Adders

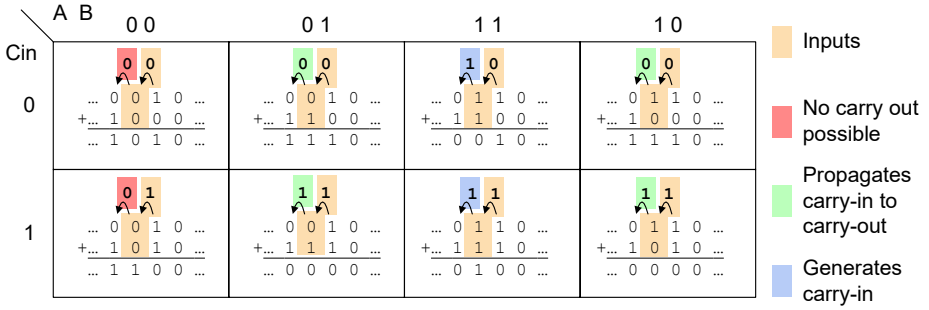
Arithmetic Circuits

Carry Look-ahead Adder (CLA)

The carry-look-ahead adder (CLA) overcomes the speed limitations of the RCA by using a different circuit to determine carry information. The CLA uses a simpler bit-slice module, and all carry-forming logic is placed in a separate circuit called the Carry Propagate/Generate (CPG) circuit. The CPG circuit receives carry-forming outputs from all bit-slices in parallel (i.e., at the same time), and forms all carry- in signals to all bit-slices at the same time. Since all carry signals for all bit positions are determined at the same time, addition results are generated much faster.

Since a CLA also deals with signals grouped as binary numbers, the bit slice approach is again indicated. Our goal is to re-examine binary number addition to identify how and where carry information is generated and propagated, and then to exploit that new knowledge in an improved circuit.

The figure below shows the same eight addition cases as were presented in the first figure. Note that in just two of the cases (3 and 7), a carry out is generated. Also note that in four cases, a carry that was previously generated will propagate through the current pair of bits, asserting a carry out even though the current bits by themselves would not have created a carry.



Truth table for carry generation.

Based on these observations, we can define two intermediate signals related to the carry out: carry generate (or G); and carry propagate (or P). G is asserted whenever a new carry is generated by the current set of inputs (i.e., when both operand inputs are a '1'), and P is asserted whenever a previously generated carry will be propagated through the current pair of bits (whenever either operand is a '1'). Based on this discussion, a truth table for the CLA bit-slice module can be completed (and you are asked to do so in the exercises).

The CLA bit-slice module generates the P and G outputs instead of a carry out bit. Note that a carry-out from the i^{th} stage in an Ripple Carry Adder (RCA) can be written as $C_{i+1} = C_i \cdot P_i + G_i$ in the CLA.

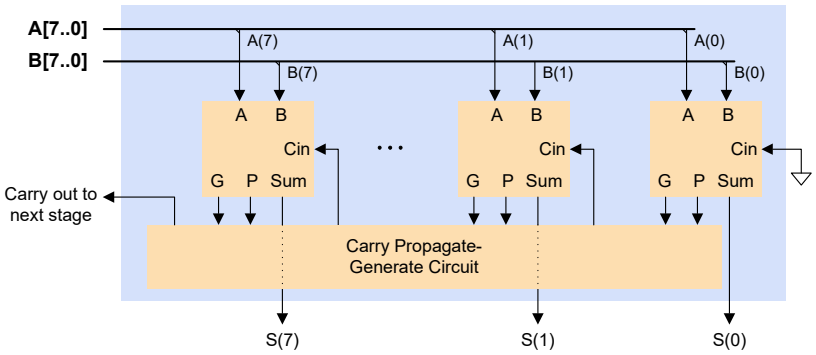
Since the carry-ins to each bit-slice in a CLA arise from the carry out (in terms of P and G) from the previous stage, the carry-ins to each stage can be written as:

