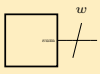
Register Transfer Level (RTL)

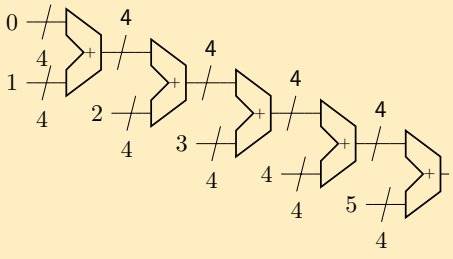
February 4, 2025

Discussion: $\sum_{i=0}^{n} i$

$\sum_{i=0}^{n} i$ combinational circuit (block)



$\sum_{i=0}^{5} i$ combinational circuit



$\sum_{i=0}^{n} i$ combinational (verilog) I

Listing 1: sum.v

```
‘timescale 1ns/1ns

module sum #(parameter W=5,parameter N=5)(
    output [W-1:0] out
);
    wire [W-1:0] s[N+1];

    assign s[0] = 0;
    for (genvar i = 1; i <= N; i++) begin
        wire [W-1:0] t,ss;
        assign t = i;
        assign ss=s[i]; //for GTKwave
        binaryAdder #(W) a(,,s[i],s[i-1],
```


$\sum_{i=0}^{n} i$ combinational (verilog) II

```
                                t,1'b0);
    end

    assign out = s[N];
endmodule
```


$\sum_{i=0}^{n} i$ combinational (testbench) I

Listing 2: sum_tb.v

```
‘timescale 1ns/1ns

module sum_tb;

    parameter W = 24;
    parameter N = 1000;

    wire [W-1:0] out;

    sum #(W,N) s(out);


    initial begin
```


$\sum_{i=0}^{n} i$ combinational (testbench) II

```
        $dumpfile("sum.vcd");
        $dumpvars;

        #10000ns;

        $display(out);
        $finish();
    end

endmodule;
```

$\sum_{i=0}^{n} i$ combinational: So?

- Probably the fastest
- Lots of hardware
- It reallly is kind of insane
- What do we do if we need several Ns?
- **Obviously we need a program**

$\sum_{i=0}^{n} i$

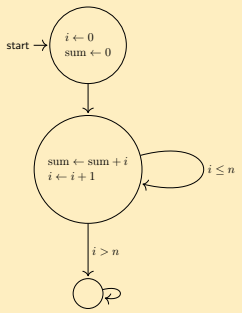
Program

$\sum_{i=0}^{n} i$ Program

```
sum = 0;
for (i = 0; i <= n; i++)
        sum = sum + i;
```

- We need to **build** the program
- Some concept of order should be introduced: States

$\sum_{i=0}^{n} i$ State Diagram Approximation



We **Split** the design

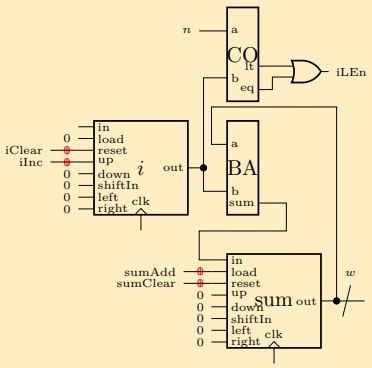
1. Data path: **What** needs to be done
2. Control: **When** what needs to be done should be done

$\sum_{i=0}^{n} i$ What needs to be Done

```
i ← 0
sum ← 0
i ← i + 1
sum ← sum + i
i ≤ n
```


Method 1: Control Controls the Controls

$\sum_{i=0}^{n} i$ Data Path (What **can** be done)



$i \leftarrow 0$

$i \leftarrow i + 1$

$\text{sum} \leftarrow 0$

$\text{sum} \leftarrow \text{sum} + i$

$\sum_{i=0}^{n} i$ Data Path Verilog I

Listing 3: demo/02_sum/sumDP.v

```
‘timescale 1ns/1ns

module sumDP #(parameter W=1) (
    output [W-1:0]out,
    output iLEn,
    input  iClear,
    input  iInc,
    input  sumClear,
    input  sumAdd,
    input  [W-1:0]n,
    input  clk
    );
```

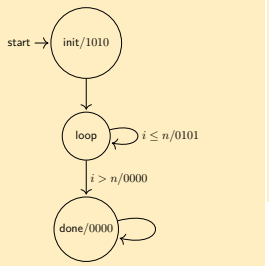
$\sum_{i=0}^{n} i$ Data Path Verilog II

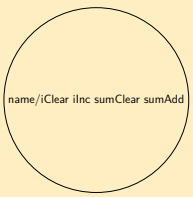
```
    wire [W-1:0] sumOut;
    wire [W-1:0] sumPi;
    register #(W) sum(sumOut,sumPi,1'b0,
                      1'b0,1'b0,
                      1'b0,1'b0,
                      sumAdd,sumClear,
                      clk);
    assign out = sumOut;

    wire [W-1:0] iOut;
    register #(W) i(iOut,W'(0),1'b0,
                            1'b0,1'b0,
                            1'b0,iInc,
                            iClear,1'b0,
                            clk);
```


