Registers

January 29, 2025

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

The n-bit register, step by step

Registers

- Registers are what software types call variables
- From hardware point of view these are **fast** variables
- Registers can be viewed as D-FFs with longer memory
- (and some other abilities)

Registers

1-Bit Register

v1

v2

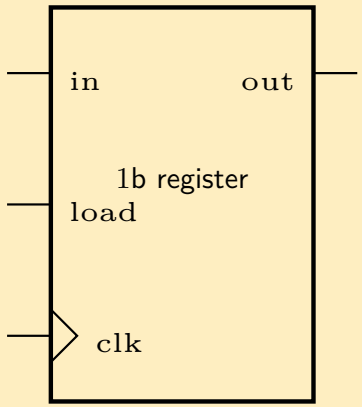
n-Bit Register

v1

v2

v3

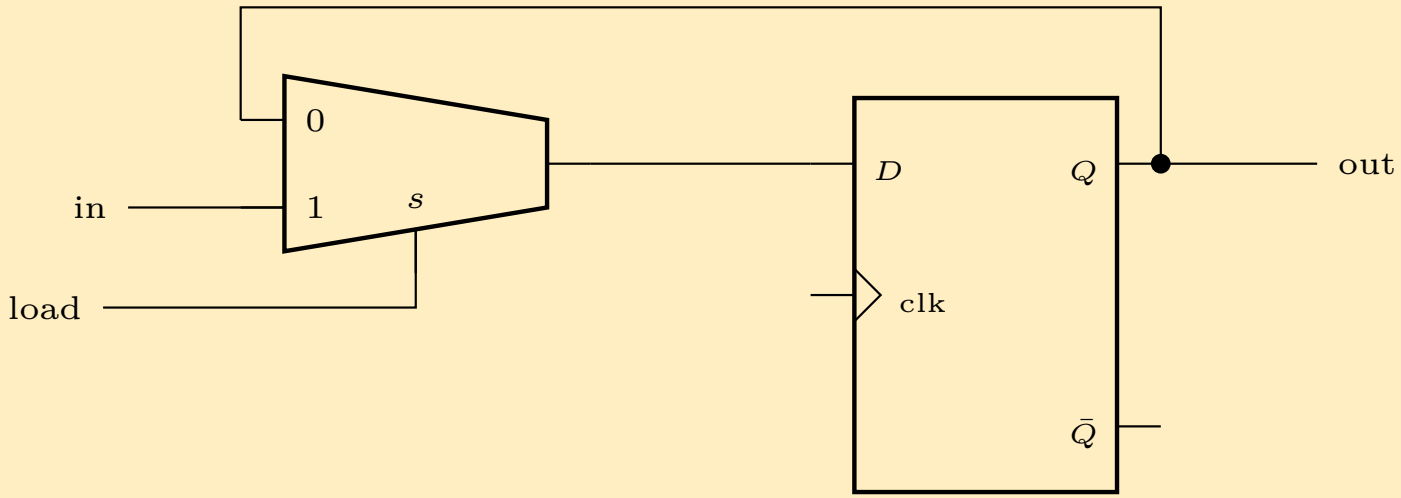
1-Bit Register (v1)



$$\text{out}(t+1) = \begin{cases} \text{out} & \text{if load} = 0, \\ \text{in} & \text{if load} = 1. \end{cases}$$

(Kind of a strange SR-FF)

1-Bit Register (v1), Possible Realization (circuit)



Make a good realiziation with SR-FF

1-Bit Register (v1) (module) I

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

Listing 1: reg1bit.v

```
‘timescale 1ns/1ns

module  reg1bit(
  output out,
  input  in,
  input  load,
  clk
);
  wire [1:0] options;
  wire q;

  assign options[0] = q;
  assign options[1] = in;
```

1-Bit Register (v1) (module) II

```
  wire d;
  mux #(1) m(d, options, load);

  dffi dff(q,,d,1'b0,1'b0,clk);

  assign out = q;

endmodule
```

1-Bit Register (v1) (testbench) I

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

Listing 2: reg1bit_tb.v

```
‘timescale 1ns/1ns

module reg1bit_tb;

  wire clk;
  clock #(80ns) c(clk);

  logic load, in;
  wire out;
  reg1bit r(out, in, load, clk);

  initial begin
    $dumpfile("reg1bit.vcd");
```

