ScalaHDL

by Yao Li

Outline

- Current State of ScalaHDL
    - “yld” in testbench (not implemented)
    - wrap around function
    - initial values
- Future Plan

“yld”

- What we expect:

```
instance {
  'rst := 0
  'a := A.next()
  'b := B.next()
  while (A.hasNext && B.hasNext) {
    'a := A.next()
    'b := B.next()
    yld(2)
  }
}
```

“yld”

- What we expect:

```
instance {
  'rst := 0
  'a := A.next()
  'b := B.next()
  while (A.hasNext && B.hasNext) {
    'a := A.next()
    'b := B.next()
    yld(2)
  }
}
```

Problem:
when the code block is executed,
we don’t really execute
A.next() or B.next(),
which results in *endless loop*…

“yld”

- There should be a solution. But we failed to find one simple enough that we don’t need to change much in current implementation of ScalaHDL.

- However, using the original ScalaHDL style test bench the users can already implement most features in the test bench of given examples.

- We suggest to leave this part as it is now, and improve it in some later phase of this project.

Wrap around function

- The previous version of Subtractor:

```
'z := ('a - 'b) % math.pow(2, size + 1).toInt
```

- Now we can write:

```
'z := wrap(('a - 'b), size + 1)
```

Wrap around function

- This wrap function will generate code like:

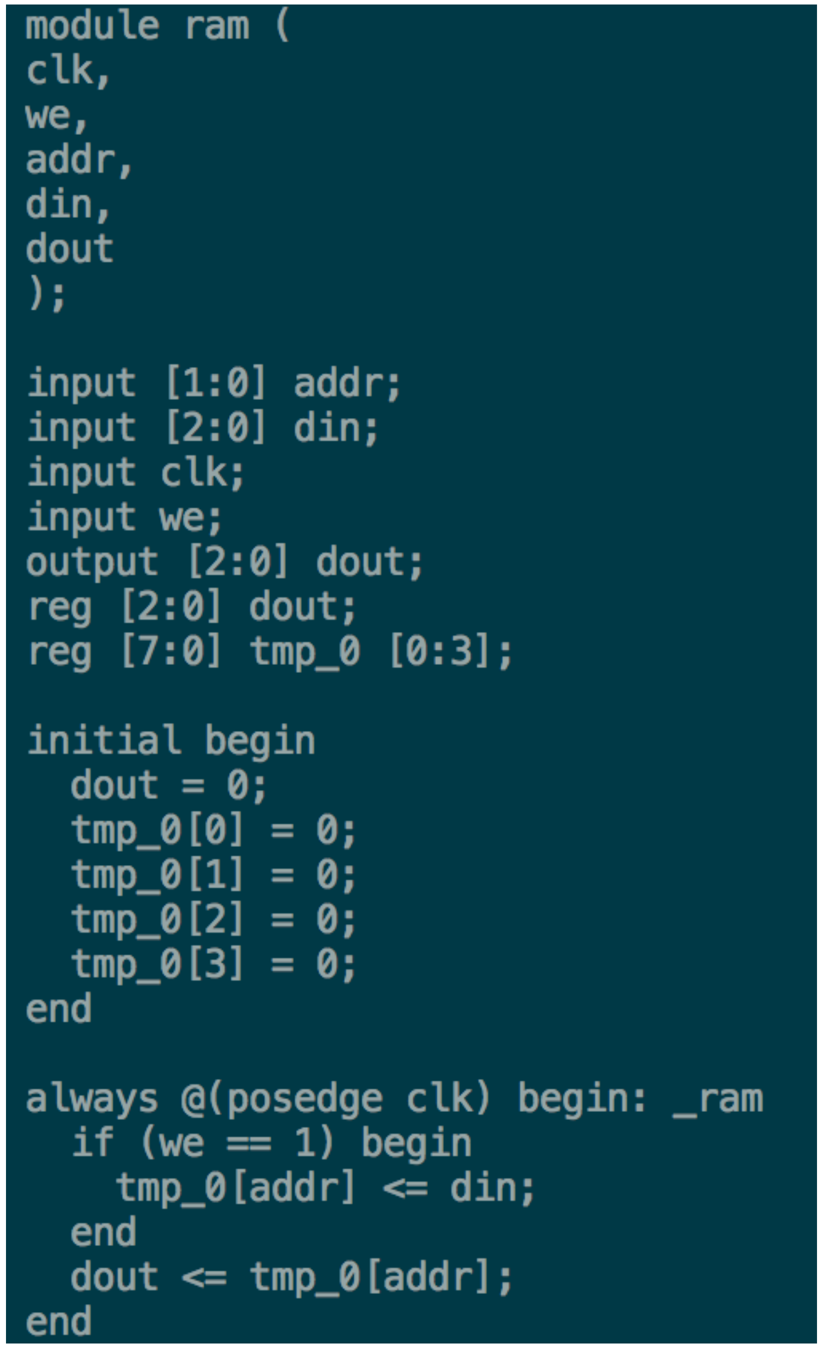
```
z <= ((a - b) % 32);
```

- We don’t use slicing because that will not produce correct result when negative number is involved.

Initial Values

- Every register will be assigned with an initial value in “initial block”.

Initial Values



```
module ram (
clk,
we,
addr,
din,
dout
);

input [1:0] addr;
input [2:0] din;
input clk;
input we;
output [2:0] dout;
reg [2:0] dout;
reg [7:0] tmp_0 [0:3];

initial begin
  dout = 0;
  tmp_0[0] = 0;
  tmp_0[1] = 0;
  tmp_0[2] = 0;
  tmp_0[3] = 0;
end

always @(posedge clk) begin: _ram
  if (we == 1) begin
    tmp_0[addr] <= din;
  end
  dout <= tmp_0[addr];
end
```

Future Plan

- A more real-world example.
- More tests.
- Better warnings and exceptions.

Any Question?

Thanks!