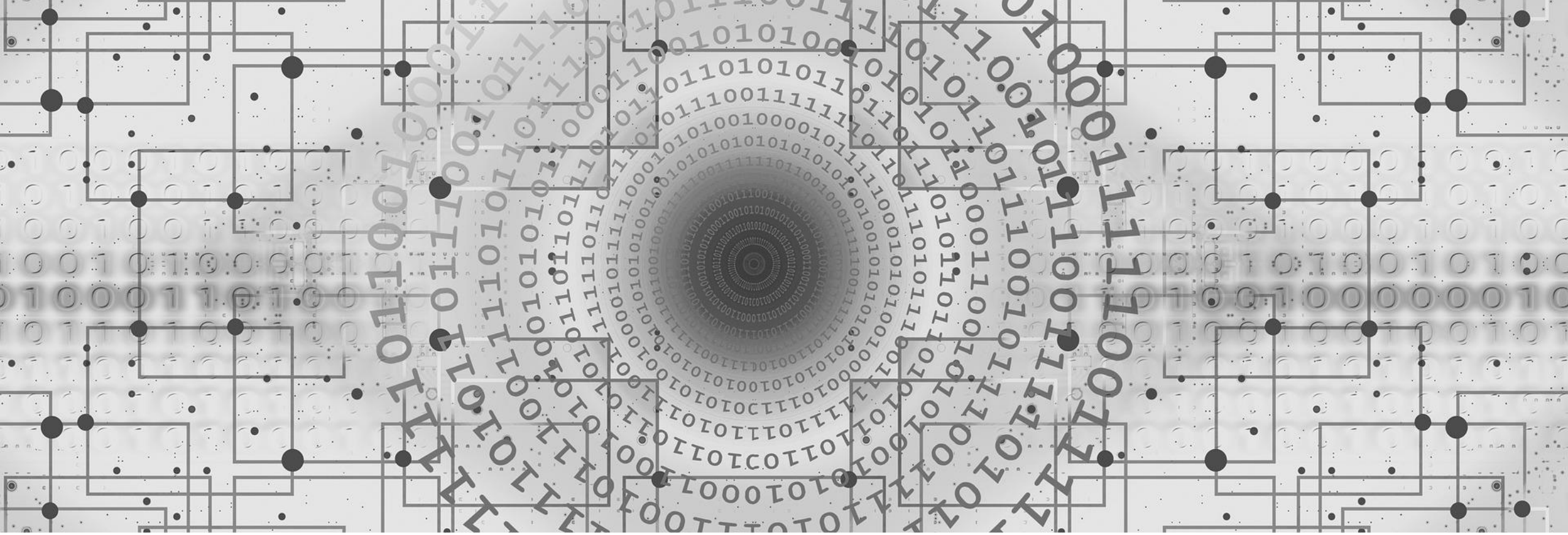


Homework-5b : Carry Look Ahead Adder

ECE-111 Advanced Digital Design Project

Vishal Karna

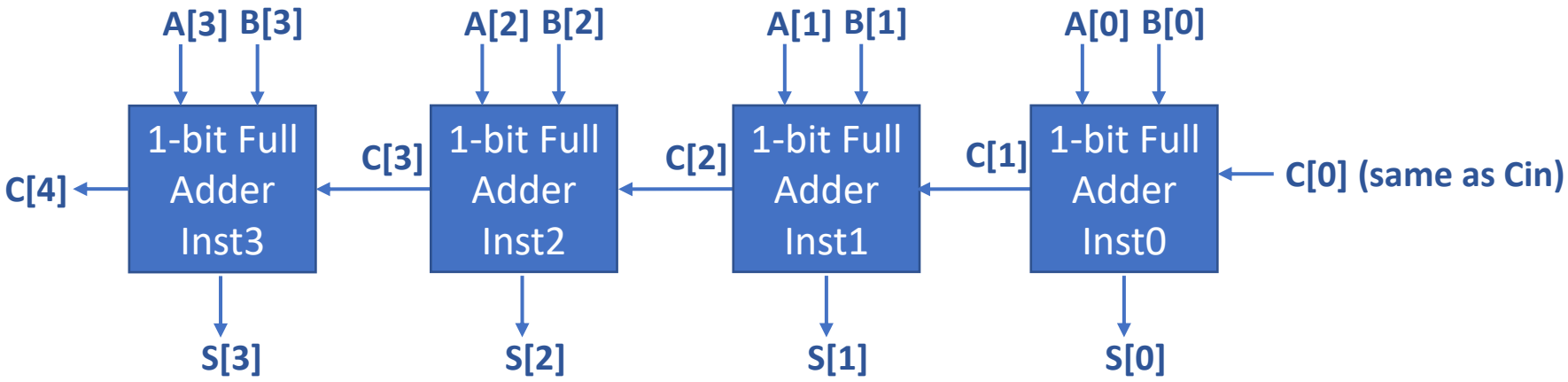
UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Outline