$\sum_{i=0}^{n} i$ Data Path Verilog III

```
    binaryAdder #(W) ba(,,sumPi,sumOut,
                            iOut,1'b0);

    comparator #(W) com(lt,eq,,iOut,n);
    orGate o(iLEn,eq,lt);
endmodule
```


$\sum_{i=0}^{n} i$ State Diagram





This is a **Mealy** machine

State Codes (1-hot)

Name	Code
init	001
loop	010
done	100

State Table and State Equations

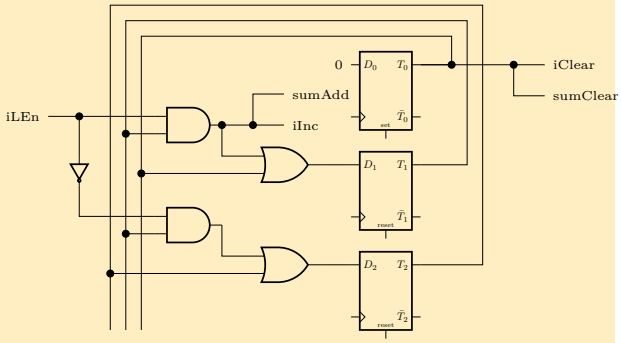
$T_2T_1T_0$	iLEn	Next $T_2T_1T_0$	iClear	iInc	sumClear	sumAdd
0 0 1	X	0 1 0	1	0	1	0
0 1 0	0	1 0 0	0	0	0	0
0 1 0	1	0 1 0	0	1	0	1
1 0 0	X	1 0 0	0	0	0	0

D-FFs so columns D_2,D_1,D_0 are not shown above

$$D_0 = 0$$
$$D_1 = T_0 + T_1\text{iLEn}$$
$$D_2 = T_2 + T_1\overline{\text{iLEn}}$$

$$\text{iClear} = T_0$$
$$\text{sumClear} = T_0$$
$$\text{iInc} = T_1\text{iLTn}$$
$$\text{sumAdd} = T_1\text{iLTn}$$

Control (circuit)



Control I

Listing 4: 02_sumCTL.v

```
‘timescale 1ns/1ns

module sumCTL (
    output iClear,
    output iInc,
    output sumClear,
    output sumAdd,
    input  iLEn,
    input  reset,
    input  clk
    );

    wire t0,t1,t2,d0,d1,d2;
```

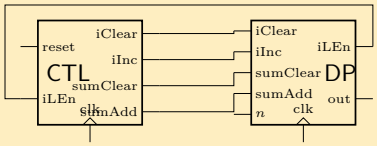

Control II

```
    dffi dff0(t0,,d0,reset,1'b0,clk);
    dffi dff1(t1,,d1,1'b0,reset,clk);
    dffi dff2(t2,,d2,1'b0,reset,clk);

    assign d0=0;

    wire iLEnn,t1LE,t1LEn;
    notGate ciLEnn(iLEnn,iLEn);
    andGate ct1LE(t1LE,t1,iLEn);
    andGate ct1LEn(t1LEn,t1,iLEnn);

    orGate cd1(d1,t0,t1LE);
    orGate cd2(d2,t2,t1LEn);

    assign iClear = t0;
```

Control III

```
    assign sumClear = t0;

    assign iInc = t1LE;
    assign sumAdd = t1LE;
endmodule
```


$\sum_{i=0}^{n} i$ Circuit



$\sum_{i=0}^{n} i$ Verilog I

Listing 5: sum.v

```
‘timescale 1ns/1ns

module sum #(parameter W=1) (
    output [W-1:0]out,
    input  [W-1:0]n,
    input  reset,
    input  clk
    );

    wire sumClear,sumAdd,iClear,iInc;
    wire iLEn;

    sumCTL  ctl(iClear,iInc,sumClear,sumAdd,
```


$\sum_{i=0}^{n} i$ Verilog II

```
                iLEn,reset,clk);
    sumDP #(W) dp(out,iLEn,iClear,iInc,
                sumClear,sumAdd,n,
                clk);
endmodule
```