1-Bit Register (v1) (testbench) II

```
    $dumpvars;

    load = 0;
    in = 0;

    repeat (100) begin
      #220ns;
      if ($urandom_range(0,2) == 0)
        load = load + 1;
      if ($urandom_range(0,2) == 0)
        in = in + 1;
    end
    $finish;
  end
endmodule;
```

1-Bit Register (v1) (testbench) III

Registers

1-Bit Register

v1

v2

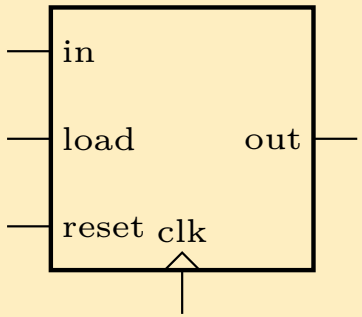
n-Bit Register

v1

v2

v3

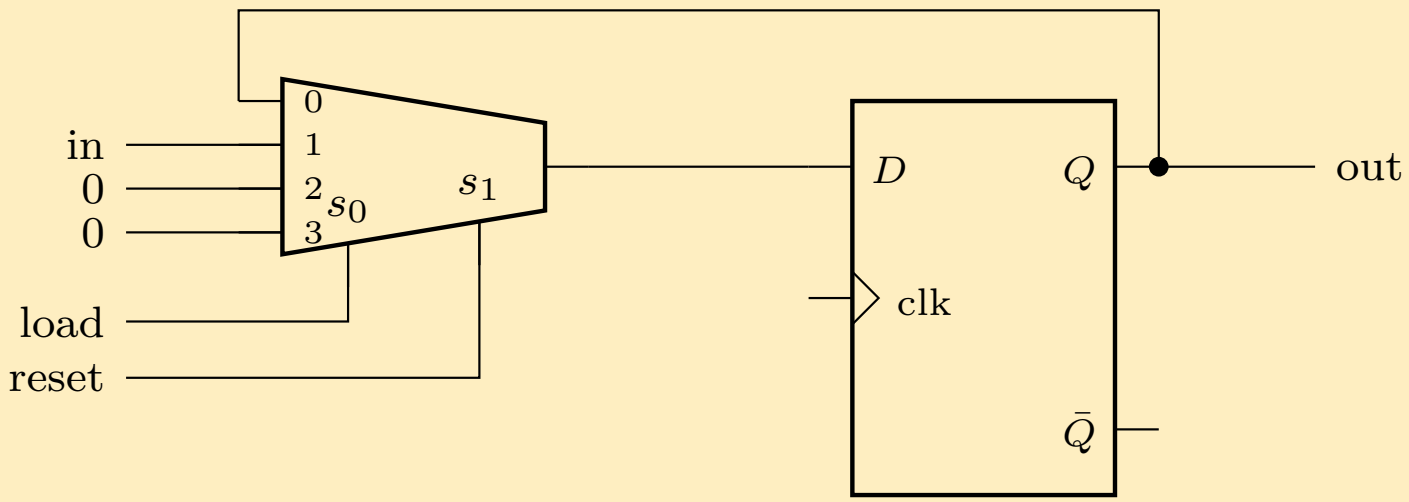
1-Bit Register (v2) (block)



$$\text{out}(t+1) = \begin{cases} \text{in} & \text{if load} = 1, \\ 0 & \text{if reset} = 1, \\ \text{out} & \text{otherwise.} \end{cases}$$

- **Do not** set both load and reset at the same time
- Specific implementation behaves differently when both controls are set

1-Bit Register (v2) (circuit)



- Realize the above with the `dffi` component
- Realize the above using the v1 register

1-Bit Register v2 (module) I

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

Listing 3: reg1bit.v

```
‘timescale 1ns/1ns

module  reg1bit(
  output out,
  input  in,
  input  load,
  input  reset,
  clk
);
  wire d, q;
  wire [3:0] options;
  wire [1:0] s;
```

1-Bit Register v2 (module) II

```
  assign s[1] = reset;
  assign s[0] = load;

  assign options[0] = q;
  assign options[1] = in;
  assign options[2] = 0;
  assign options[3] = 0;

  assign out = q;

  mux #(2) m(d, options, s);
  dffi dff(q,,d,1'b0,1'b0,clk);

endmodule
```