- **Let's first understand Ripple Carry Adder Method to add two numbers**
  - Concept, SystemVerilog Code For reference, Netlist and Simulation snapshot
- **Let's understand alternate implementation of adder : Carry Lookahead Adder**
  - Concept, Implementation details and hints
- **Homework Requirements for Carry Lookahead Adder**

Let's understand First Ripple Carry Adder !

- **A Ripple Carry Adder is made of a number of full adders cascaded together.**
  - Since carry bit from previous full adder ripples (connected) to the next full adder, hence the name Ripple carry adder
  - It is used to add together two binary numbers using simple logic gates
  - Ripple-carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder.
  - The figure below shows 4 full-adders connected together to produce a 4-bit ripple carry adder.
    - only the first full adder may be replaced by a half adder (under the assumption that C[0]= 0).



4-bit Ripple Carry Adder

Lets' Review how to develop Ripple Carry Adder using Generate Code

```
module ripple_carry_adder  #(parameter N = 4)(
  input logic[N-1:0] A, B,
  input logic CIN,
  output logic[N:0] result);
  logic[N:0] l_carry;
  logic[N-1:0] l_sum;
  // assign Carry in to first full adder carryin
  assign l_carry[0] = CIN;

  // Instantiate Full Adder for 'N' instances
  genvar i;
  generate
    for(i=0; i<N; i=i+1) begin: fa_loop
      fulladder fa_inst(
        .a(A[i]),
        .b(B[i]),
        .cin(l_carry[i]),
        .sum(l_sum[i]),
        .cout(l_carry[i+1]));
    end: fa_loop
  endgenerate

  // final result of addition and carry
  assign result = {l_carry[N], l_sum};
endmodule: ripple_carry_adder
```

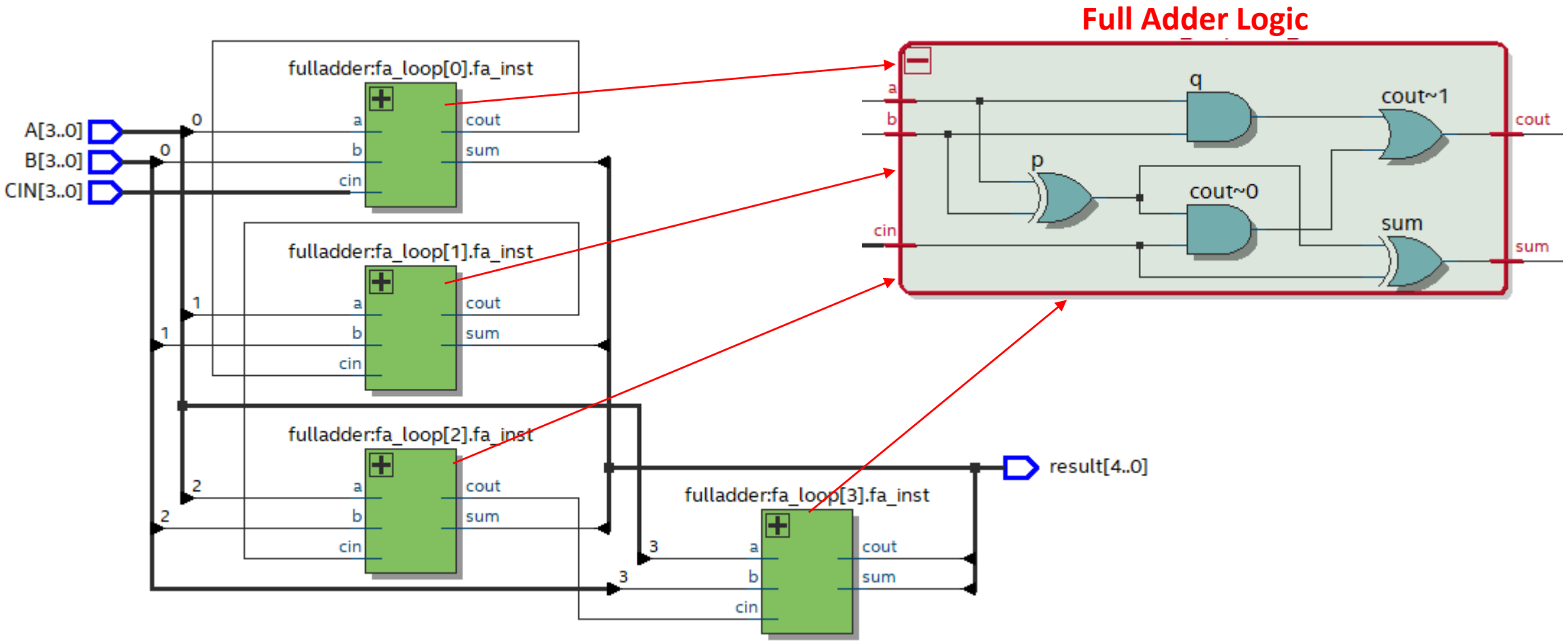
genvar 'i' controls generate for loop iteration

