ECE426/516 Lab 3

Professor: Chao Lu
Email: chaolu@siu.edu

Lab 3 Verilog for 4-bit adder design

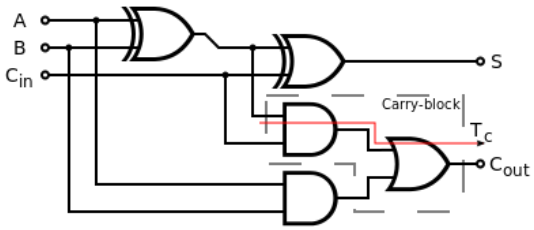
We have learned ripple carry adder and carry lookahead adder in ECE 327. A ripple-carry adder is a logic circuit in which the carry-out of each full adder is the carry in of the succeeding next most significant full adder. It is called a ripple carry adder, because each carry bit gets rippled into the next stage. A carry-lookahead adder is a fast adder, which improves speed by reducing the amount of time required to determine carry bits. In this lab, we will implement both of them in Verilog to compare their area and speed.

1. Constructing ripple carry adder using full adders.

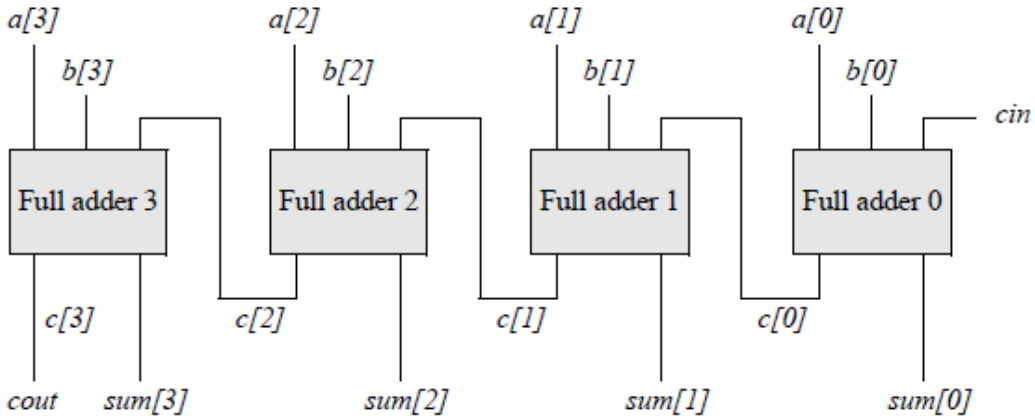
1-bit full adder Truth table

A	B	C_{IN}	C_{OUT}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S = A \oplus B \oplus C_{IN}$, $C_{OUT} = AB + AC_{IN} + BC_{IN} = AB + C_{IN}(A \oplus B)$