1-Bit Register v2 (testbench) I

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

Listing 4: reg1bit_tb.v

```
`timescale 1ns/1ns

module reg1bit_tb;
  wire clk;
  clock #(80ns) c(clk);

  logic load, in, reset;
  wire out;
  reg1bit r(out, in, load, reset, clk);

  initial begin
    $dumpfile("reg1bit.vcd");
    $dumpvars;
```

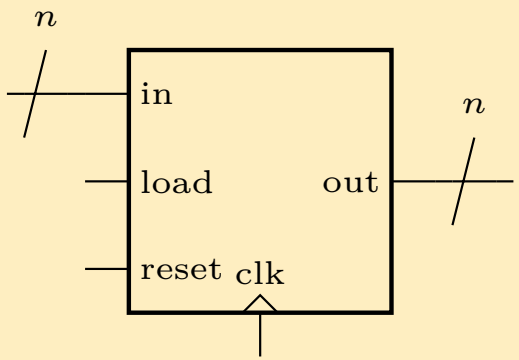
1-Bit Register v2 (testbench) II

```
load = 0;
in = 0;
reset = 1;

repeat (100) begin
  #80ns;
  if (reset == 0) begin
    if ($urandom_range(0,5) == 0)
      reset++;
  end else begin
    if ($urandom_range(0,1) == 0)
      reset++;
  end
  if ($urandom_range(0,1) == 0)
```

1-Bit Register v2 (testbench) III

```
        load++;
      if ($urandom_range(0,2) == 0)
        in++;
    end
    $finish;
  end
endmodule
```

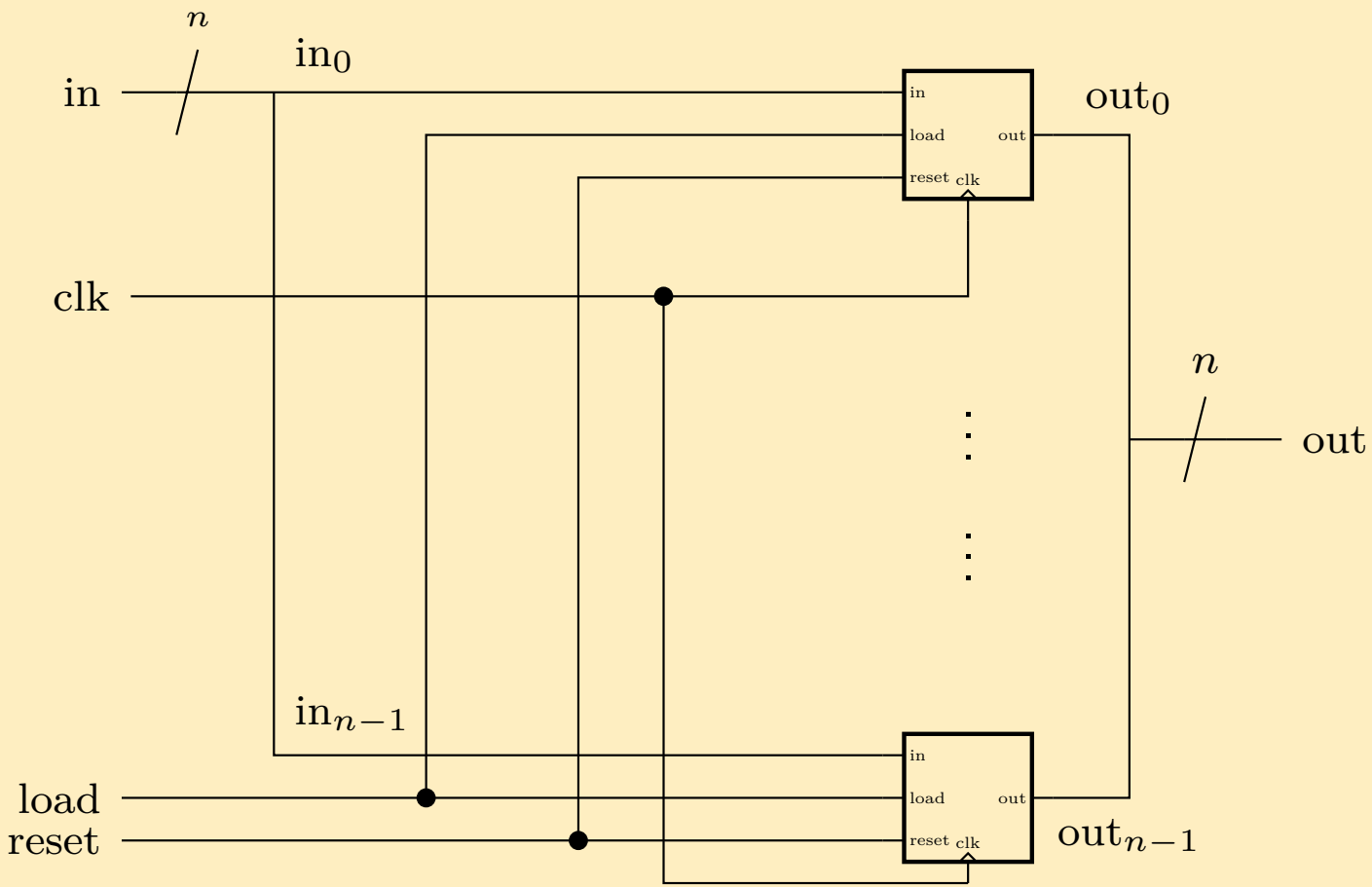
n-Bit Register v1 (block diagram)