mandatory to have named for loop inside generate body

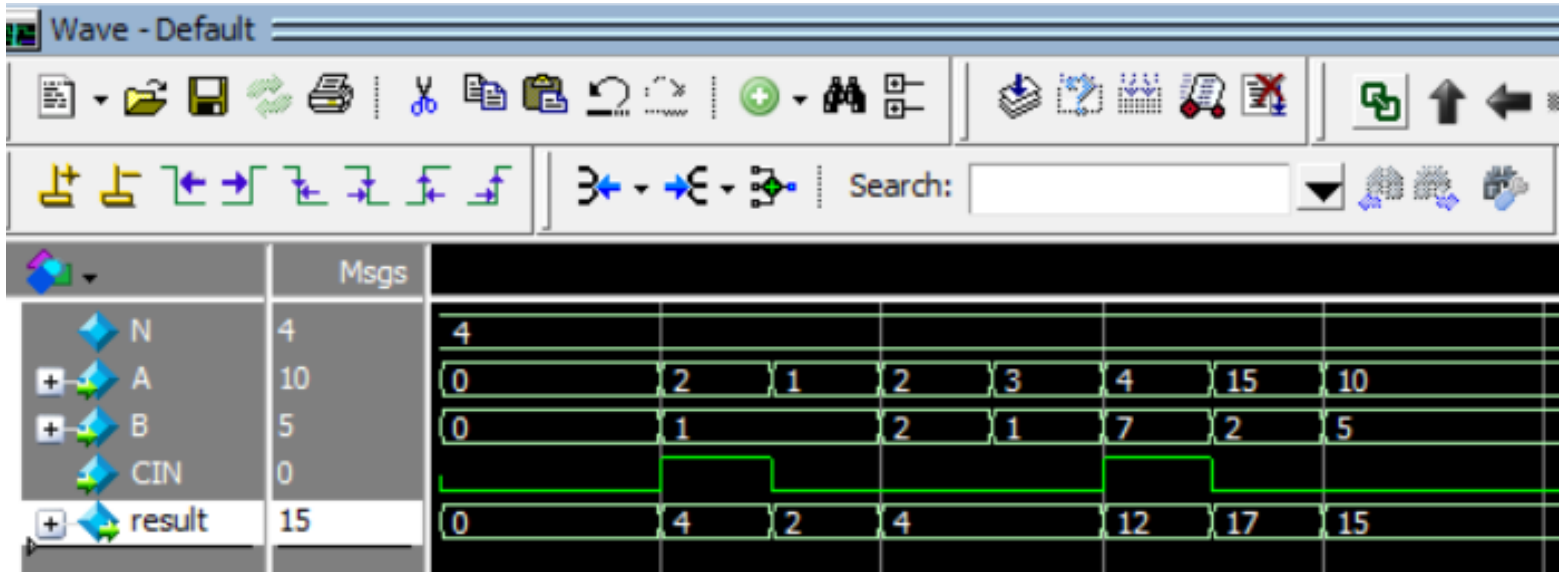
generate for loop statement will create 'N' number of full adder module instances

```
fulladder fa_inst0(
      .a(A[0]),
      .b(B[0]),
      .cin(l_carry[0]),
      .sum(l_sum[0]),
      .cout(l_carry[1]);

fulladder fa_inst1(
      .a(A[1]),
      .b(B[1]),
      .cin(l_carry[1]),
      .sum(l_sum[1]),
      .cout(l_carry[2]);

fulladder fa_inst2(
      .a(A[2]),
      .b(B[2]),
      .cin(l_carry[2]),
      .sum(l_sum[2]),
      .cout(l_carry[3]);

fulladder fa_inst3(
      .a(A[3]),
      .b(B[3]),
      .cin(l_carry[3]),
      .sum(l_sum[3]),
      .cout(l_carry[4]);
```