$\sum_{i=0}^{n} i$ Testbench I

Listing 6: sum_tb.v

```
‘timescale 1ns/1ns

module sum_tb;
    parameter W = 24;
    parameter N = 1000;

    clock #(200) cclk(clk);

    logic reset;
    wire [W-1:0]value;
    wire iLTn;

    sum #(W) sum(value,W’(N),reset,clk);
```


$\sum_{i=0}^{n} i$ Testbench II

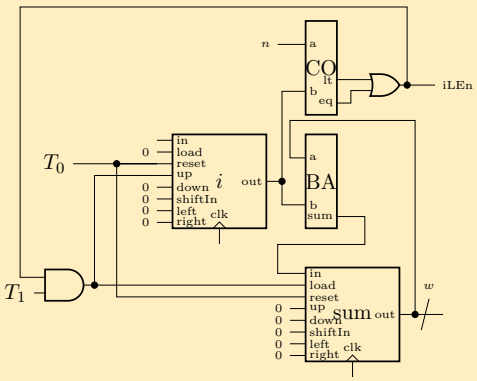
```
    initial begin
        $dumpfile("sum.vcd");
        $dumpvars;

        reset=1;
        #300ns;
        reset=0;

        #1000000ns;

        $display(value);
        $finish();
    end
```


$\sum_{i=0}^{n} i$ Testbench III

```
endmodule;
```


Method 2: Control controls the States

$\sum_{i=0}^{n} i$ Data Path (What **can** be done)



Data path I

Listing 7: demo/03_sum/sumDP.v

```
‘timescale 1ns/1ns

module sumDP #(parameter W=1) (
    output [W-1:0]out,
    output iLEn,
    input  t2,
    input  t1,
    input  t0,
    input  [W-1:0]n,
    input  clk
    );

    wire [W-1:0]sumOut;
```

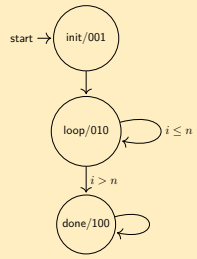

Data path II

```
    wire [W-1:0]sumPi;
    register #(W) sum(sumOut,sumPi,1'b0,
                      1'b0,1'b0,
                      1'b0,1'b0,
                      tmp,t0,
                      clk);
    assign out = sumOut;

    wire [W-1:0]iOut;
    register #(W) i(iOut,W'(0),1'b0,
                           1'b0,1'b0,
                           1'b0,tmp,
                           t0,1'b0,
                           clk);
```


Data path III

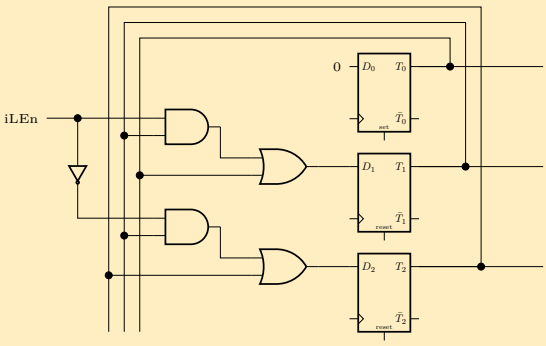
```
    binaryAdder #(W) ba(,,sumPi,sumOut,
                               iOut,1'b0);

    comparator #(W) com(lt,eq,,iOut,n);
    orGate o(iLEn,eq,lt);

    wire tmp;
    andGate incAdd(tmp,t1,iLEn);
    assign iInc = tmp;
    assign sumAdd = tmp;
endmodule
```

$\sum_{i=0}^{n} i$ State Diagram



This is a **Moore** machine

Control (circuit)



Control I

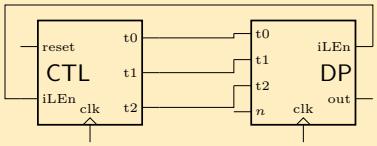
Listing 8: demo/03_sum/ctl.v

```
‘timescale 1ns/1ns

module sumCTL (
    output t2,
    output t1,
    output t0,
    input  iLEn,
    input  reset,
    input  clk
    );

    wire d0,d1,d2;
    dffi dff0(t0,,d0,reset,1’b0,clk);
```


Control II

```
    dffi dff1(t1,,d1,1’b0,reset,clk);
    dffi dff2(t2,,d2,1’b0,reset,clk);

    assign d0=0;

    wire iLEnn,t1LE,t1LEn;
    notGate ciLEnn(iLEnn,iLEn);
    andGate ct1LE(t1LE,t1,iLEn);
    andGate ct1LEn(t1LEn,t1,iLEnn);

    orGate cd1(d1,t0,t1LE);
    orGate cd2(d2,t2,t1LEn);
endmodule
```