$$\text{out}(t+1) = \begin{cases} \text{in} & \text{if load} = 1, \\ 0 & \text{if reset} = 1, \\ \text{out} & \text{otherwise.} \end{cases}$$

- Do not set load and reset at the same time
- When both controls are set, unexpected behavior

n-Bit Register v1 (circuit)



n-Bit Register v1 (module)

Registers
1-Bit Register
v1
v2
n-Bit Register
v1
v2
v3

Listing 5: regNbit.v

```
‘timescale 1ns/1ns

module  regNbit #(parameter WIDTH=1) (
  output [WIDTH-1:0] out,
  input  [WIDTH-1:0] in,  // value to store
  input  load,            // true for load
  input  reset,           // true for reset
  input  clk
);
  for (genvar i = 0; i < WIDTH; i++)
    reg1bit r(out[i],in[i],load,reset,clk);
endmodule
```

n-Bit Register v1 (testbench) I

Registers
1-Bit Register
v1
v2
n-Bit Register
v1
v2
v3

Listing 6: regNbit.v

```
`timescale 1ns/1ns

module regNbit_tb;

  parameter WIDTH=8;

  wire clk;
  clock   #(80) c(clk);

  logic load, reset;
  logic [WIDTH-1:0] in;
  wire  [WIDTH-1:0] out;
```

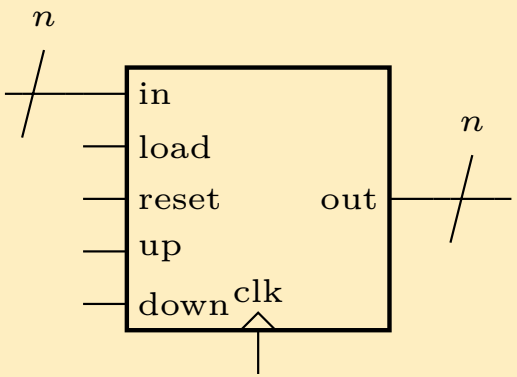
n-Bit Register v1 (testbench) II

```
regNbit #(WIDTH) r(out,in,load,
                   reset,clk);

initial begin
  $dumpfile("regNbit.vcd");
  $dumpvars;

  load = 0;
  in = 0;
  reset = 1;

  repeat (100) begin
    #80ns;
    if (reset == 0) begin
      if ($urandom_range(0,5) == 0)
```

n-Bit Register v1 (testbench) III

```
            reset++;
        end else begin
          if ($urandom_range(0,1) == 0)
            reset++;
        end
        if ($urandom_range(0,1) == 0)
          load++;
        if ($urandom_range(0,2) == 0)
          in = $urandom_range(0,2**WIDTH-1);
      end
      $finish;
    end

endmodule
```

n-Bit Register v2 (block diagram)



$$\text{out}(t+1) = \begin{cases} \text{in} & \text{if load} = 1, \\ \text{out} + 1 & \text{if up} = 1, \quad (+ \text{ is addition}) \\ \text{out} - 1 & \text{if down} = 1, \\ 0 & \text{if reset} = 1, \\ \text{out} & \text{otherwise.} \end{cases}$$