Ripple Carry Adder Post Synthesis Netlist



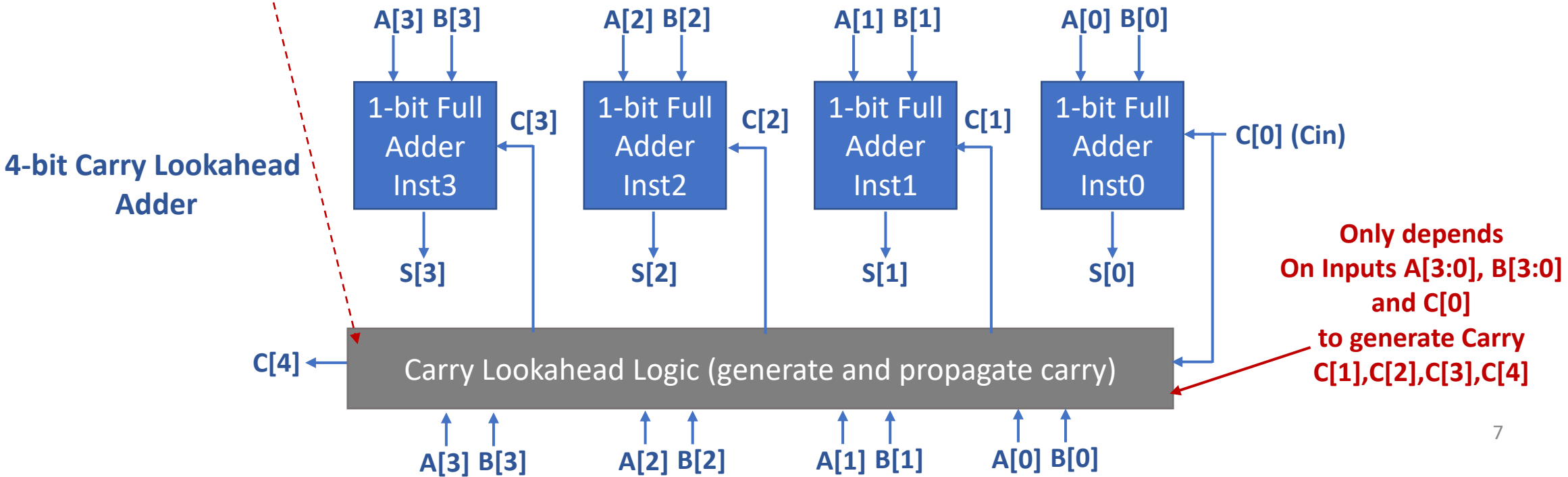
Ripple Carry Adder Simulation Snapshot



Let's Understand Alternate Adder Implementation : Carry Lookahead Adder

❑ A Carry Lookahead Adder is made of a number of full adders cascaded together.