$\sum_{i=0}^{n} i$ Circuit



$\sum_{i=0}^{n} i$ Verilog I

Listing 9: sum.v

```
‘timescale 1ns/1ns

module sum #(parameter W=1) (
    output [W-1:0]out,
    input  [W-1:0]n,
    input  reset,
    input  clk
    );

    wire iLEn;
    wire t2,t1,t0;

    sumCTL  ctl(t2,t1,t0,iLEn,reset,clk);
```


$\sum_{i=0}^{n} i$ Verilog II

```
    sumDP #(W) dp(out,iLEn,t2,t1,t0,n,clk);
endmodule
```


$\sum_{i=0}^{n} i$ Testbench I

Listing 10: sum_tb.v

```
‘timescale 1ns/1ns

module sum_tb;
    parameter W = 24;
    parameter N = 1000;

    clock #(200) cclk(clk);

    logic reset;
    wire [W-1:0]value;
    wire iLTn;

    sum #(W) sum(value,W’(N),reset,clk);
```

$\sum_{i=0}^{n} i$ Testbench II

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

        reset=1;
        #300ns;
        reset=0;

        #1000000ns;

        $display(value);
        $finish();
    end
endmodule;
```


$\sum_{i=0}^{n} i$ Testbench III

RTL
©C.M.
$\sum_{i=0}^{n} i$ comb.
$\sum_{i=0}^{n} i$ program
Method 1
Method 2
RTL examples
$\sum_{i=0}^{n} i$ rtl
InvokeSum
Fibonacci

A Higher Level Notation

RTL
©C.M.
$\sum_{i=0}^{n} i$ comb.
$\sum_{i=0}^{n} i$ program
Method 1
Method 2
RTL examples
$\sum_{i=0}^{n} i$ rtl
InvokeSum
Fibonacci

Register Transfer Level

- Up until now we used Verilog at the Gate Level
- There are higher levels of description, e.g., RTL
- Verilog can be used also for RTL
- However, it is too abstract for the course needs
- So we will use an easy variation

Several examples will do

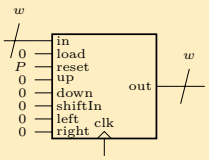
RTL
©C.M.
$\sum_{i=0}^{n} i$ comb.
$\sum_{i=0}^{n} i$ program
Method 1
Method 2
RTL examples
$\sum_{i=0}^{n} i$ rtl
InvokeSum
Fibonacci

Example 1

rtl

```
P: A ← 0
```

Drawing



Verilog

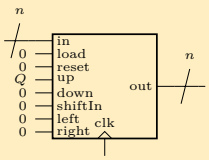
```
register #(N) A(,,1'b0,
                1'b0,1'b0,
                1'b0,1'b0,
                1'b0,P,);
```


Example 2

rtl

```
Q: A ← A + 1
```

Drawing



Verilog

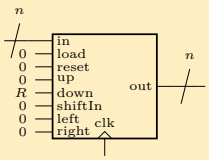
```
register #(N) A(,,1'b0,
                1'b0,1'b0,
                1'b0,Q,
                1'b0,1'b0,);
```


Example 3

rtl

```
R: A ← A - 1
```

Drawing



Verilog

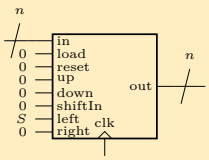
```
register #(N) A(,{N{1'b0}},1'b0,
                1'b0,1'b0,
                R,1'b0,
                1'b0,1'b0,);
```


Example 4

rtl

```
S: A ← A << 1
```

Drawing



Verilog

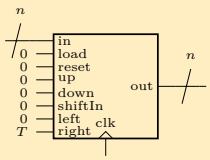
```
register #(N) A(,N'(0),1'b0,
                1'b0,S,
                1'b0,1'b0,
                1'b0,1'b0,);
```

Example 5

rtl

```
T: A ← A >> 1
```

Drawing



Verilog

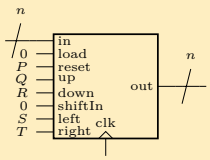
```
register #(N) A(,{N{1'b0}},1'b0,
                T,1'b0,
                1'b0,1'b0,
                1'b0,1'b0,);
```


Example 6

rtl

```
P: A ← 0
Q: A ← A + 1
R: A ← A − 1
S: A ← A << 1
T: A ← A >> 1
```

Drawing



Verilog

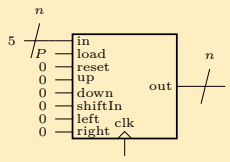
```
register #(N) A(,N'(0),1'b0,
                T,S,
                R,Q,
                1'b0,P,);
```


Example 7

rtl