Ripple Carry Adder: We can make an n-bit adder by using n full adders, Each Cout is connected to Cin for the next stage. In this way carries can "ripple" through the adder stage at a time.



```
//behavioral full adder
module full_adder_bh (a, b, cin, sum, cout);
//define inputs and outputs
input a, b, cin;
output sum, cout;
//inputs are wire by default, but can be declared if so desired
wire a, b, cin;
reg sum, cout; //target variables in always are type reg
always @ (a or b or cin)
begin
sum = a ^ b ^ cin;
cout = (a & b) | (a & cin) | (b & cin);
end
endmodule

//structural 4-bit ripple adder
module adder_ripple4_struc (a, b, cin, sum, cout);
input [3:0] a, b;
input cin;
output [3:0] sum;
output cout;
wire [3:0] a, b;
wire cin;
wire [3:0] sum;
wire [3:0] c; //define internal nets for carries
wire cout;
assign cout = c[3];
full_adder_bh inst1 (
.a(a[0]),
.b(b[0]),
.cin(cin),
.sum(sum[0]),
.cout(c[0])
);
full_adder_bh inst2 (
.a(a[1]),
.b(b[1]),
.cin(c[0]),
.sum(sum[1]),
.cout(c[1])
);
full_adder_bh inst3 (
.a(a[2]),
.b(b[2]),
.cin(c[1]),
.sum(sum[2]),
.cout(c[2])
);
full_adder_bh inst4 (
.a(a[3]),
.b(b[3]),
.cin(c[2]),
.sum(sum[3]),
.cout(c[3])
);
endmodule
```

```
`timescale 1ns/1ns

`include "full_adder_bh.v"
`include "adder_ripple4_struc.v"

module testbench;
  reg [3:0]a, b;
  reg cin;
  wire [3:0] sum;
  wire cout;

  initial
  begin
  #0  a = 4'b1111; b = 4'b0000; cin = 1'b0;
  #10 a = 4'b1111; b = 4'b0001; cin = 1'b0;
  #10 a = 4'b0011; b = 4'b0111; cin = 1'b0;
  #10 a = 4'b0101; b = 4'b0101; cin = 1'b0;
  #10 a = 4'b1001; b = 4'b1001; cin = 1'b0;
  #10 a = 4'b1110; b = 4'b0001; cin = 1'b0;
  #10 a = 4'b1101; b = 4'b1101; cin = 1'b0;
  #10 a = 4'b1111; b = 4'b1111; cin = 1'b0;