- Carry lookahead adder is similar like ripple carry adder with the difference that it calculates the carry bit for each full adder instance before the full adder operation on inputs is performed.
- **Generate and Propagate Carry Logic** only depends on inputs A, B, very first Carry input (C[0]). These values are all known upfront.
- Since Carry input for each Full Adder instance is generated upfront, all full adder instances can concurrently perform add operation on its respective input bits and input carry
- Advantage of carry lookahead adder is that it adds two numbers faster than ripple carry adder
- The drawback is that carry lookahead adder takes more logic gates to implement. So higher in resource usage !
- If faster performance is required when adding two number then select carry lookahead adder as implementation choice
- For lower resource usage requirement, ripple carry adder can be selected as an implementation choice
- The figure below shows 4 full-adders connected together to produce a 4-bit lookahead carry adder.



Carry Lookahead Adder Hint (Carry Out Boolean Expression)

- **In Yellow marked rows in full adder truth table, Cout = 1 when either A=1 or B =1, provided Cin = 1**
  - In another words, incoming carry (Cin=1) is propagated to carry out (Cout) if one of the input's is 1 and other is 0
  - This can be represented as : **(A[i] | B[i]) & Cin[i]**
  - This is known as **propagate** carryin to carryout

inputs			outputs	
A	B	Cin	Cout	Sum
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

Full Adder Truth Table

- **In red marked rows in full adder truth table, Cout = 1 when both A = 1 and B = 1, whether Cin = 1 or 0**
  - A carry out (Cout) is generated if both A and B are 1 irrespective of the fact that previous carry (Cin) was 1 or 0
  - This can be represented as : **A[i] & B[i]**
  - This is known as **generate** carryout

inputs			outputs	
A	B	Cin	Cout	Sum
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

- **Hence final Boolean expression for Carry out (Cout) generation is combination of above mentioned :**
  - Cout[i+1] = (**(A[i] | B[i]) & Cin[i])** | **(A[i] & B[i])**

Carry Lookahead Adder Hint (Carry Out Boolean Expression)

- Carry Generate : G(i) = (A[i].B[i]) // Note : & is represented as .
- Carry Propagate : P(i) = (A[i]+B[i]) // Note : | is represented as +
- Generate and Propagate Carry : C[i+1] = G[i] + P[i].C[i]
- Carry inputs for each Full Adder Instance equations can be expanded as shown below

C[1] = G[0] + P[0].C[0]
= G[0] + P[0].Cin // Note : Replaced C[0] with Cin. C[0] is the input carry to first stage full adder

C[2] = G[1] + P[1].C[1] // Note : C[1] can be replaced with G[0] + P[0].Cin
= G[1] + P[1] (G[0] + P[0].Cin)
= G[1] + P[1].G[0] + P[1].P[0].Cin

C[3] = G[2] + P[2].C[2] // Note : C[2] can be replaced with G[1] + P[1].G[0] + P[1].P[0].Cin
= G[2] + P[2] (G[1] + P[1].G[0] + P[1].P[0].Cin)
= G[2] + P[2].G[1] + P[2].P[1].G[0] + P[2].P[1].P[0].Cin

C[4] = G[3] + P[3].C[3] // Note : C[3] can be replaced with G[2] + P[2].G[1] + P[2].P[1].G[0] + P[2].P[1].P[0].Cin
= G[3] + P[3] (G[2] + P[2].G[1] + P[2].P[1].G[0] + P[2].P[1].P[0].Cin)
= G[3] + P[3].G[2] + P[3].P[2].G[1] + P[3].P[2].P[1].G[0] + P[3].P[2].P[1].P[0].Cin

Carry Lookahead Adder Hint (Carry Out Boolean Expression)

- G[i] and P[i] can be further replaced with A[i].B[i] and A[i]+B[i] below in each of the C[1], C[2], C[3], C[4] carry equations

C[1] = G[0] + P[0].Cin

= (A[0].B[0]) + (A[0]+B[0]).Cin

C[2] = G[1] + P[1].G[0] + P[1].P[0].Cin

= (A[1].B[1]) + (A[1]+B[1]). (A[0].B[0]) + (A[1]+B[1]). (A[0]+B[0]).Cin

C[3] = G[2] + P[2].G[1] + P[2].P[1].G[0] + P[2].P[1].P[0].Cin

= (A[2].B[2]) + ((A[2]+B[2]).(A[1].B[1])) + ((A[2]+B[2]). (A[1]+B[1]).(A[0].B[0])) + ((A[2]+B[2]).(A[1]+B[1]).(A[0]+B[0]).Cin)