```
P: A ← 5
```

Drawing



Verilog

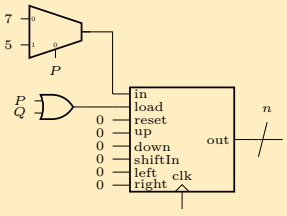
```
register #(N) A(,N'(5),1'b0,
                1'b0,1'b0,
                1'b0,1'b0,
                P,1'b0,);
```


Example 8 rtl, drawing

rtl

```
P: A ← 5
Q: A ← 7
```

Drawing



Example 8 verilog

Verilog

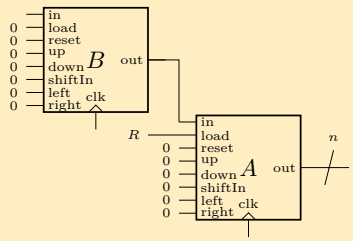
```
assign options[0] = N'(d7);
assign options[1] = N'(d5);
muxMulti #(1,N) m(d,options,P);
orGate o(PoQ,P,Q);
register #(N) A(,d,1'b0,
                        1'b0,1'b0,
                        1'b0,1'b0,
                        PoQ,1'b0,);
```


Example 9 rtl, drawing

rtl

R: A ← B

Drawing



Example 9 verilog

Verilog

```
register #(N) A(,d,1'b0,
                        1'b0,1'b0,
                        1'b0,1'b0,
                        R,1'b0,);
register #(N) B(d,,1'b0,
                        1'b0,1'b0,
                        1'b0,1'b0,
                        1'b0,1'b0,);
```


Example 10 rtl, drawing

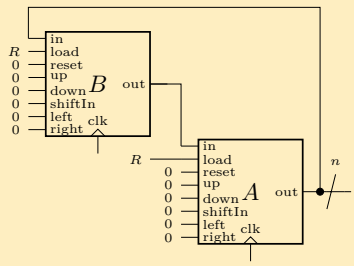
rtl

R: A ← B, B← A

rtl

R: A ← B
R: B ← A

Drawing



Example 10 verilog

Verilog

```
register #(N) A(e,d,1'b0,
                        1'b0,1'b0,
                        1'b0,1'b0,
                        R,1'b0,);
register #(N) B(d,e,1'b0,
                        1'b0,1'b0,
                        1'b0,1'b0,
                        R,1'b0,);
```


$\sum_{i=0}^{n} i$ RTL

When	What
reset :	$T_0 \leftarrow 1, T_1 \leftarrow 0, T_2 \leftarrow 0$
T_0 :	$\text{sum} \leftarrow 0, \text{seq} \leftarrow 1$
T_0 :	$i \leftarrow 0$
$T_1(i \leq n)$:	$i++, \text{sum} \leftarrow \text{sum} + i, \text{seq} \leftarrow 1$
$T_1(i > n)$:	$\text{seq} \leftarrow 2$
T_2 :	$\text{seq} \leftarrow 2$

When	What
T_2 :	$\text{seq} \leftarrow 2$
$T_1(i \leq n)$:	$i++, \text{sum} \leftarrow \text{sum} + i, \text{seq} \leftarrow 1$
T_0 :	$\text{sum} \leftarrow 0, \text{seq} \leftarrow 1$
$T_1(i > n)$:	$\text{seq} \leftarrow 2$
T_0 :	$i \leftarrow 0$
reset :	$T_0 \leftarrow 1, T_1 \leftarrow 0, T_2 \leftarrow 0$

So?

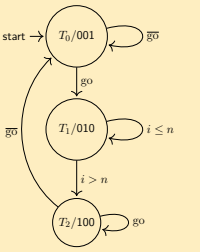
- We need to count cycles?
- How else the initiator knows when the sum is correct?
- A protocol is called for

(Carmi) Lecture 22 reached here

$\sum_{i=0}^{n} i$ RTL

reset :	$\text{seq} \leftarrow 0$
T_0 :	$\text{sum} \leftarrow 0, i \leftarrow 0, \text{done} \leftarrow 0$
$T_0\overline{\text{go}}$:	$\text{seq} \leftarrow 0$
$T_0\text{go}$:	$\text{seq} \leftarrow 1$
$T_1(i \leq n)$:	$i++, \text{sum} \leftarrow \text{sum} + i, \text{seq} \leftarrow 1$
$T_1(i > n)$:	$\text{done} \leftarrow 1, \text{seq} \leftarrow 2$
$T_2\text{go}$:	$\text{seq} \leftarrow 2$
$T_2\overline{\text{go}}$:	$\text{done} \leftarrow 0, \text{seq} \leftarrow 0$

$\sum_{i=0}^{n} i$ State Diagram



This is a Moore machine

- Method 1 data path is used
- Another method is used for the control: Sequence

$\sum_{i=0}^{n} i$ Control I

Listing 11: demo/04_sum/sumCTL.v