- Set **only one** of load, reset, up and down at a time

n-bit register v2 (module) I

Listing 7: register.v

```
`timescale 1ns/1ns

module  register #(parameter WIDTH=1) (
  output [WIDTH-1:0] out,
  input  [WIDTH-1:0] in,  // value to store
  input  down,
  input  up,
  input  load,            // true for load
  input  reset,           // true for reset
  input  clk
);

  wire [2:0] sel;
```

n-bit register v2 (module) II

```
assign sel[2] = load;
assign sel[1] = up;
assign sel[0] = down;

wire [WIDTH-1:0] options[7:0];
for (genvar i=4; i < 8; i++)
  assign options[i] = in;
for (genvar i=2; i < 4; i++)
  assign options[i] = oneMore;
assign options[1] = oneLess;
assign options[0] = d;


wire [WIDTH-1:0] oneMore,d;
```


Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

n-bit register v2 (module) III

```
  binaryAdder #(WIDTH) a(,, oneMore, d,
       {{WIDTH-1{1'b0}},1'b1},1'b0);


  wire [WIDTH-1:0] oneLess;
  binaryAdder #(WIDTH) s(,, oneLess, d,
       {{WIDTH{1'b1}}},1'b0);


  wire [WIDTH-1:0] in;
  muxMulti #(3,WIDTH) m(in, options, sel);

  regNbit #(WIDTH) r(d, in, load, reset, clk);

endmodule
```

n-bit register v2 (module) IV

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

n-bit register v2 (Testbench) I

Registers
1-Bit Register
v1
v2
n-Bit Register
v1
v2
v3

Listing 8: register_tb.v

```
‘timescale 1ns/1ns

module register_tb;

  parameter WIDTH=8;

  wire clk;
  clock   #(80) c(clk);

  logic load, reset,up,down;
  logic [WIDTH-1:0] in;
  wire [WIDTH-1:0] out;
```

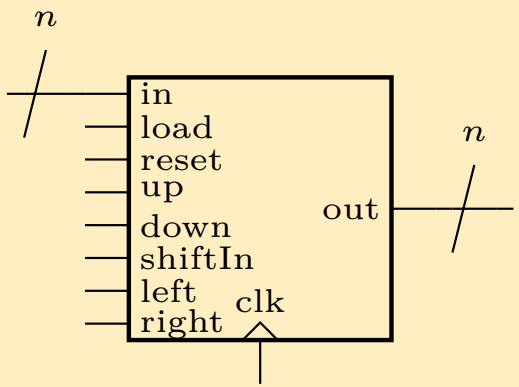
n-bit register v2 (Testbench) II


```
register #(WIDTH) r(out,in,down,up,load,reset,clk

initial begin
  $dumpfile("register.vcd");
  $dumpvars;

  load = 0;
  in = 0;
  reset = 1;

  repeat (100) begin
    #80ns;
    if (reset == 0) begin
      if ($urandom_range(0,5) == 0)
        reset++;
```

n-bit register v2 (Testbench) III

```
      end else begin
        if ($urandom_range(0,1) == 0)
          reset++;
      end
      if ($urandom_range(0,1) == 0)
        load++;
      if ($urandom_range(0,2) == 0)
        in = $urandom_range(0,2**WIDTH-1);
    end
    $finish;
  end

endmodule
```

n-Bit Register v3 (block diagram)



$$\text{out}(t+1) = \begin{cases} \text{in} & \text{if load} = 1, \\ \text{out} + 1 & \text{if up} = 1, \\ \text{out} - 1 & \text{if down} = 1, \\ \text{out} << 1 + \text{shiftIn} & \text{if left} = 1, \\ \text{out} >> 1 + \text{shiftIn} \times 2^{n-1} & \text{if right} = 1, \\ 0 & \text{if reset} = 1, \\ \text{out} & \text{otherwise.} \end{cases}$$