C[4] = G[3] + P[3].G[2] + P[3].P[2].G[1] + P[3].P[2].P[1].G[0] + P[3].P[2].P[1].P[0].Cin

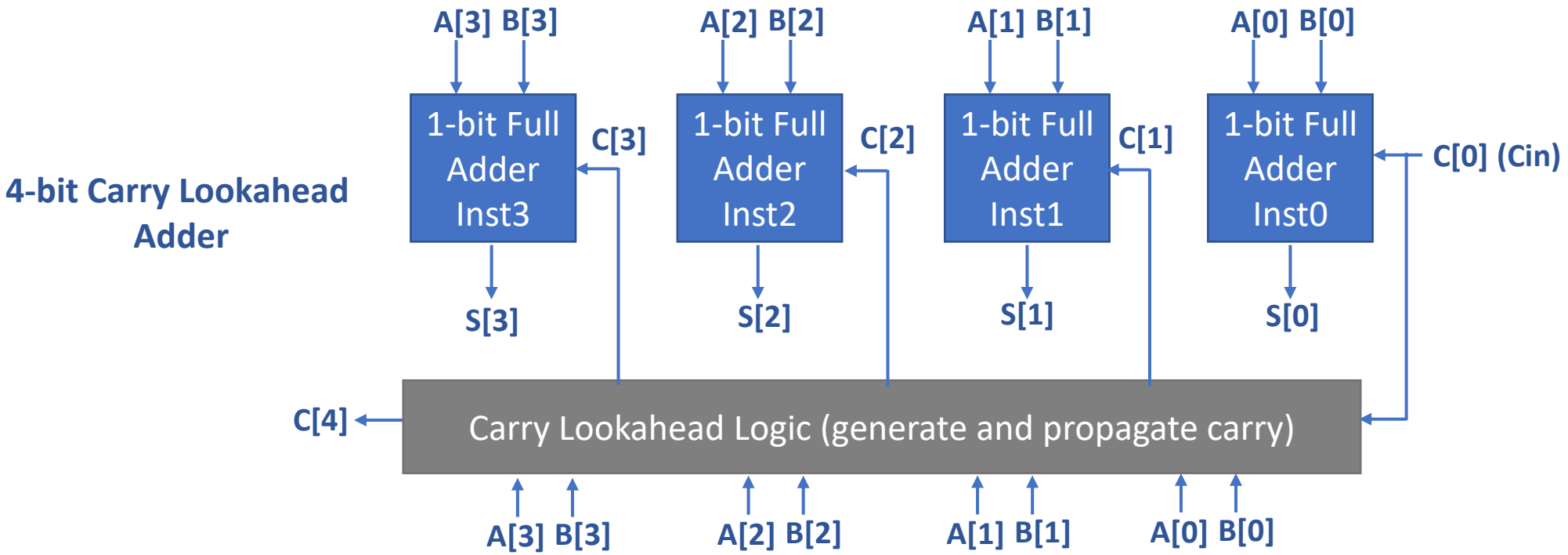
= (A[3].B[3]) + ((A[3]+B[3]).(A[2].B[2])) + ((A[3]+B[3]). (A[2]+B[2]).(A[1].B[1])) + ((A[3]+B[3]). (A[2]+B[2]).(A[1]+B[1]).(A[0].B[0]))
+ ((A[3]+B[3]). (A[2]+B[2]).(A[1]+B[1]).(A[0]+B[0]).Cin)

C[1],C[2],C[3],C[4] depends on inputs A[0], A[1], A[2], A[3], B[0], B[1], B[2], B[3] and Cin, whose values are known upfront and hence Carry inputs C[1] to C[4] can be computed right away and hence all 4 full adders can perform operation simultaneously to allow fast add operation !