```
‘timescale 1ns/1ns

module sumCTL (
    output done,
    output iClear,
    output iInc,
    output sumClear,
    output sumAdd,
    input  iLEn,
    input  go,
    input  reset,
    input  clk
    );
```


$\sum_{i=0}^{n} i$ Control II

```
    wire [1:0]seqReg,seqIn;
    wire seqUp,seqLoad,seqReset;
    register #(2) seq(seqReg,seqIn,1'b0,
                      1'b0,1'b0,
                      1'b0,seqUp,
                      seqLoad,seqReset,
                      clk);

    assign seqLoad=0;
    or3Gate cseqReset(seqReset,reset,t0gon,t2gon);
    orGate cseqUp(seqUp,t0go,t1GT);

    wire [3:0]t;
```

$\sum_{i=0}^{n} i$ Control III

```
    decoder #(2) timing(t,seqReg);

    wire gon;
    notGate cgon(gon,go);

    wire t0go,t0gon;
    andGate ct0go(t0go,t[0],go);
    andGate ct0gon(t0gon,t[0],gon);

    wire t2go,t2gon;
    andGate ct2go(t2go,t[2],go);
    andGate ct2gon(t2gon,t[2],gon);

    wire iGTn,t1LE,t1GT;
    notGate ciLEnn(iGTn,iLEn);
```


$\sum_{i=0}^{n} i$ Control IV

```
    andGate ct1LE(t1LE,t[1],iLEn);
    andGate ct1LEn(t1GT,t[1],iGTn);

    assign iClear = t[0];
    assign sumClear = t[0];
    orGate cdone(done,t1GT,t2go);

    assign iInc = t1LE;
    assign sumAdd = t1LE;
endmodule
```


$\sum_{i=0}^{n} i$ verilog I

Listing 12: sum.v

```
`timescale 1ns/1ns

module sum #(parameter W=1) (
    output done,
    output [W-1:0]out,
    input  [W-1:0]n,
    input  go,
    input  reset,
    input  clk
    );

    wire sumClear,sumAdd,iClear,iInc;
    wire iLEn;
```


$\sum_{i=0}^{n} i$ verilog II

```
    sumCTL  ctl(done,iClear,iInc,sumClear,sumAdd,
                iLEn,go,reset,clk);
    sumDP #(W) dp(out,iLEn,iClear,iInc,
                sumClear,sumAdd,n,
                clk);
endmodule
```

$\sum_{i=0}^{n} i$ Testbench I

Listing 13: sum_tb.v

```
‘timescale 1ns/1ns

module sum_tb;
    parameter W = 24;
    parameter N = 1000;

    clock #(200) cclk(clk);

    logic reset,go;
    wire [W-1:0]value;

    sum #(W) sum(done,value,W’(N),go,reset,clk);
```


$\sum_{i=0}^{n} i$ Testbench II

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

        reset=1;
        #150ns;
        reset=0;
        go=1;

        #1000000ns;

        $display(value);
        $finish();
    end
```


$\sum_{i=0}^{n} i$ Testbench III

```
    always @(posedge clk) begin
        if (done == 1)
            go = 0;
    end

endmodule;
```

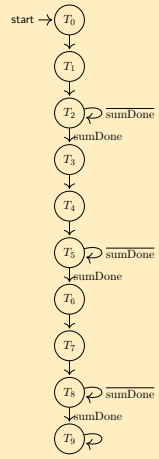

Invoking sum

Invoking sum rtl

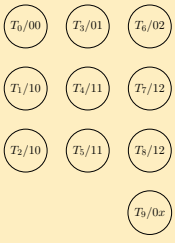
$$
\begin{array}{ll}
\text{reset}: & \text{seq} \leftarrow 0 \\
T_0: & \text{sumMux} \leftarrow 0 \\
T_1: & \text{sumGo} \leftarrow 1 \\
T_2 \overline{\text{sumDone}}: & \text{seq} \leftarrow 2 \\
T_3: & \text{sumGo} \leftarrow 0, \text{sumMux} \leftarrow 1 \\
T_4: & \text{sumGo} \leftarrow 1 \\
T_5 \overline{\text{sumDone}}: & \text{seq} \leftarrow 5 \\
T_6: & \text{sumGo} \leftarrow 0, \text{sumMux} \leftarrow 2 \\
T_7: & \text{sumGo} \leftarrow 1 \\
T_8 \overline{\text{sumDone}}: & \text{seq} \leftarrow 8 \\
T_9: & \text{sumGo} \leftarrow 0, \text{seq} \leftarrow 9
\end{array}
$$

- Default behavior of seq: $\text{seq}++$
- Output hold until explicit change

Invoke State Diagram (not really needed)