  #10 a = 4'b1111; b = 4'b1111; cin = 1'b0;
  #10 a = 4'b1110; b = 4'b0001; cin = 1'b1;
  #10 a = 4'b1101; b = 4'b1101; cin = 1'b1;
  #10 a = 4'b1110; b = 4'b1111; cin = 1'b1;
  #10 a = 4'b1111; b = 4'b1111; cin = 1'b1;

  #50 $stop;

  end

  //instantiate the module into the test bench
  adder_ripple4_struc inst1 (
  .a(a),
  .b(b),
  .cin(cin),
  .sum(sum),
  .cout(cout)
  );

initial
begin
  $monitor ($time, " hins a=%b, b=%b, cin=%b, cout=%b, sum=%b", a, b, cin, cout,
sum);
end

endmodule
```

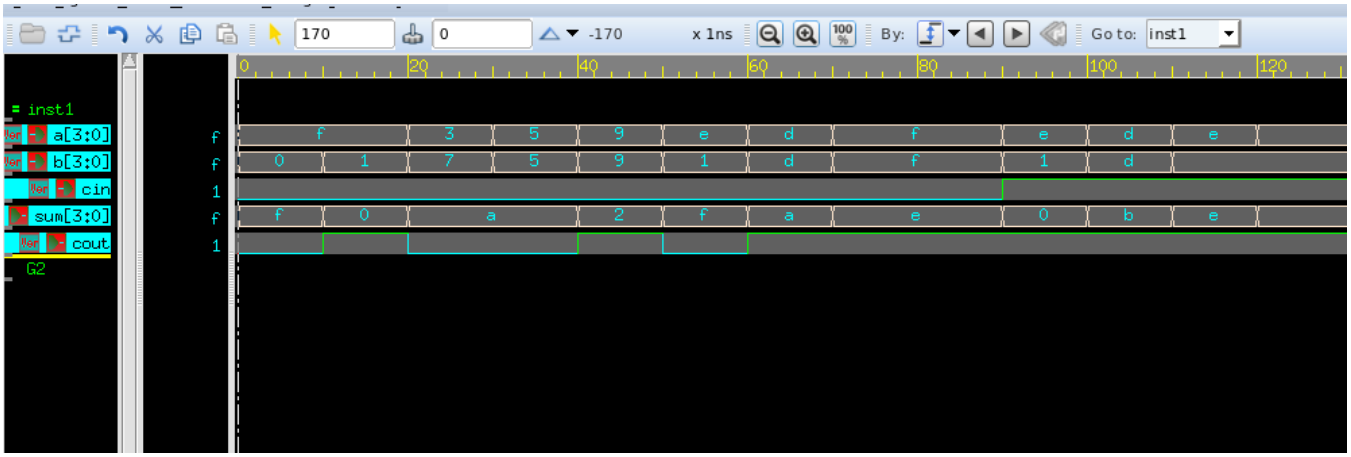
First using VCS Simulator to run simulation of your design:

```
> vlogan full_adder_bh.v
> vlogan adder_ripple4_struc.v
```

```
> vlogan testbench.v
> vcs testbench.v –debug_acc+all -debug_region+cell+encrypt -kdb
>./simv
```

```
Compiler version T-2022.06-SP1; Runtime version T-2022.06-SP1;  Feb  7 13:11 2023
                   0 ns a=1111, b=0000, cin=0, cout=0, sum=1111
                  10 ns a=1111, b=0001, cin=0, cout=1, sum=0000
                  20 ns a=0011, b=0111, cin=0, cout=0, sum=1010
                  30 ns a=0101, b=0101, cin=0, cout=0, sum=1010
                  40 ns a=1001, b=1001, cin=0, cout=1, sum=0010
                  50 ns a=1110, b=0001, cin=0, cout=0, sum=1111
                  60 ns a=1101, b=1101, cin=0, cout=1, sum=1010
                  70 ns a=1111, b=1111, cin=0, cout=1, sum=1110
                  90 ns a=1110, b=0001, cin=1, cout=1, sum=0000
                 100 ns a=1101, b=1101, cin=1, cout=1, sum=1011
                 110 ns a=1110, b=1111, cin=1, cout=1, sum=1110
                 120 ns a=1111, b=1111, cin=1, cout=1, sum=1111
$stop at time 170 Scope: testbench File: testbench.v Line: 30
ucli%
```