(+ is addition)

n-bit register v3 (module) I

Listing 9: register.v

```
`timescale 1ns/1ns

module  register #(parameter WIDTH=1) (
  output [WIDTH-1:0] out,
  input  [WIDTH-1:0] in,  // value to store
  input  shiftIn,
  input  right,
  input  left,
  input  down,
  input  up,
  input  load,              // true for load
  input  reset,             // true for reset
  input  clk
```

n-bit register v3 (module) II

```
);

  wire [4:0] sel;
  wire [WIDTH-1:0] oneMore,d;
  wire [WIDTH-1:0] oneLess;
  wire [WIDTH-1:0] shiftLeft;
  wire [WIDTH-1:0] shiftRight;
  wire [WIDTH-1:0] in;

  assign sel[4] = load;
  assign sel[3] = up;
  assign sel[2] = down;
  assign sel[1] = left;
  assign sel[0] = right;
```

n-bit register v3 (module) III

```
wire [WIDTH-1:0] options[31:0];
for (genvar i=16; i < 32; i++)
  assign options[i] = in;
for (genvar i=8; i < 16; i++)
  assign options[i] = oneMore;
for (genvar i=4; i < 8; i++)
  assign options[i] = oneLess;
for (genvar i=2; i < 4; i++)
  assign options[i] = shiftLeft;
assign options[1] = shiftRight;
assign options[0] = d;

binaryAdder #(WIDTH) a(,, oneMore, d,
     {{WIDTH-1{1'b0}},1'b1},1'b0);
```

n-bit register v3 (module) IV

```
binaryAdder #(WIDTH) s(,, oneLess, d,
      {{WIDTH{1'b1}}},1'b0);

assign shiftLeft[0] = shiftIn;
for (genvar i = 0; i < WIDTH-1; i++)
  assign shiftLeft[i+1] = d[i];

assign shiftRight[WIDTH-1] = shiftIn;
for (genvar i = 0; i < WIDTH-1; i++)
begin: sr
  assign shiftRight[i] = d[i+1];
end


wire [WIDTH-1:0]value;
```

n-bit register v3 (module) V


```
  muxMulti #(5,WIDTH) m(value, options, sel);

  wire realLoad;
  or5Gate o(realLoad,load,up,down,right,left);

  regNbit #(WIDTH) rN(d, value, realLoad, reset, cl

  assign out=d;

endmodule
```

Registers
1-Bit Register
v1
v2
n-Bit Register
v1
v2
v3

Register v3 (Testbench) I

Listing 10: register_tb.v

```
`timescale 1ns/1ns

module register_tb;

  parameter WIDTH=8;

  wire clk;
  clock #(80) c(clk);

  logic load, reset,up,down,right,left;
  logic shiftIn;
  logic [WIDTH-1:0] in;
  wire [WIDTH-1:0] out;
```

Register v3 (Testbench) II

```
register #(WIDTH) r(out,in,shiftIn,
                    right,left,down,
                    up,load,reset,clk);

initial begin
  $dumpfile("register.vcd");
  $dumpvars;

  load = 0;
  in = 0;
  reset = 1;

  repeat (100) begin
    #80ns;
```

Register v3 (Testbench) III

```
      if (reset == 0) begin
        if ($urandom_range(0,5) == 0)
          reset++;
      end else begin
        if ($urandom_range(0,1) == 0)
          reset++;
      end
      if ($urandom_range(0,1) == 0)
        load++;
      if ($urandom_range(0,2) == 0)
        in = $urandom_range(0,2**WIDTH-1);
    end
    $finish;
  end
```

Register v3 (Testbench) IV

```
endmodule
```


Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

Library Modules

```
reg1bit(output out, input in,load,reset,clk)
regNbit #(N) (output [N-1:0]out,input [N-1:0]in,
                                  input load,reset,clk)
register #(N) (output [N-1:0]out,input [N-1:0]in,
                    input shiftIn,right,left,down,up,
                                        load,reset,clk)
```

(Carmi) Lecture 20 reached here

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

Overlapping Sequence (module) I

Listing 11: register.v

```
‘timescale 1ns/1ns

module  seq  (
  output found,
  input  in,
  input  clk
);

  wire [4:0]d;
  logic reset,up;
  register #(5) r(d, 5’b0,in,1’b1,1’b0,
                                1’b0,1’b0,1’b0,
                                1’b0, clk);
```