Invokesum Outputs (not really needed)



invokeSum Data Path I

Listing 14: invokeSumDP.v

```
‘timescale 1ns/1ns

module invokeSumDP #(parameter W=1) (
    output sumDone,
    input  [1:0]sumMux,
    input  sumGo,
    input  reset,
    input  clk
    );

    wire [W-1:0]options[4];
    assign options[0] = 10;
    assign options[1] = 20;
```

invokeSum Data Path II

```
    assign options[2] = 30;
    assign options[3] = 40;

    wire [W-1:0]n;
    muxMulti #(2,W) mux(n,options,sumMux);

    wire [W-1:0] sumAnswer;
    sum #(W) sum(sumDone,sumAnswer,n,sumGo,
                 reset,clk);
endmodule
```


Invoke Control I

Listing 15: demo/05_invokeSum/invokeSumCTL.v

```
`timescale 1ns/1ns

module invokeSumCTL #(parameter W=1) (
    output [1:0]sumMux,
    output sumGo,
    input  sumDone,
    input  reset,
    input  clk
);
    wire [3:0]seqReg;
    wire seqUp;
    register #(4) seq(seqReg,,,
                      1'b0,1'b0,
```


Invoke Control II

```
                      1'b0,seqUp,
                      1'b0,reset,clk);

    wire sumNDone;
    notGate csumNDone(sumNDone,sumDone);
    wire t2sumNDone,t5sumNDone,t8sumNDone;
    andGate ct2sumNDone(t2sumNDone, t[2],
                                    sumNDone);
    andGate ct5sumNDone(t5sumNDone, t[5],
                                    sumNDone);
    andGate ct8sumNDone(t8sumNDone, t[8],
                                    sumNDone);
    nor4Gate cseqNUp(seqUp,t2sumNDone,
            t5sumNDone,t8sumNDone,t[9]);
```


Invoke Control III

```
    wire [15:0]t;
    decoder #(4) timing(t,seqReg);

    wire sumGo;
    nor4Gate csumGo(sumGo,t[0],t[3],t[6],
                          t[9]);

    wire sumMuxUp;
    register #(2) csumMux(sumMux,,,
                          1'b0,1'b0,
                          1'b0,sumMuxUp,
                          1'b0,t[0],clk);
    orGate csumMuxUp(sumMuxUp,t[3],t[6]);
endmodule
```

Invoke Control IV

Invoke verilog I

Listing 16: invokeSum.v

```
‘timescale 1ns/1ns

module invokeSum #(parameter W=1) (
    input  reset,
    input  clk
);
    wire [1:0]sumMux;

    invokeSumCTL #(W) invokeSumCTL(
            sumMux,
            sumGo,
            sumDone,
            reset,
```


Invoke verilog II

```
            clk);

    invokeSumDP #(W) invokeSumDP(
            sumDone,
            sumMux,
            sumGo,
            reset,
            clk);
endmodule
```


Invoke Testbench I

Listing 17: invokeSum_tb.v

```
‘timescale 1ns/1ns

module invokeSum_tb;
    parameter W = 24;

    logic reset;
    wire clk;

    clock #(400) cclk(clk);

    invokeSum #(W) invokeSum(reset,clk);

    initial begin
```

Invoke Testbench II

```
        $dumpfile("invokeSum.vcd");
        $dumpvars;

        reset=1;
        #250ns;
        reset=0;

        #1000000ns;

        $finish();
    end

endmodule;
```


Fibbonaci Sequence

Fibonacci rtl

reset :	$\text{seq} \leftarrow 0$
T_0 :	$\text{done} \leftarrow 0, b \leftarrow 0, c \leftarrow 1, i \leftarrow 0$
$T_1\overline{\text{go}}$:	$\text{seq} \leftarrow 1$
$T_1\text{go}$:	$\text{seq} \leftarrow 2$
$T_2(i < n)$:	$i++, b \leftarrow c, c \leftarrow b + c, \text{seq} \leftarrow 2$
$T_2(i \geq n)$:	$\text{done} \leftarrow 1, \text{seq} \leftarrow 3$
$T_3\text{go}$:	$\text{seq} \leftarrow 3$
$T_3\overline{\text{go}}$:	$b \leftarrow 0, c \leftarrow 1, i \leftarrow 0, \text{done} \leftarrow 0, \text{seq} \leftarrow 1$

- done - wire
- b,c,i: Registers

Fibonacci Data Path I

Listing 18: demo/06_fibo/fiboDP.v

```
`timescale 1ns/1ns

module fiboDP #(parameter W=1) (
    output [W-1:0]out,
    output iLTn,
    input  [W-1:0]n,
    input  iUp,
    input  iReset,
    input  cMux,
    input  cLoad,
    input  bLoad,
    input  bReset,
    input  clk
```