Run the command “./simv -gui” to view the necessary waveforms of inputs and outputs signals.



Use Design Vision to analyze your design, find the critical path.
First set up the following “run_dc_ripple.tcl” file

```
set search_path ".
/synopsys/GPDK/SAED_EDK90nm/Digital_Standard_Cell_Library/synopsys/models"
set link_library "saed90nm_max.db"
set target_library "saed90nm_max.db"
set symbol_library "saed90nm_max.db"
analyze -library WORK -format verilog
{/home/staff/eeluchao/ECE426/lab3/adder_ripple4_struc.v }
elaborate adder_ripple4_struc -library WORK
link
compile -exact_map
```

And then, run the above tcl file:

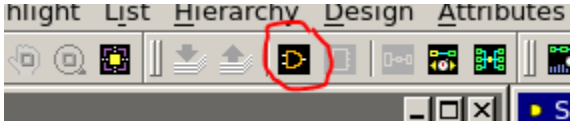
```
>dc_shell -f run_dc_structural.tcl
```

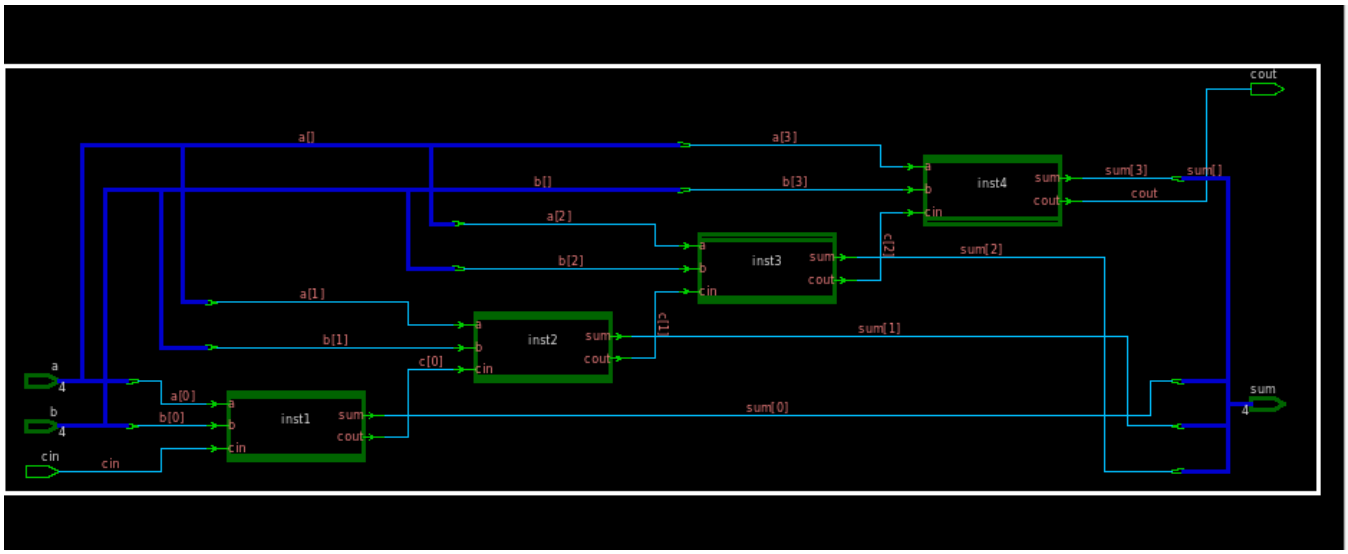
```
/home/staff/eeluchao/ECE426/lab3-238> dc_shell -f run_dc_structural.tcl

                           Design Compiler Graphical
                                 DC Ultra (TM)
                                  DFTMAX (TM)
                              Power Compiler (TM)
                                 DesignWare (R)
                                 DC Expert (TM)
                               Design Vision (TM)
                               HDL Compiler (TM)
                               VHDL Compiler (TM)
                                  DFT Compiler
                               Design Compiler(R)

                Version S-2021.06-SP5-4 for linux64 - Aug 05, 2022

                    Copyright (c) 1988 - 2022 Synopsys, Inc.
   This software and the associated documentation are proprietary to Synopsys,
 Inc. This software may only be used in accordance with the terms and conditions
 of a written license agreement with Synopsys, Inc. All other use, reproduction,
   or distribution of this software is strictly prohibited.  Licensed Products
     communicate with Synopsys servers for the purpose of providing software
    updates, detecting software piracy and verifying that customers are using
    Licensed Products in conformity with the applicable License Key for such
  Licensed Products. Synopsys will use information gathered in connection with
    this process to deliver software updates and pursue software pirates and
                                 infringers.

 Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on
            Inclusivity and Diversity" (Refer to article 000036315 at
                        https://solvnetplus.synopsys.com)
Initializing
```

```
                                  TOTAL
  ELAPSED            WORST NEG    SETUP    DESIGN
    TIME      AREA      SLACK      COST    RULE COST         ENDPOINT
  --------- --------- --------- --------- --------- -------------------------
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0


  Beginning Area-Recovery Phase  (cleanup)
  ----------------------------

                                  TOTAL
  ELAPSED            WORST NEG    SETUP    DESIGN
    TIME      AREA      SLACK      COST    RULE COST         ENDPOINT
  --------- --------- --------- --------- --------- -------------------------
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
    0:00:00       0.0      0.00       0.0       0.0
Loading db file '/synopsys/GPDK/SAED_EDK90nm/Digital_Standard_Cell_Library/synopsys/models/saed90nm_max.db'

Note: Symbol # after min delay cost means estimated hold TNS across all active scenarios


  Optimization Complete
  ---------------------
1
1
dc shell>
```

And then, we start the gui interface, and click on the red-circled button, we can see the schematic of your design.

```
dc_shell> start_gui
dc_shell> Current design is 'adder_ripple4_struc'.
dc_shell> 4.1
Current design is 'adder_ripple4_struc'.
dc_shell>
```