Overlapping Sequence (module) II

```
  comparator #(5) c(,found,,d,5'b01100);
endmodule
```

Overlapping Sequence (Testbench) I

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

Listing 12: register_tb.v

```
`timescale 1ns/1ns

module seq_tb;

  wire clk;
  clock #(500) c(clk);

  wire found;
  logic in;
  seq s(found,in,clk);

  initial begin
    $dumpfile("seq.vcd");
```

Overlapping Sequence (Testbench) II

```
    $dumpvars;

    in = 0;
    #20000ns;

    $finish;
  end

  always @(posedge(clk)) begin
    #5ns;
    in <= $urandom_range(0,1);
  end


endmodule
```

Overlapping Sequence (Testbench) III

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

Overlapping Sequence Fix (module) I

Listing 13: register.v

```
‘timescale 1ns/1ns

module  seq (
  output found,
  input  in,
  input  reset,
  input  clk
);

  wire [4:0]d;
  register #(5) r(d, 5’b0,in,1’b1,1’b0,
                             1’b0,1’b0,1’b0,
                             1’b0, clk);
```

Overlapping Sequence Fix (module) II

```
  wire eq;
  comparator #(5) cmp(,eq,,d,5'b01100);

  wire [2:0]step;
  wire work,up;

  notGate n(up,work);
  register #(3) cnt(step,3'b0,1'b0,1'b0,1'b0,
                    1'b0,up,1'b0,reset,clk);
  comparator #(3) cmp5(,work,,step,3'b101);

  andGate mask(found,eq,work);
endmodule
```

Overlapping Sequence Fix (Testbench) I

Listing 14: register_tb.v

```
`timescale 1ns/1ns

module seq_tb;

  wire clk;
  clock #(500) c(clk);

  wire found;
  logic in;
  logic reset;

  seq s(found,in,reset,clk);
```

Overlapping Sequence Fix (Testbench) II

```
initial begin
  $dumpfile("seq.vcd");
  $dumpvars;

  in = 0;
  reset=1;
  #350ns;
  reset=0;

  #20000ns;

  $finish;
end

always @(posedge(clk)) begin
```

Overlapping Sequence Fix (Testbench) III

```
    #5ns;
    in <= $urandom_range(0,1);
  end


endmodule
```

Disjoint Sequence (module) I

Listing 15: register.v

```
‘timescale 1ns/1ns

module  seq (
  output found,
  input  in,
  input  reset,
  input  clk
);

  wire [4:0]d;
  register #(5) r(d, 5’b0,in,1’b1,1’b0,
                                1’b0,1’b0,1’b0,
                                reset, clk);
```

Disjoint Sequence (module) II

```
  wire eq;
  comparator #(5) cmp(,eq,,d,5'b01100);

  wire [2:0]step;
  wire work,up;

  orGate o(realReset,reset,work);

  register #(3) cnt(step,3'b0,1'b0,1'b0,1'b0,
                    1'b0,1'b1,1'b0,realReset,clk);
  comparator #(3) cmp5(,work,,step,3'b100);

  andGate mask(found,eq,work);
endmodule
```

Disjoint Sequence (module) III

Registers

1-Bit Register

v1

v2

n-Bit Register

v1

v2

v3

Disjoint Sequence (Testbench) I

Listing 16: register_tb.v

```
`timescale 1ns/1ns

module seq_tb;

  wire clk;
  clock #(500) c(clk);

  wire found;
  logic in;
  logic reset;

  seq s(found,in,reset,clk);
```

Disjoint Sequence (Testbench) II

```
initial begin
  $dumpfile("seq.vcd");
  $dumpvars;

  in = 0;
  reset=1;
  #350ns;
  reset=0;

  #20000ns;

  $finish;
end

always @(posedge(clk)) begin
```

Disjoint Sequence (Testbench) III

```
    #5ns;
    in <= $urandom_range(0,1);
  end


endmodule
```

(Carmi) Lecture 21 reached here

1. The above took way longer than expected
2. The RTL attempt was disaster
3. We will redo next time