$\mathbf{0}^{th}Stage$	$C_{in} = C_0$
$\mathbf{1}^{st}Stage$ $i = 0$	$C_1 = C_0 \cdot P_0 + G_0$
$\mathbf{2}^{nd}Stage$ $i = 1$	$C_2 = C_1 \cdot P_1 + G_1 = (C_0 \cdot P_0 + G_0) \cdot P_1 + G_1 = C_0 \cdot P_0 \cdot P_1 + G_0 \cdot P_1 + G_1$
$\mathbf{3}^{rd}Stage$ $i = 2$	$C_3 = C_2 \cdot P_2 + G_2 = ((C_0 \cdot P_0 + G_0) \cdot P_1 + G_1) \cdot P_2 + G_2$ $+ = C_0 \cdot P_0 \cdot P_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + G_1 \cdot P_2 + G_2$
$\mathbf{4}^{th}Stage$ $i = 3$	etc. the equations can be expanded to any number of i

Restated in written form, carry-ins to the first few CLA bit-slices are formed as follows:

$\mathbf{0}^{th}Stage$	C_{in} **is simply connected to the overall, global carry in (called C_0).**
$\mathbf{1}^{st}Stage$ $i = 0$	C_{in} is '1' if a carry is generated in stage 0, or if a carry is propagated in stage 0 and C_0 is '1'
$\mathbf{2}^{nd}Stage$ $i = 1$	C_{in} is '1' if a carry is generated in stage 1, or if a carry is propagated in stage 1 and generated in stage 0, or if a carry is propagated in stages 1 and 0 and C_0 is a '1'
$\mathbf{3}^{rd}Stage$ $i = 2$	C_{in} is '1' if a carry is generated in stage 2, or if a carry is propagated in stage 2 and generated in stage 1, or if a carry is propagated in stages 2 and 1 and generated in stage 0, or if a carry is propagated in stages 2, 1, and 0 and C_0 was a '1'
$\mathbf{4}^{th}Stage$ $i = 3$	The pattern continues for any number of stages.

The carry-in logic equations for each stage are implemented in the CPG circuit block shown below. A complete CLA adder requires the CLA bit- slice modules and the CGP circuit. This complete CLA circuit involves a bit more design work than the RCA, but because the CGP circuit drives the carry-in of each CLA bit-slice module in parallel, it avoids the excessive delays associated with the RCA.



Block diagram of a carry look-ahead adder (CLA)

Example: Designing a 4-bit Binary Adder Structurally and Behaviorally

Implement the Adder Structurally

Full Adder (FA)

Create a Verilog module for a full adder.

```
module FA(
    input A,
    input B,
    input Cin,
    output S,
    output Cout
    );

assign S = A ^ B ^ Cin;
assign Cout = (A & B) | ((A ^ B) & Cin);

endmodule
```

Half Adder (HA)

Create another Verilog module for a half adder.

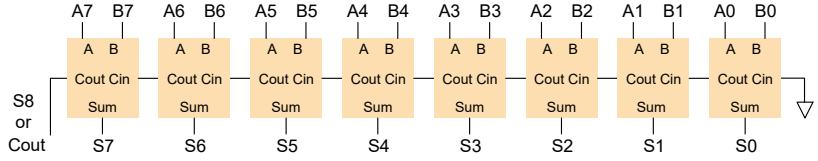
```
module HA(
    input A,
    input B,
    output S,
    output Cout
    );

assign S = A ^ B;
assign Cout = A & B;

endmodule
```

Implement 4-bit Adder using FA and HA

Create a Verilog module called adder to wrap three FAs with one HA to form a 4-bit adder based on figure 7 (PS! This example only uses 4 bits wherease the figure represents 8 bits). Check your wrapper code below after you're done by clicking "Show Code".



Ripple-Carry Adder Block Diagram

```
module adder(
    input [3:0] A,
    input [3:0] B,
    output [3:0] S,
    output Cout
    );

wire [3:0] Carry;

HA add_0 (
    .A(A[0]),
    .B(B[0]),
    .S(S[0]),
    .Cout(Carry[0])
    );

FA add_1 (
    .A(A[1]),
    .B(B[1]),
    .Cin(Carry[0]),
    .S(S[1]),
    .Cout(Carry[1])
    );

FA add_2 (
    .A(A[2]),
    .B(B[2]),
    .Cin(Carry[1]),
    .S(S[2]),
    .Cout(Carry[2])
    );

FA add_3 (
    .A(A[3]),
    .B(B[3]),
    .Cin(Carry[2]),
    .S(S[3]),
    .Cout(Carry[3])
    );

assign Cout = Carry[3];

endmodule
```