2. Constructing a carry lookahead adder.

1-bit full adder Truth table

A	B	C_{IN}	C_{OUT}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

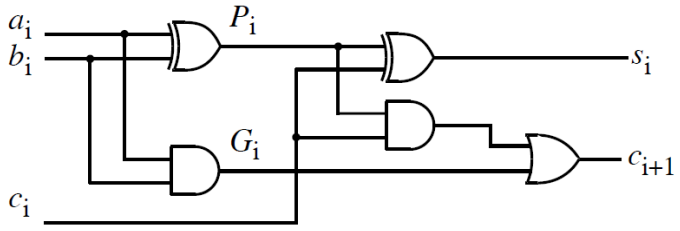
$S = A \oplus B \oplus C_{IN}$, $C_{OUT} = AB + AC_{IN} + BC_{IN} = AB + C_{IN}(A \oplus B)$

We can see from the implementation of ripple carry adder, that the critical path of the ripple carry adder is in the “carry path”. In order to minimize the delay of our adder

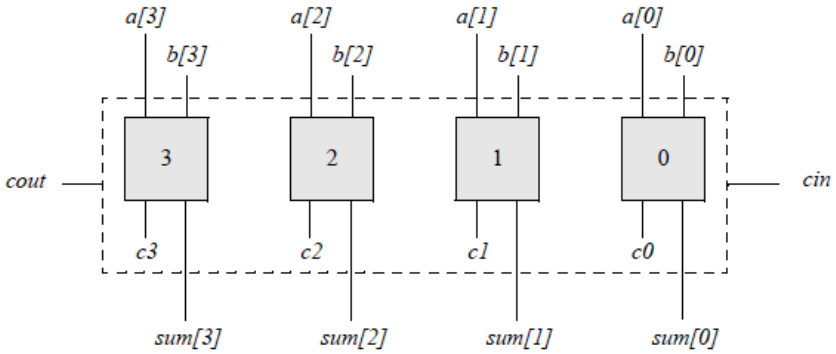
design, we will use the lookahead adder. The idea of carry lookahead adder is to calculate the carries to each stage in parallel.

$C_{out} = AB + AB'C_{in} + A'BC_{in} = AB + (AB'+A'B)\,C_{in} = G + P\,C_{in}$

$G = AB$, $P = A \oplus B$



The block diagram and port details are implemented as the following figure.



