39
40        end //}
41        -> finished; //Triger the event 'finished'
42    endtask:run
43 endclass: cmd_generator
44
```

#### 4. Driver



The screenshot shows a web browser window at <https://www.edaplayground.com/x/46DE>. The interface includes a toolbar with 'Run', 'Save\*', and 'Copy\*' buttons. Below the toolbar, there are tabs for 'testbench.sv', 'txn\_base.svh', 'txn\_definition.svh', 'generator.svh', and 'driver.svh'. The 'driver.svh' tab is active, displaying the following Verilog code:

```
1 //*****
2
3 class cmd_driver;
4     //virtual intf intf1;
5     mailbox mbox;
6     event finished;
7
8     function new(mailbox mbox);
9         this.mbox=mbox;
10    endfunction: new
11
12    task run();
13        cmd_base txn;
14
15        forever begin //{
16            //Wait for a command
17            mbox.get(txn);
18            txn.print("Command received in Driver: ");
19            //Drive the transaction into DUT interface
20
21        end //}
22
23    endtask:run
24
25 endclass: cmd_driver
```

## 5. Environment

The screenshot shows a web browser window at <https://www.edaplayground.com/x/46DE>. The interface includes a toolbar with 'Run', 'Save\*', and 'Copy\*' buttons. Below the toolbar, there are tabs for different files: 'testbench.sv', 'txn\_base.svh', 'txn\_definition.svh', 'generator.svh', and 'driver.svh'. The 'testbench.sv' tab is active, displaying the following Verilog code:

```
1 class my_env;
2   cmd_generator gen;
3   cmd_driver   drv;
4   mailbox      gen2drv;
5
6   function new();
7     gen2drv= new();
8
9     gen = new(gen2drv);
10    drv = new(gen2drv);
11  endfunction: new
12
13  task run();
14    fork
15      gen.run();
16      drv.run();
17    join_none
18
19    wait ( gen.finished.triggered );
20
21    $finish();
22
23  endtask: run
24
25 endclass
```

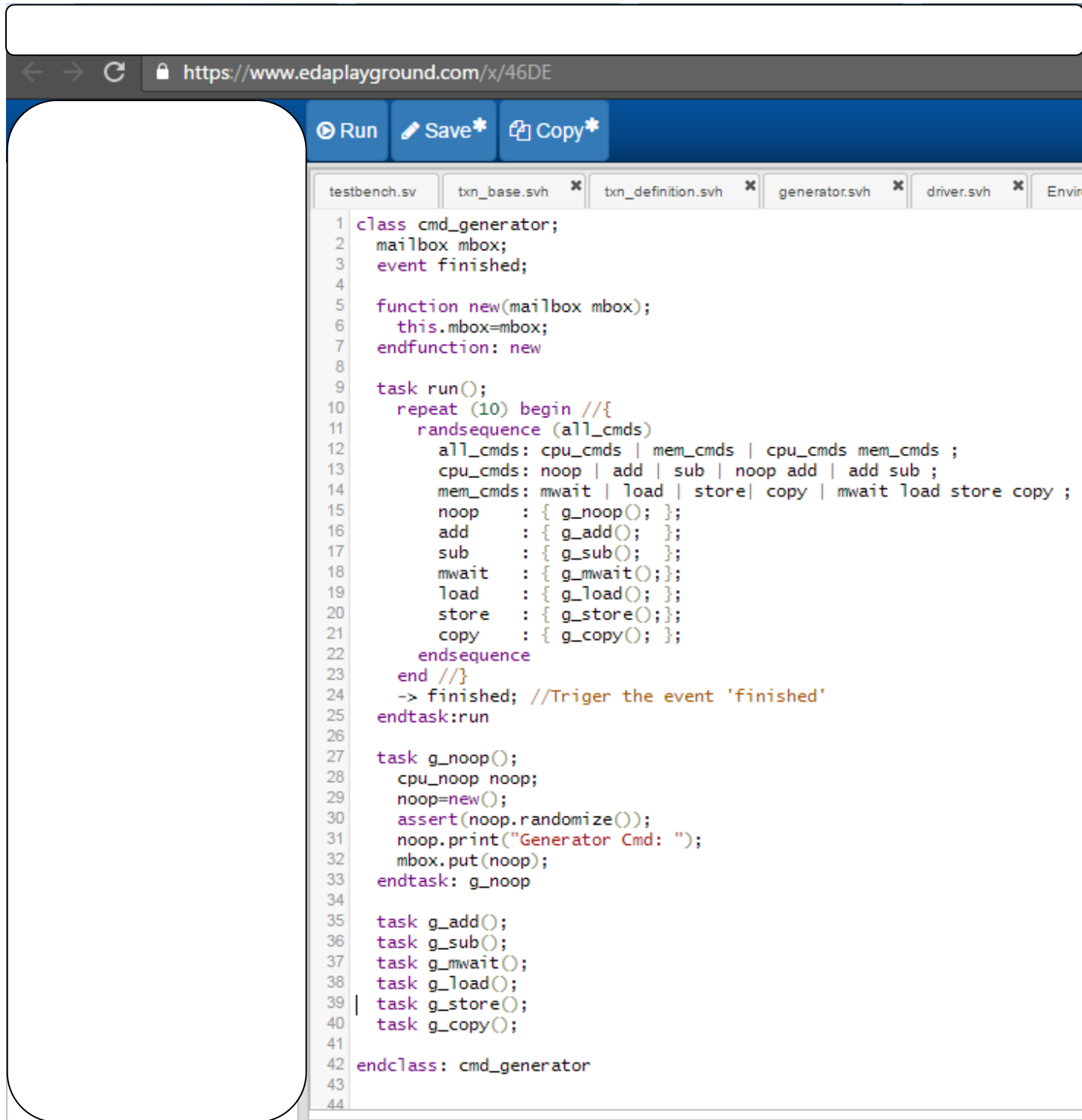
## 6. Test Bench

The screenshot shows the Edaplayground web interface for writing and running Verilog code. The browser address bar displays <https://www.edaplayground.com/x/46DE>. The interface includes a toolbar with 'Run', 'Save\*', and 'Copy\*' buttons. A tabbed editor shows the following code in `testbench.sv`:

```
1 // Code your testbench here
2
3 `include "txn_base.svh"
4 `include "txn_definition.svh"
5 //`include "generator.svh"
6 `include "generator_sequence.svh"
7 `include "driver.svh"
8 `include "Environment.svh"
9
10 module my_test_bench();
11
12     //Instantiate and connect the DUT.
13     my_design mydesing1();
14
15     // Instantiate the verification Env.
16     my_env env;
17
18     initial begin //{
19         env=new();
20         env.run();
21     end //}
22
23 endmodule: my_test_bench
24
```

Other open tabs include `txn_base.svh`, `txn_definition.svh`, `generator.svh`, and `driver.svh`.

## 7. Generator using Randsequence



The screenshot shows the Edaplayground web interface. The browser address bar displays <https://www.edaplayground.com/x/46DE>. The interface includes a toolbar with 'Run', 'Save\*', and 'Copy\*' buttons. Below the toolbar, a tabbed editor shows several files: 'testbench.sv', 'txn\_base.svh', 'txn\_definition.svh', 'generator.svh', 'driver.svh', and 'Environment.svh'. The 'generator.svh' file is active, displaying the following Verilog code:

```
1 class cmd_generator;
2     mailbox mbox;
3     event finished;
4
5     function new(mailbox mbox);
6         this.mbox=mbox;
7     endfunction: new
8
9     task run();
10        repeat (10) begin //{
11            randsequence (all_cmds)
12            all_cmds: cpu_cmds | mem_cmds | cpu_cmds mem_cmds ;
13            cpu_cmds: noop | add | sub | noop add | add sub ;
14            mem_cmds: mwait | load | store | copy | mwait load store copy ;
15            noop      : { g_noop(); };
16            add       : { g_add(); };
17            sub       : { g_sub(); };
18            mwait     : { g_mwait(); };
19            load      : { g_load(); };
20            store     : { g_store(); };
21            copy      : { g_copy(); };
22        endsequence
23        end //{
24        -> finished; //Triger the event 'finished'
25    endtask:run
26
27    task g_noop();
28        cpu_noop noop;
29        noop=new();
30        assert(noop.randomize());
31        noop.print("Generator Cmd: ");
32        mbox.put(noop);
33    endtask: g_noop
34
35    task g_add();
36    task g_sub();
37    task g_mwait();
38    task g_load();
39    task g_store();
40    task g_copy();
41
42 endclass: cmd_generator
43
44
```