Fibonacci Data Path II

```
    );
    wire [W-1:0]bReg;
    register #(W) b(bReg,regC,1'b0,
                    1'b0,1'b0,
                    1'b0,1'b0,
                    bLoad,bReset,
                    clk);

    wire [W-1:0]optC [2], inC;
    assign optC[0] = W'(1);
    assign optC[1] = sumBC;
    muxMulti #(1,W) muxC(inC,optC,cMux);

    wire [W-1:0]regC;
    register #(W) c(regC,inC,1'b0,
```


Fibonacci Data Path III

```
                    1'b0,1'b0,
                    1'b0,1'b0,
                    cLoad,1'b0,
                    clk);


    wire [W-1:0]iReg;
    register #(W) i(iReg,W'(0),1'b0,
                        1'b0,1'b0,
                        1'b0,iUp,
                        1'b0,iReset,
                        clk);

    wire [W-1:0]sumBC;
    binaryAdder #(W) ba(,,sumBC,bReg,
```


Fibonacci Data Path IV

```
                        regC,1'b0);

    comparator #(W) com(iLTn,,,iReg,n);

    assign out=bReg;

endmodule
```


Fibonacci Control I

Listing 19: demo/06_fibo/fiboCTL.v

```
`timescale 1ns/1ns

module fiboCTL (
    output  iUp,
    output  iReset,
    output  cMux,
    output  cLoad,
    output  bLoad,
    output  bReset,
    output done,
    input  iLTn,
    input  go,
    input  reset,
```

Fibonacci Control II

```
    input  clk
    );

    wire [1:0]seqReg, seqIn;
    wire seqUp, seqLoad;
    register #(2) seq(seqReg,2'(1),1'b0,
                      1'b0,1'b0,
                      1'b0,seqUp,
                      seqLoad,reset,
                      clk);

    comparator #(2) cmp(,eq,,seqReg,2'(3));
    andGate cseqLoad(seqLoad,eq,t3gon);

    or3Gate cseqUp(seqUp,t[0],t1go,t2LTn);
```


Fibonacci Control III

```
    wire [3:0]t;
    decoder #(2) timing(t,seqReg);

    andGate cdone(done,t[2],iLTnn);

    orGate cbRreset(bReset,t[0],t3gon);
    andGate cbLoad(bLoad,t[2],iLTn);

    or3Gate ccLoad(cLoad,t[0],t2LT,t3gon);
    assign cMux = t2LT;

    orGate ciReset(iReset,t[0],t3gon);

    andGate ciUp(iUp,t[2],iLTn);
```


Fibonacci Control IV

```
    wire gon;
    notGate cgon(gon,go);

    wire t1gon,t3gon;
    andGate ct1gon(t1gon,t[1],gon);

    wire t1go;
    andGate c1go(t1go,t[1],go);

    wire iLTnn,t1LT,t1LTn;
    notGate ciLTnn(iLTnn,iLTn);
    andGate ct1LT(t2LT,t[2],iLTn);
    andGate ct1LTn(t2LTn,t[2],iLTnn);
```


Fibonacci Control V

```
    wire t3go;
    andGate ct3go(t3go,t[3],go);
    andGate ct3gon(t3gon,t[3],gon);


endmodule
```

Fibonacci verilog I

Listing 20: demo/06_fibo/fibo.v

```
`timescale 1ns/1ns

module fibo #(parameter W=1) (
    output [W-1:0]out,
    output done,
    input  [W-1:0]n,
    input  go,
    input  reset,
    input  clk
    );

    wire iLTn;
    wire t2,t1,t0;
```


Fibonacci verilog II

```
    fiboCTL  ctl(iUp,iReset,cMux,cLoad,bUp,bReset,
                 done,iLTn,go,reset,clk);
    fiboDP #(W) dp(out,iLTn,n,
                 iUp,iReset,cMux,cLoad,bUp,bReset,
                 clk);
endmodule
```


Fibonacci Testbench I

Listing 21: demo/06_fibo/fibo_tb.v

```
`timescale 1ns/1ns

module fibo_tb;
    parameter W = 24;
    parameter N = 10;

    clock #(200) cclk(clk);

    logic reset,go;
    wire [W-1:0]value;
    wire iLTn;

    wire done;
```


Fibonacci Testbench II

```
    fibo #(W) fibo(value,done,W'(N),go,reset,clk);

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

        reset=1;
        go = 0;
        #150ns;
        reset=0;
        go = 1;

        #1000000ns;
```

Fibonacci Testbench III

```
                $finish();
    end

    always @(posedge clk) begin
        if (done == 1)
            go = 0;
    end

endmodule;
```