Reference: **Lecture 2 Combinational Logic Overview slides**

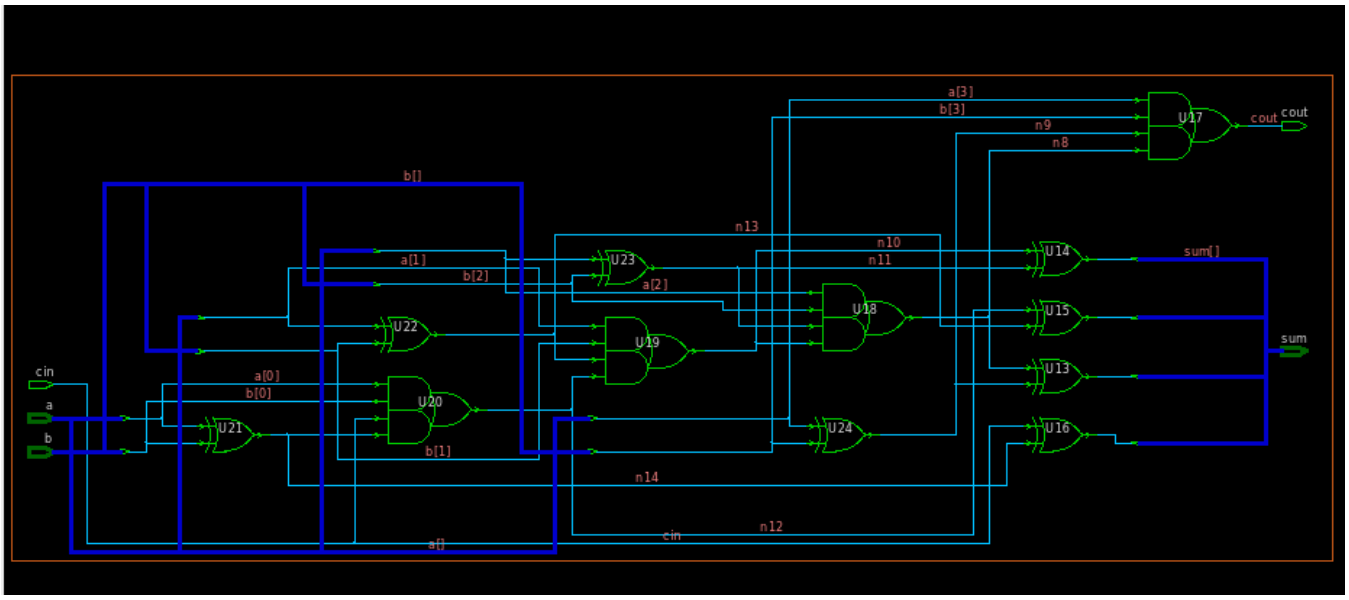
```
// a 4-bit CLA adder using assign statements

module adder_cla4 (a, b, cin, sum, cout);

input [3:0] a, b;
input cin;
output [3:0] sum;
output cout;

// internal wires
wire p0, g0, p1, g1, p2, g2, p3, g3;
wire c4, c3, c2, c1;

// compute the p for each stage
assign p0 = a[0]^b[0];
assign p1 = a[1]^b[1];
assign p2 = a[2]^b[2];
assign p3 = a[3]^b[3];

// compute the g for each stage
assign g0 = a[0]&b[0];
assign g1 = a[1]&b[1];
```

```
assign g2 = a[2]&b[2];
assign g3 = a[3]&b[3];

// compute the carrb for each stage
assign c1 = g0 | (p0 & cin);
assign c2 = g1|(p1&g0)|(p1&p0&cin);
assign c3 = g2|(p2 & g1)|(p2&p1&g0)|(p2&p1 &p0&cin);
assign c4 = g3|(p3&g2)|(p3&p2&g1)|(p3&p2& p1&g0)|(p3&p2&p1&p0&cin);

// compute Sum
assign sum[0] = p0^cin;
assign sum[1] = p1^c1;
assign sum[2] = p2^c2;
assign sum[3] = p3^c3;

// assign carrb output
assign cout = c4;

endmodule
```

```
`timescale 1ns/1ns

`include "adder_cla4.v"

module testbench;
  reg [3:0]a, b;
  reg cin;
  wire [3:0] sum;
  wire cout;

  initial
  begin
  #0  a = 4'b1111; b = 4'b0000; cin = 1'b0;
  #10 a = 4'b1111; b = 4'b0001; cin = 1'b0;
  #10 a = 4'b0011; b = 4'b0111; cin = 1'b0;
  #10 a = 4'b0101; b = 4'b0101; cin = 1'b0;
  #10 a = 4'b1001; b = 4'b1001; cin = 1'b0;
  #10 a = 4'b1110; b = 4'b0001; cin = 1'b0;
  #10 a = 4'b1101; b = 4'b1101; cin = 1'b0;
  #10 a = 4'b1111; b = 4'b1111; cin = 1'b0;
  #10 a = 4'b1111; b = 4'b1111; cin = 1'b0;
  #10 a = 4'b1110; b = 4'b0001; cin = 1'b1;
  #10 a = 4'b1101; b = 4'b1101; cin = 1'b1;
  #10 a = 4'b1110; b = 4'b1111; cin = 1'b1;
  #10 a = 4'b1111; b = 4'b1111; cin = 1'b1;

  #50 $stop;

  end

  //instantiate the module into the test bench
  adder_cla4 inst1 (
  .a(a),
  .b(b),
  .cin(cin),
  .sum(sum),
```

```
   .cout(cout)
   );

initial
begin
   $monitor ($time, " hins a=%b, b=%b, cin=%b, cout=%b, sum=%b", a, b, cin, cout,
sum);
end

endmodule
```

After synthesis, you will see the schematic below.



Demo

You should demo the following aspects of your design to TA.

1. Verilog code for 4-bit ripple carry adder and 4-bit carry lookahead adder.
2. Simulation waveforms demonstrating correct functionality of these above designs.

End of Lab Guide