Carry Lookahead Adder Implementation Hint

- **Instantiate 4 Full Adder Modules using "generate for" statement**
  - As per diagram below Cout from each full adder is not passed to next Full Adder hence in each of the full adder instance Cout can be left connected.

```
generate for i=0 to i<N (where N = 4)
  fulladder fa_inst(
      ....
      ....
      ....
      .cout());  //empty parenthesis () after cout, means unconnected cout
end: fa_loop
```



Carry Lookahead Adder Implementation Hint

- **C[1], C[2], C[3], C[4]** carryout are based on previous A and B inputs.
- **C[1], C[2], C[3], C[4]** each is created from "**generate and propagate carry**" logic.
- This can be implemented by using below mentioned Boolean expression and putting this logic in generate for statement
  - declare logic[N:0] l_carry; // N = 4
  - Using generate for where **i** is 0 to 3 create l_carry[1] to l_carry[4] which is same as C[1] to C[4] in diagram on previous page

    assign l_carry[i+1] = **((A[i] | B[i]) & Cin[i]) | (A[i] & B[i])**
- **First stage full adder Carryin C[0] is, l_carry[0] = CIN (CIN is input signal)**
- **Final out result = {l_carry[N], S[3], S[2], S[1], S[0]} // where S[0], S[1], S[2], S[3] is "sum" output from each full adder instance**

```
// assign Carry in to first full adder carryin
assign l_carry[0] = CIN;

// generate carry : G(i)=A(i).B(i)
// propogate carry : P(i)=A(i)+B(i)
// Carry out: C(i+1)=G(i)+P(i).C(i)
genvar j;
generate
  for (j=0; j<N; j=j+1)
    begin : carry_gen_loop
      ....
      ....
      ....
    end : carry_gen_loop
endgenerate

// final result is concatenation of Final stage carry c[4] and sum S[3:0]
assign result = {......};
```

// Note : . can be represented as &
// Note : + can be represented as |

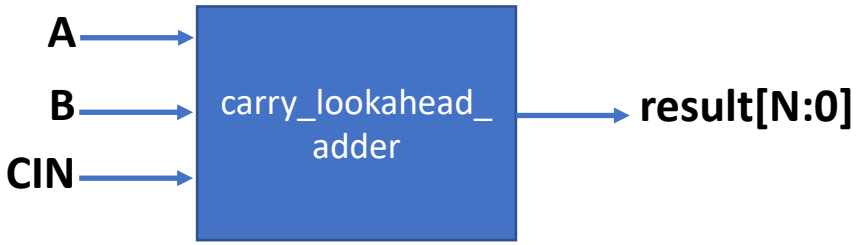
Note : continuous assign statements are allowed within generate-for begin end body

- Remember that in case of Carry Lookahead Adder, each Full Adder instance does addition of its Inputs concurrently.
- Hence Carryin for each full adder has to be made available without waiting for previous full adder addition operation completion.
- For each Full Adder Carry input is computed using previous full adder inputs and previous full adder carryin which is always available !

Homework Assignment-5b : Carry Lookahead Adder

- **Develop SystemVerilog RTL model for N-bit Carry Lookahead Adder**
  - Synthesize carry look ahead adder design and run simulation using testbench provided with N=4
  - Review synthesis results (resource usage and RTL netlist/schematic)
  - Review input and output signals in simulation waveform.
  - Assume below mentioned primary port names and SystemVerilog RTL module name **carry_lookahead_adder.**

- **Primary Ports for carry_lookahead_adder module**
  - **Input [N-1:0] A, B :** These are values to be added using carray lookahead adder logic)
  - **Input CIN :** carryin to first stage Full Adder Instance in carry lookahead adder
  - **Output result :** final output addition result and final stage carryout)
    - i.e. result of A[N-1:0] + B[N-1:0] + CIN)
    - **Note :** Output signal **result** MSB bit includes final stage fulladder carryout



Homework Assignment-5b : Carry Lookahead Adder

- **Report should include :**
  - SystemVerilog design code
  - Synthesis resource usage and schematic generated from RTL netlist viewer
  - Simulation snapshot and explain simulation result to confirm RTL model developed works as carry lookahead adder
  - Post-Mapping schematic is optional to submit.
  - Explanation of FPGA resource usage in report is not required.

- **Lab5 folder includes :**
  - Full adder design code (fulladder.sv) which is required when developing carry lookahead adder design module. Add fulladder.sv file in Quartus and Modelsim for compilation.
  - Template design file : carry_lookahead_adder.sv
  - Full testbench code for carry look ahead adder. For learning purpose, student can change the stimulus in initial block in testbench file.

Carry Lookahead Adder Simulation Snapshot For Reference